Pós-Graduação em Ciência da Computação

“Social Machines: A Unified Paradigm to Describe, Design and Implement Emerging Social Systems”

Por

Vanilson André de Arruda Burégio

Tese de Doutorado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE/2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

VANILSON ANDRÉ DE ARRUDA BURÉGIO

“Social Machines: A Unified Paradigm to Describe, Design and Implement Emerging Social Systems"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE PH.D. EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Silvio Romero de Lemos Meira

COORIENTADOR: Nelson Souto Rosa

RECIFE/2014

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da Silva, CRB4-1217

B952s Burégio, Vanilson André de Arruda
Social machines: a unified paradigm to describe, design and implement emerging social systems / Vanilson André de Arruda Burégio. – Recife: O Autor, 2014.
187 f.: il., fig., tab.

Orientador: Silvio Romero de Lemos Meira.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2014.
Inclui referências e apêndices.

1. Engenharia de software. 2. Arquitetura de software. 3. Redes sociais. I. Meira, Silvio Romero de Lemos (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2015-04

Tese de Doutorado apresentada por **Vanilson André de Arruda Burégio** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Social Machines: A Unified Paradigm to Describe, Design and Implement Emerging Social Systems**" orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. André Luis de Medeiros Santos
Centro de Informática / UFPE

_____

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____

Prof. Vinicius Cardoso Garcia
Centro de Informática / UFPE

_____

Prof. Zakaria Maamar
College of Information Technology / Zayed University

_____

Prof. Gibeon Soares de Aquino Junior
Departamento de Informática e Matemática Aplicada/UFRN

Visto e permitida a impressão.
Recife, 10 de junho de 2014.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Acknowledgments

A Deus, por ter guiado os meus caminhos e dado paz e tranqüilidade nos momentos mais difíceis. Aos meus pais, José Vanilson Burégio e Maria Luiza Burégio, e irmãos, Ana Carla Burégio e Luís Roberto Burégio, por terem me estimulado e apoiado nessa nova jornada.

À minha esposa e companheira, Vívian Damasceno, que sempre demonstrou amor, companheirismo e me deu muito incentivo, para que eu conseguisse concluir esse projeto.

Ao meu orientador, Silvio Meira, por ter me guiado sempre com muita paciência, dedicação, amizade e conversas extremamente inspiradoras. Obrigado pela oportunidade e por me fazer acreditar que tudo seria possível.

A Nelson Rosa, um novo amigo e co-orientador que eu conheci nesta jornada, sempre prestativo, me fazendo questionamentos e atendendo minhas solicitações com muita dedicação. Obrigado pelos preciosos conselhos.

A todos os professores do Centro de Informática da UFPE com os quais pude aprender bastante, em especial ao professor Vinícius Garcia e ao seu grupo de pesquisa por proporcionar ótimas discussões e trocas de ideias.

Aos meus amigos que precisaram compreender a necessidade da minha ausência ao longo desses anos para concluir um dos mais importantes projetos da minha vida profissional e pessoal. A todos os amigos do SERPRO que me deram força ao longo dessa caminhada e torceram pelo meu sucesso.

Enfim, um muito obrigado a todos que direta ou indiretamente colaboraram para o resultado desse projeto.

Resumo

A abordagem aberta e distribuída da Web, bem como a prevalência de relacionamentos entre aplicações e serviços estão transformando tanto a forma como desenvolvemos software quanto como eles funcionam e interagem uns com os outros. Como resultado, uma nova geração de aplicações está emergindo e consequentemente novos modelos mentais se fazem necessários para lidar elas. Neste contexto, Máquinas Sociais aparecem como um modelo promissor para o desenvolvimento de software. Entretanto, é um tema novo, com conceitos e definições provenientes de diferentes campos de pesquisa, o que torna o entendimento unificado do conceito um esforço desafiador. Nesta tese, nós fornecemos uma base conceitual mais coerente para entender máquinas sociais como um paradigma unificado para descrever, projetar e implementar aplicações e serviços sociais emergentes. Para isso, primeiramente revisitamos o conceito de relacionamento e estendemos a noção de máquinas sociais como um modelo de abstração comum a ser utilizado para fundir elementos computacionais e sociais em software. Em segundo lugar, para descrever máquinas sociais, apresentamos diretrizes para a análise que abordam algumas questões relacionadas com o exercício de engenharia de sistemas existentes. Em terceiro lugar, definimos a *Social Machine-oriented Architecture* (SoMAr) - um estilo arquitetural híbrido para projetar máquinas sociais através da combinação de diferentes princípios da prática atual da engenharia de software. Por fim, discutimos as experiências e lições aprendidas com a aplicação do paradigma de máquinas sociais em diferentes contextos.

Palavras-chave: Máquinas Sociais. Engenharia de Software. Arquitetura de Software. Sistemas Sociais. Sistemas Orientados a Web.

Abstract

The open, distributed approach of the Web and the relationship's prevalence of applications and services are transforming both the way we develop software and how they operate and interact with each other. As a result, a novel breed of applications is emerging, and consequently new mental models are needed to deal with them. In this context, Social Machines appear as a promising model for developing software. However, it is a fresh topic, with concepts and definitions coming from different research fields, making a unified understanding of the concept a somewhat challenging endeavor. In this thesis we provide a more coherent conceptual basis for understanding Social Machines as a unified paradigm to describe, design and implement emerging social applications and services. To do that, we revisited the concept of relationship and extend the notion of Social Machines to establish a common abstraction model that is used for blending computational and social elements into software. Second, to describe social machines, this proposal presents an analysis guideline that addresses some issues related to the engineering exercise of existing systems. Third, we provide the Social Machine-oriented Architecture (SoMAr) - a hybrid style to design social machines through the combination of different principles from current software engineering practice. Finally, we discuss the experiences and lessons learned from applying the social machines paradigm in different contexts.

Keywords: Social Machines. Software Engineering. Software Architecture. Social Systems. Web-oriented Systems.

List of Figures

List of Tables

Contents

Introduction

"Successive transition from one paradigm to another via revolution is the usual developmental pattern of mature science."

Thomas S. Kuhn (1922 – 1996)
Philosopher of Science

In his classic book *"The Structure of Scientific Revolutions"* (KUHN, 1970), Thomas S. Kuhn, considered by many to be the father of paradigms, suggests that scientific progress is a process of "*paradigm shift*". Basically, he claims that researchers in a branch of science accept as normal a set of "established beliefs" that conduct and limit their investigations into new phenomena. Because of this set of accepted beliefs and assumptions, new ways of looking at the world are often suppressed or ignored. These facts also take place in the context of software. When software paradigms evolve, revolutionary challenges and opportunities emerge in the theory and practice of software development. Then, inspired by Kuhn's thoughts, from time to time it is important to look back and see the progress of software engineering over the years in order to be prepared and to better understand how to anticipate and turn our attention to the next possible "*paradigm shift*". Excited by such idea, this chapter summarizes the motivation and objectives of this thesis and highlights the structure of the document.

1.1. Motivation

In the last decades, the classic notion of software has been changing in a significant way. From the seminal definition of computing machine specified by Turing (TURING, 1936) to the present, software started to become part of our daily lives and has been turned pervasive and ubiquitous with the introduction of personal computers, the Internet, smartphones and, later, the Internet of

Things (IoT). In fact, one can say that software and the internet have changed the way we communicate and the way business is done. Indeed, the internet is even now changing the way software is developed, deployed, and used. Recently, computing means connecting (ROUSH, 2005). Therefore it is possible that developing software is now as simple as connecting existing services.

The early internet was a Web of mostly static content, basically HTML pages presented in a read-only mode or possibly systems with a very simple transactional capability from the user's point of view. This is the Web we could classify as "1.0" (NATH; DHAR; BASISHTHA, 2014). As a further development, simultaneously with the appearance of new technologies, Web pages became more interactive and allowed content sharing, social interaction and collaboration, which led to blogs, wikis and social networks. This is the read/write Web, which is also known as Web "2.0" (MURUGESAN, 2007).

Recently, a new phase has been emerging, the Web "3.0" (PATTAL; LI; ZENG, 2009)(NATH; DHAR; BASISHTHA, 2014), the Web as a programming platform (BENIOFF, 2008) and networks as infrastructures for innovation. As a result, anyone and everyone can start developing, deploying and providing information services using the infrastructures for computing, communicating, and controlling in a manner not unlike utilities, such as electricity.

The Web 3.0 is the networked space-time where innovation lies on the power of developing software for the Web, through the Web, and in the Web, using the Web as both programming platform (in lieu of the usual computer/operating system/development environment platform) and deployment and execution environment. Several examples of this scenario are current developments in *Facebook*, *Twitter*, *Yahoo!*, *Salesforce*, *Google*, *Amazon* and many other corporations that are making their APIs available for anyone to develop applications that interact with their services.

Although there have been many studies about the future of the internet and concepts such as Web 3.0, programmable Web (YU; WOODARD, 2009) (HWANG; ALTMANN; KIM, 2009), linked data (BIZER; HEATH; BERNERS-LEE, 2009) (HALB; RAIMOND; HAUSENBLAS, 2008) and semantic Web (HITZLER; KRÖTZSCH; RUDOLPH, 2009), the segmentation of data and the issues regarding the communication among systems obfuscates the

interpretation of this future. Kevin Kelly, of Wired fame, is quoted as having said once:

> *"The internet is the most reliable machine ever made. It's made from imperfect, unreliable parts, connected together, to make the most reliable thing we have".*

Unstructured data, unreliable parts and problematic, non-scalable protocols are all native characteristics of the internet that has been evolving for 40 years; at the same time, they are the good, the bad and the ugly of a Web in which we rely more and more in the everyday life of everything, that needs a unifying view and explanations in order to be developed, deployed and used in a more efficient and effective way.

Indeed, the Web is changing in a fundamental way and approaches such as *SOA*(ERL, 2007), *REST*(FIELDING, 2000), *XaaS*(HAZRA, 2009) and *Cloud Computing*(HAYES, 2008) play important roles in this emerging Web. Nowadays, the Web is experiencing a new wave of applications associated with the proliferation of social networks and the growth of relationship's prevalence among people, applications and services. As a result, almost "everything" is getting social and a novel breed of *socially connected applications* is emerging, what led us to think about new software abstractions to deal with them. In this context, *Social Machines* has appeared as a promising option for blending computational and social aspects into software. "Social Machine" is a term firstly introduced by Tim Berners-Lee (BERNERS-LEE, 1999) in his book, "Weaving the Web", in which he states that:

> *"Real life is and must be full of all kinds of social constraint – the very processes from which society arises. Computers can help if we use them to create abstract social machines on the Web: processes in which the people do the creative work and the machine does the administration ..."*

Recently, the term has gained momentum and used to designate lots of different social-technical systems enabled on the Web, ranging from social networks to crowdsourcing and collaborative platforms of personal APIs and socially connected objects. Thus, as a research area, it is a fresh topic, with concepts and definitions coming from different research fields, making a unified understanding of the concept a somewhat challenging endeavor.

Social Machines are recent enough to represent very serious difficulties in understanding their basic elements and how they can be efficiently combined to develop real, practical systems in either personal, social or enterprise contexts. There has not been a clear, precise description of each and every entity on this new rising Web and we believe it is necessary to create new mental models of such a Web as a platform, in order to better understand this young, upcoming and possibly highly innovative phase of software development.

In practice, interested researchers and practitioners often raise several questions when they hear about Social Machines. *What do Social Machines really mean? How to describe them? Which are their main elements? What are the principles that guide the analysis and design of Social Machines? How to implement them?*

Social Machines are indeed in their infancy, experiencing a number of new emerging trends and visions. As this paradigm matures, we expect that any entity, be it software, a person or object, can be socially connected with each other, making relationships and determining different levels of interactions. Social Machines will eventually become a simple, formal and unified manner to explain networks of social systems; an informational paradigm to deal with the complexity of this new emerging Web around us, and a practical way to explain each and every entity connected to it.

1.2. Thesis Statement and Methodology

Motivated by the aforementioned issues and thoughts, this thesis aims to **provide a more common and coherent conceptual basis for understanding Social Machines as a unified paradigm to describe, design and implement emerging social systems**.

In order to achieve this aim, we defined a set of *goals, questions* and *evidences* that guided our research roadmap as a whole. In this case, "*evidences*" represent acceptable outcomes (e.g., an artifact, model, taxonomy, etc.) obtained via the process of answering the established questions. Figure 1.1 shows the final result from this process which, in a general way, helped us to determine a 4-stage roadmap for our research effort, as follows:

1. **Understanding**. First of all, we aimed at providing a *common conceptual basis of understanding* which relates the main concepts and existing approaches in order to categorize the types of systems as well as map the different existing views of social machines;
2. **Describing**. Our second stage consisted of explaining the "world" though the lens of the proposed paradigm by *describing existing* systems under a unified perspective of social machines;
3. **Designing**. This stage involved the design of systems as social machines. Here we defined an architectural style with general principles, constraints and guidelines to the design of social machine-oriented architectures;

4. **Implementing**. Finally, this stage consisted of experiences and evaluation of the proposed paradigm through the implementation of systems within different contexts.

The 4-stage roadmap in Figure 1.1 was used during the development of this work as a reference guide to the research effort. In order to reach each established goal we used specific research methodologies, which are explained along this document, such as *mapping study* (BUDGEN et al., 2007), *brainstorming and focus groups* (SINGER; SIM; LETHBRIDGE, 2008), *survey with experts*, *case studies* (HOST; RUNESON, 2007) and others.

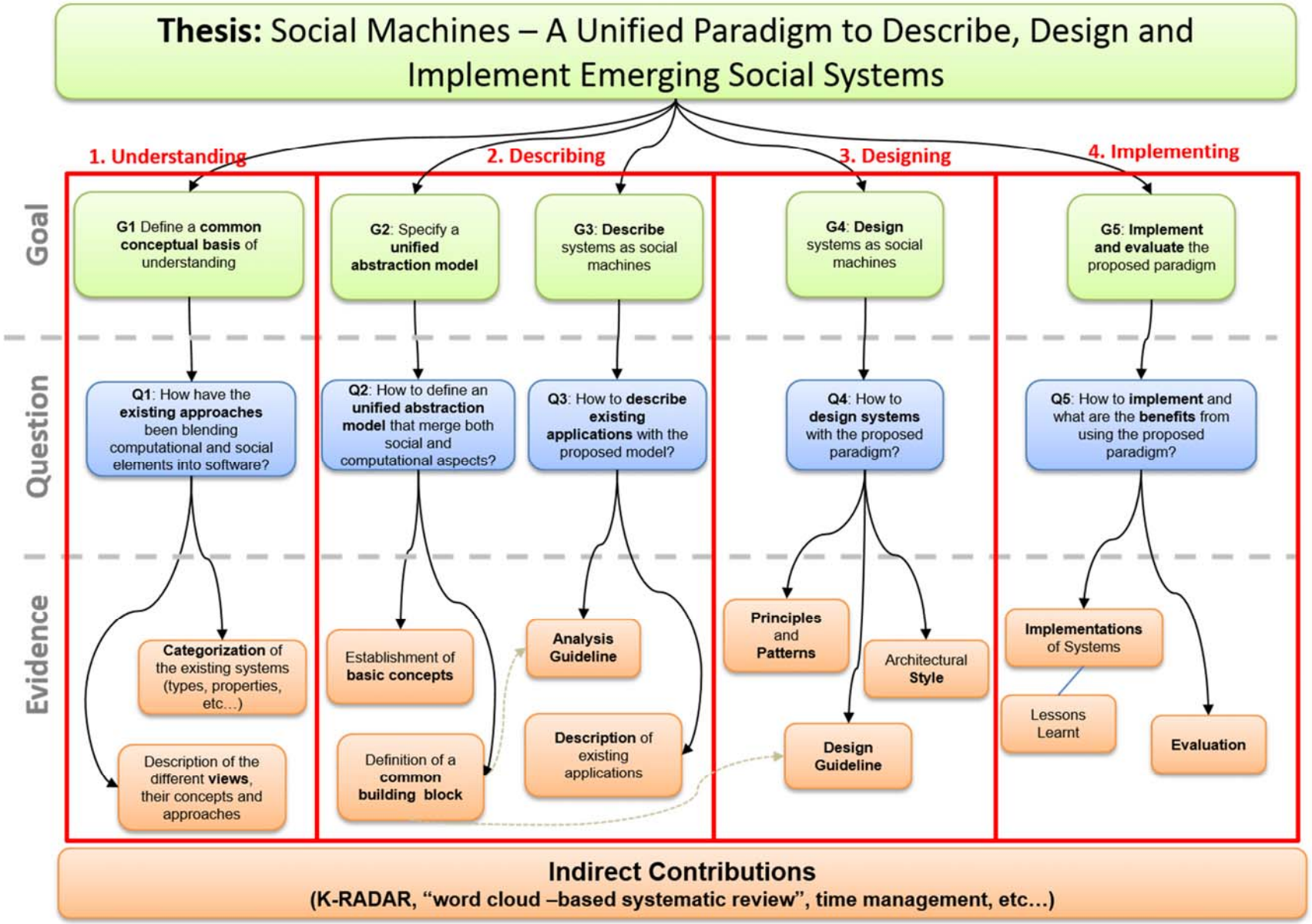

Figure 1.1 – 4-stage roadmap to guide the whole research effort

1.3. Proposed Solution Overview

The main evidences obtained from this 4-stage process can be seen in our proposed solution overview, illustrated in Figure 1.2. As shown, on top of a *common base of understanding*, we proposed the idea of *sociable software* and also revisited the concept of *relationship*. These [re]definitions of basic concepts were fundamental to extend the notion of Social Machines through the establishment of a *unified abstraction model* that was used for blending computational and social elements into software. Then, to describe social machines with such model, we provided an *analysis guideline* to address some issues related to the engineering exercise of existing systems. On top of the unified abstraction model, we also defined the *Social Machine-oriented Architecture* (SoMAr) - a hybrid style used to design social machines through the combination of different principles and constraints from current software engineering practice. A *design guideline* is specified as well, and the implementations of some systems are proposed with the aim of discussing the experiences and lessons learned from applying the social machines paradigm in different contexts.

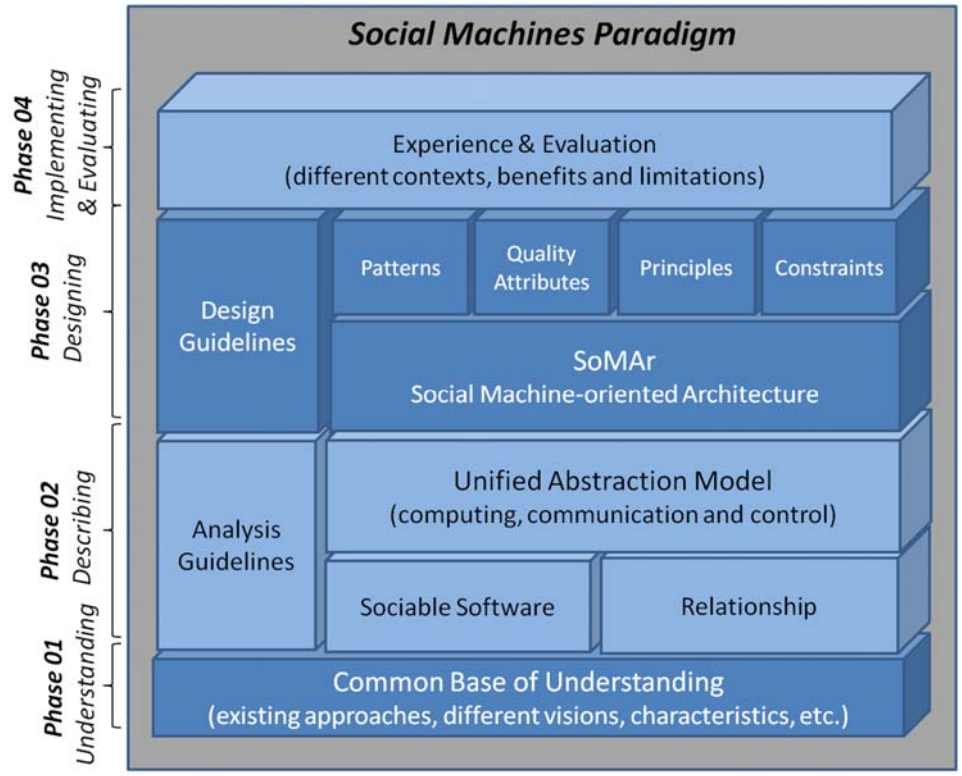

Figure 1.2- Proposed Solution Overview

1.4. Statement of Contributions

As a result of the work presented in this thesis, the following contributions can be enumerated:

- A study of the key developments in the field of Social Machines, in order to analyze this research area and identify the main visions, concepts, and approaches;
- A common and coherent conceptual basis for understanding Social Machines as a paradigm;
- The definition of a classification scheme to characterize the different kinds of existing Social Machines;
- The establishment of the concept of sociable and "relationship-aware" software;
- The definition of a unified building block to describe Social Machines as *socially connected computing units*;
- An analysis guideline for supporting the description of systems as Social Machines;
- The establishment of Social Machine-oriented Architecture (SoMAr) as a hybrid architectural style that defines a set of constraints, principles and desired properties;
- A design guideline for supporting the creation of Social Machine-oriented architectures;
- A reference architecture that generalizes a set of adopted solutions and defines a higher level template that can be used as reference for instantiating specific Social Machines;
- An analysis of existing systems (e.g., Facebook, Twitter, Dropbox) as relationship-aware Social Machines;
- A discussion on the design and implementations of systems following the Social Machines paradigm in different scenarios;
- An analysis of practical developments of Social Machines in both academic and industrial context;

Indirect contributions

- The adaptation of a mapping study process by including word cloud to support the definition of classification schemes;
- The establishment of the *Knowledge Radar* (K-RADAR) - a monitoring approach for measuring research progress (Appendix A);

1.4.1. Actual Status of Publications

As a partial result related to the work presented in this thesis, the following direct and indirect results can be enumerated:

- **IEEE IC Magazine:** V. A. Burégio, Z. Maamar, and S. L. Meira. *An Architecture and Guiding Framework for the Social Enterprise.* IEEE Internet Computing Magazine, 2015

- Maamar, Z., Buregio, V.A., Rosa, N.S.: *From Business Artifacts to Social Artifacts.* (To be published), 2015.

- ***WWW 2014:*** Buregio, V., Nascimento, L., Rosa, N., Meira, S.: "*Personal APIs as an Enabler for Designing and Implementing People as Social Machine*", in the proceedings of the 23rd International World Wide Web Conference Companion (WWW'14 Companion), Seoul, Korea, pp. 867–872, 2014.

- ***WWW 2014:*** Nascimento, L., Buregio, V., Garcia, V., Meira, S.: "*A New Architecture Description Language for Social Machines*", in the proceedings of the *23rd International World Wide Web Conference Companion* (*WWW'14 Companion*), Seoul, Korea, 2014

- ***EDOCW 2013:*** 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 (EDOCW2013), Vancouver, Canada, pp. 43–52, 2013.

- ***WWW 2013:*** 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, 2013.

- ***Open Session at sociam 2013***: presentation in the open session of the *First Internatinal Workshop on the Theory and Practice of Social* Machines, http://sociam.org/www2013/;

- **Academia.edu 2013**: Meira, S., Burégio V.A., Nascimento L., Araújo S., "*On the Internet, Privacy and the Need for a New Architecture of Networked Information Services*", 2013.

- ***SEKE 2012:*** K. S. Brito, L. E. Abadie, P. F. Muniz, L. Marques, V.A. de A. Buregio, C. Vinicius, and S. Meira, "*Implementing Web Applications as Social Machines Composition: a CaseStudy*," The 24th International Conference on Software Engineering and Knowledge Engineering, vol. (SEKE'2012), pp. 311–314, 2012

- ***COMPSAC 2011:*** Meira S., Buregio V. A., Nascimento L. M., Figueiredo E., Neto M., Encarnacao B., and Garcia V. C., "*The Emerging Web of Social Machines," in 2011 IEEE 35th Annual Computer Software and Applications Conference, 2011.*

- **Seminal Paper (2010):** S. R. L. Meira, V. A. A. Buregio, L. M. Nascimento, E. G.M. de Figueiredo, M. Neto, B. P. Encarnação, and V. Garcia,"*The Emerging Web of Social Machines*", Cornell University Library, vol.abs/1010.3, Oct. 2010

- **SPLC 2010:** Buregio, V. A. A. ; Almeida, E. ; Meira, S. R. L. *Characterizing Dynamic Software Product Lines: A Preliminary Mapping Study*. Proceedings of the 14th Software Product Lines, 2010. v. 2. p. 53-60, 2010, Jeju Island.

1.5. Document's Structure

The remaining chapters of this proposal are organized as follows.

- **Chapter 2** presents a brief history about the Web's evolution and conducts an investigation on the popular abstraction models for developing and designing software;
- **Chapter 3** presents the results of a mapping study performed on related topics in order to build a classification scheme that structures the science of Social Machines;
- **Chapter 4** revisits the concept of *relationship* and define a common abstraction model that is used as basis for establishing *Social Machines* as a unified paradigm to describe the emerging Web-oriented systems;

- **Chapter 5** presents and discusses the ***So****cial* ***M****achine-oriented* ***Ar****chitecture* (SoMAr) - an architectural style that serves as an abstracting framework for guiding the design of Social Machines;
- **Chapter 6** describes the experiences and evaluation from applying Social Machines into different contexts;
- **Chapter 7** concludes the thesis, discussing our contribution and offering directions for the next steps.

A Brief History of the Web and Software Abstractions

"A Master is not he who always teaches, but he who suddenly learns..."

João Guimarães Rosa (1908 – 1967)
Brazilian novelist

This chapter provides a brief history about the Web's evolution and conducts an investigation on the popular abstraction models for developing and designing software. It categorizes the paradigms along the time and identifies their main characteristics. Based on the influences of the Web's evolution and the analysis of paradigms found, we propose some directions in order to evolve the state of the art of software development.

2.1. The Waves of the Web

The conceptual and technological foundations of the *World Wide Web*'s (aka 'The Web') were built during the first years of 90s. Since of then, a number of developments have significantly changed the Web and made it much bigger than Tim Berners-Lee's original idea of creating a *collaborative space* in which people could *communicate* through sharing information (BERNERS-LEE, 1996).

Recently, the emergence of *mashups* (YU et al., 2008), the popularization of Web-based systems providing *Open APIs* for third-party developers (CHEN et al., 2009; KIWON LEE, 2009; YE ZHOU; YANG JI, 2011) and other concepts, such as *Software as a Service* (TURNER; BUDGEN; BRERETON, 2003) and *Cloud Computing* (PATTERSON; FOX, 2012) have played an important role in the way the Web has been influential on software development. In fact, the Web has spread from a global system of interlinked hypertext documents to a platform for open, interactive, distributed applications and services (MAXIMILIEN; RANABAHU; GOMADAM, 2008).

As a consequence, the modern Web has transformed the way we think about software. Nowadays, several software products became "*services avatars*" (GRAY, 2012), what means that there is no need to "physically" install and configure them in local workstations, instead of that, they are provided and consumed as services on the Web. Lots of static websites have also evolved for dynamic and sophisticated systems and the Web has moved towards a *global programmable platform*.

With the aim of facilitating the study of the transformations that led to the current stage of maturity of Web applications and services, we can assume an adaptation of Marc Benioff's taxonomy (BENIOFF, 2008) by dividing the Web's history into three main waves, namely: *Read Only; Read/Write* and *Programmable*. We summarize such waves in Figure 2.1. As can be seen, these three waves are not defined by precise periods of time. Actually, they represent overlapping waves in the Web's history, each of them with its own set of features. Furthermore, these historical waves are not mutually exclusive. Thus, even today, it is possible to have the coexistence of applications and services in various stages of this Web's taxonomy.

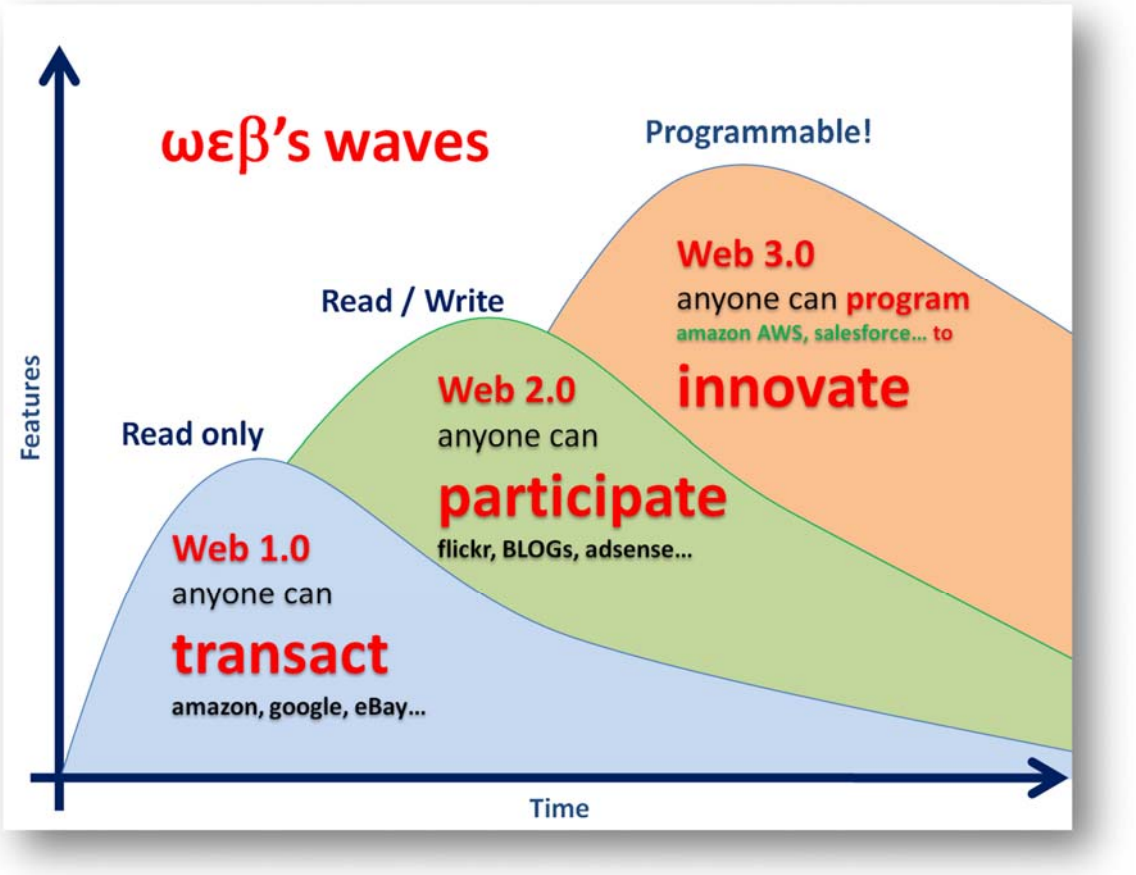

Figure 2.1 -The waves of the Web

2.1.1. The Read Only Web

The first wave of the Web is characterized by the *read-only websites*, which emerged in the 1990s. According to Benioff (BENIOFF, 2008), it is the *Web 1.0* in which anyone can read and transact. The read only Web comprises the first search engines, e-commerce services and other examples of Web applications that present to users the possibility to interact with the presented information, but without allowing them to update or add new content. Such *Web 1.0* applications allow little interaction and communication among users, given that their main goal is to provide some kind of information in a one-way direction.

This phase also includes applications that allow users to transact on the Web. Companies like *eBay*, *Amazon.com* and *Google* were the forerunners of launching Web-based systems in which anyone could make transactions of goods and knowledge as well (BENIOFF, 2008). Searching on *Google* and

making a purchase on *Amazon.com* are good examples of the kinds of transactions that emerged at that time. Even today the *Web 1.0* remains present in our lives and will continue to exist for some time.

2.1.2. The Read/Write Web

The second wave of the Web is characterized by applications and services that transformed the semantics of "content" on the Web: content is no more static at the point of publication; instead, it is dynamic and can be changed by users as we do, for example, in Wiki-based systems. Then, it is the Web in which anyone can participate. The centerpiece of this wave is featured by a culture of *community*, *participation*, *collaboration* and *co-creation* of content. Given the level of transformation enabled by these tools, Tim Oreilly call them as the "*Web 2.0*" (O'REILLY, 2005), using the term "2.0" to emphasize the evolving nature of these next generation of systems, similar to what we do with software versions.

The Web 2.0 features formed the basis to the construction of other interactive tools like online social networks, blogs and other services that have completely changed the way we communicate and share information with each other. Nowadays, online social networks such as *Facebook*, *Twitter*, *Google+*, *LinkedIn*, and *Foursquare* have become really popular all over the world and play a significant role in people's daily lives. *Facebook* is a proof of that, the world's biggest online social network of today, with more than one billion active users (LEE, 2012).

2.1.3. The Programmable Web

This is the Web 3.0, in which anyone can innovate. This is because the entire needed infrastructure to code, deploy and evolve a system is provided on the Web. The infrastructure of this programmable Web is supported by some interrelated concepts, such as *Cloud Computing*, *Utility Computing* and *XaaS* (Everything-as-a-service), which have risen to prominence recently by the emergence of services like *Amazon EC2*, *Google App Engine*, *Salesforce*, *Microsoft Azure* and so on. The main idea of these concepts, and therefore the services that implement them, is to enable the widely availability of

computational resources that can be accessed on demand to create and connect lots of systems.

Thus, the Web 3.0 is characterized by the progressive migration from the concept of a Web as a large-scale information system to a Web that is now emerging as a means of connecting distributed applications and services. Currently, more and more Web-based systems have published APIs that enable software developers to easily integrate data and services instead of building them from scratch. Hence, in the Web 3.0, the successful large-scale information system gives place to a platform for an ecosystem of connected people, applications, services and also "physical objects" (i.e., things) (AGRAWAL; DAS, 2011). These huge numbers of online connections among different entities have radically redefined the way we think about *connectivity*, including new forms of *relationships* and *interactions* that have also led us to significantly transform the way we develop software (BURÉGIO et al., 2013b).

Hence, in the context of software engineering, one question is how software and its abstractions have been changed to meet this new trend? The next section provides an overview of software evolution and conducts a brief investigation on the popular abstraction models for developing and designing software, presenting some directions in order to evolve the state of the art on software development.

2.2. Evolution of Software Abstractions

The abstraction notion is central to understanding the semantics of software. Abstracting can be defined by the process of understanding and expressing the world around us under the perspective of specific mental models. This process involves *questions* (as abstractions of *problems*) and *answers* (as abstractions of *solutions*).

During this work, we "philosophically" define software and its development in terms of abstractions, and our conclusion is that:

> *"Software is the **creative** and **evolutionary** abstraction of 'everything'; its development involves a process of abstractions which consists on making questions and providing answers about the real (virtual and concrete) world."*

We need to creatively make the correct questions in order to try to understand the (sometimes unknown) world around us. Furthermore, even more important, we should use *appropriate* mental models due to the dynamic evolution of such world. In fact, we indeed live in a constant changing environment and from time to time we need to look at the world around us under a different perspective that better fits the current reality. Occasionally these different thinking processes form the basis for creating new mental models and possibly the establishment of new paradigms.

Software Paradigms

There are many meanings associated with the term 'paradigm'. In a general way, it can be seen as an approach to something, a school of thought about something or a combined set of rules applied within a predefined scope. In the context of software engineering, a paradigm can be considered as an approach that governs the design of logic (ERL, 2007). It usually is accomplished by building upon an abstraction model which is used as building blocks for the development of software. The object orientation is a typical example of an accepted paradigm of software development. It defines a *common abstraction model* (i.e., object) and provides a set of principles that drives the analysis and design of solutions in a way that is possible to achieve specific goals.

With the rapid increase in complexity of software systems and technology, the models for developing and integrating software evolve as well. In Figure 2.2, we highlight some popular software paradigms. It is important to note that we do not have the intention to make a comprehensive list of all programming paradigms, but instead summarize the development of the practice of software engineering along the time via what, in the diagram, we call "era".

As can be seen in Figure 2.2, the evolution of software paradigms has passed through various eras, including structured programming, object-oriented era, component-based development, service-oriented era, resource-oriented era and, more recently, the social era.

2.2.1. Structured Era

The "boom" of the era of structured development can be placed at the 1970s. It represents a software paradigm in which there is a clear view of *code* operating on *data*. It permits us to abstract and express the essentials of an algorithm and, therefore, to partition a program into meaningful manageable units (e.g., *subroutines*, *block structures*) with the aim of improving the clarity and quality of software.

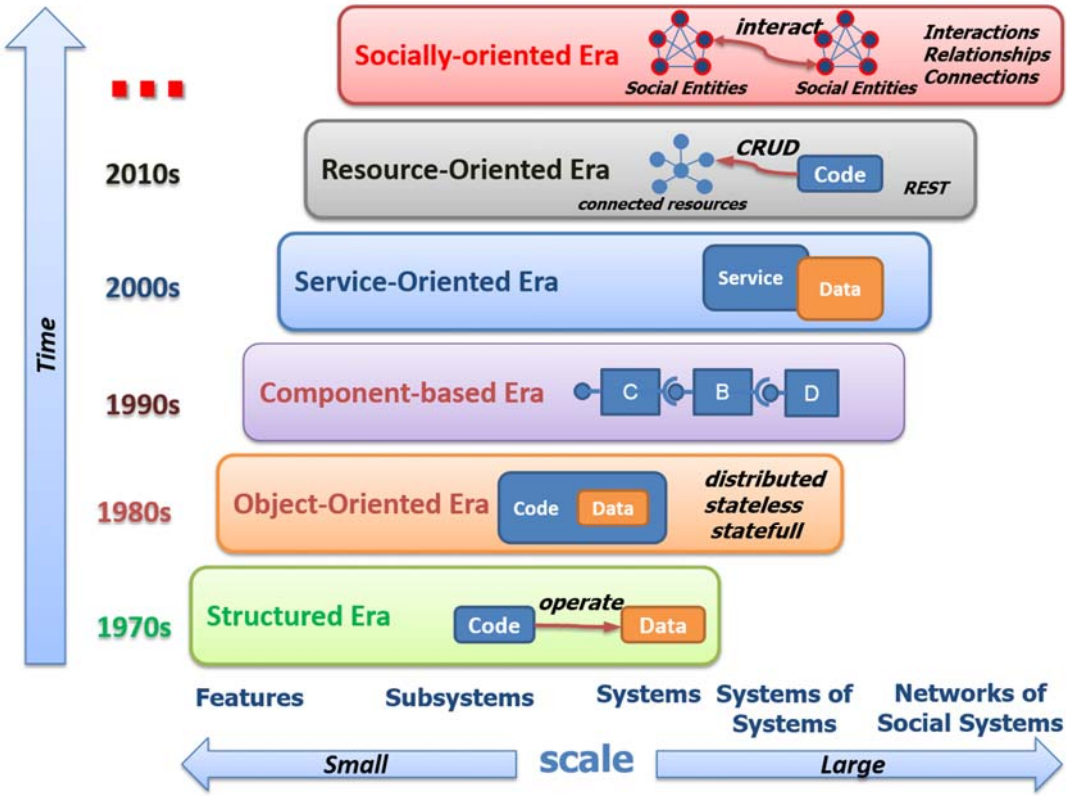

Figure 2.2 - Evolution of Software Paradigms

In such period, the practical concepts of software development were focused on structured coding (MCGOWAN, 1975), in which program logic is expressed in terms of *flow-of-control forms*, such as *SEQUENCE*, *IF-THEN-ELSE*, *WHILE-DO*, and other structures that continue present in the existing contemporary programming languages. This era also includes structured analysis and design, and some variations between process orientation (DEMARCO, 1979) and data orientation (YOURDON, 1988) until it finally brought us to the notion of object orientation, where data and functions come

together to form objects (classes), and create the basis of the well-known object-oriented paradigm.

2.2.2. Object-oriented Era

The Object-oriented (OO) paradigm became popular in the 1980s (TSAI; ZUALKERNAN, 1988) with the promises of improving productivity and reducing maintenance costs (SCHIEL; MISTRIK, 1990). It uses *objects* as its key abstraction for analysis and design of solutions. An *object* encapsulates data structures and exposes a set of methods that can be invoked to manipulate such data structures whose implementations are hidden from the invoker. During this period, lots of issues could be realized in the way objects were used to build systems (GOYAL, 1991).

Initially, *statefull* objects (i.e., objects that keep state) came up as a natural manner to model objects in the real world. However, the use of *stateless* objects (i.e., objects that not keep state) started to become more frequent due to the necessity of representing stateless functions (which do not retain any information from one method call to the next), and, after, influenced by the intrinsic *stateless* interaction model of the first Web-based systems (i.e., each request from a user for a Web page results in the requested pages being served, but without the server remembering the previous request). In this context, the necessity to create distributed objects also became evident (NICOL; WILKES; MANOLA, 1993), what created the basis for the introduction of *distributed software components*.

2.2.3. Component-based Era

Component-based software engineering, built on the foundation of object orientation, improved the state of the art by providing the idea of components as units of deployments (SZYPERSKI, 2002). The essence of componentization is to breakdown a system into reusable pieces of software that can be plugged into other software components with relatively little effort. Component-based technologies made a substantial impact on the design and engineering of many kinds of software systems (HEINEMAN; COUNCILL, 2001). With the premise of improving reuse, the techniques of component-based programming have *interoperability* as one of their goals.

The use of *components* as units of deployment progressively gave way to a protocol of remote invocation of those components over distributed network. Several standards for remote invocation of distributed components emerged at that time such as *CORBA*, *Microsoft Component Object Model* (COM) and *Java Remote Method Invocation* (RMI).

With the aim of bringing the flexible, open-standards-based, distributed computing to the Internet, *Web Services* (WS) emerged as a new breed of "application components" accessible over open protocols. In the Web service model, providers and consumers of services are separated by an interface (i.e., contract) that allows more flexibility for the underlying technology to be used. Lots of standards emerged under the umbrella of Web services such as, e.g., SOAP, WSDL and UDDI, facilitating the invocation of WS over a network. With the broad use of Web services, software development reaches its next stage of evolution: the *service-oriented era*.

2.2.4. Service-oriented Era

In this era, more and more organizations started to offer access to their information through Web services. By combining different Web services, it is possible to conduct business transactions and also create other value-added services or applications to provide functionalities that were not designed earlier (LANTHALER; GRANITZER; GUETL, 2010). In the enterprise context, Web services became the default implementation of service-oriented architectures (SOA) (ERL, 2005).

SOA is a software architecture style that defines the use of services as a reusable software component to automate business processes and consequently increase productivity. The concepts of SOA can be implemented using any service-based technologies. However in practice, most efforts on implementing SOA are focused on Web services that use SOAP (*Simple Object Access Protocol*) as the messaging protocol responsible for transferring data between different Web services (CHUNG, 2005).

Despite SOAP being built using XML and relying on common Internet transport protocol like HTTP to transport its messages, the usage of SOAP-based services is traditionally mainly appropriate to the integration of legacy

systems within and across enterprise environments, rather than in the open context of the Web. According to Lanthaler and Gütl (LANTHALER; GÜTL, 2010), one of the reasons for this is the high complexity of the SOAP stack on handling the serialization and transport of XML-encoded data.

Because of that, lightweight protocols like *ATOM* (SAYRE, 2005) and *RSS* (PREECHAVEERAKUL; KAEWNOPPARAT, 2009) to push data to consumers started to be used by major Web service providers like *Amazon*, *Google*, *Microsoft*, *Yahoo* and others. Such providers also began to expose their services as simple and lightweight REST-style APIs, and, in some cases, to replace their exiting SOAP-based services with REST services (DANIEL; PRZEMYSLAW; LARS, 2010), giving rise to what we call the *resource-oriented era*.

2.2.5. Resource-oriented Era

REST (*Representational State Transfer)* (FIELDING, 2000) has been growing in popularity since 2005, when it inspired the design of services like *Twitter API*. However, its massive use was consolidated in 2010 (almost ten years later of its definition) with a significant amount of attention from industry (WEBBER; PARASTATIDIS; ROBINSON, 2010) and academia (WS-REST, 2010). In spite of being based on the principles of service computing, the introduction of REST significantly changes the manner we abstract software solutions on the Web, and, for this reason, we consider it as a paradigmatic milestone in software development.

REST defines a novel architectural style and an alternative way for enabling services on the Web (aka, 'RESTful Web services') by using the standard Web infrastructure. It basically uses the existing HTTP methods to apply *CRUD* operations (*Create*, *Read*, *Update* and *Delete*) to *resources* (any entity on the Web) defined by their URI. The main advantages of REST are its uniform service interface and universality (DANIEL; PRZEMYSLAW; LARS, 2010) and also the fact that it improves system flexibility, scalability, and performance as compared to the SOAP-based Web services (UPADHYAYA et al., 2011).

Nevertheless, it addresses only basic distributed interaction/coordination (ISSARNY et al., 2011), leaving open some issues that have to be tackled in the context of the social Web ecosystem, such as dealing with the establishment of *relationships* in globally connected and interacting systems that offer real platforms of services such as, e.g., *Facebook* and *Twitter*, which is one of the aspects of the current *social era of software*.

2.2.6. Socially-oriented Era

We have never been so connected as nowadays. Actually, we are becoming more and more social. It means that we are not simply connected, but establishing different levels of *relationships* and *interactions* to share information with each other. The Web is the backbone of this "new" socially-oriented era, what suggests that “everything” connected to the Web is getting more social. We believe that we are in the beginning of an age in which almost “everything” will be socially connected. That is the case of people, software and things, all of them considered *social entities* in this age. More than one decade ago, some researchers such as Zakaria Maamar (MAAMAR, 2003) insisted on the fact that in the near future social aspects should be considered a significant element of any software development. Since then, his opinion has reinforced the thought that both people’s mentalities as well as software engineering practices need to be changed. More recently, and exactly one decade after Maamar’s claim, Tan et al. (TAN et al., 2013) support his thought by stating that:

> *“... Currently, most social networks connect people or groups who expose similar interests or features. In the near future, we expect that such networks will connect other entities, such as software components, Web-based services, data resources, and workflows.”*

In the context of the Web, *platforms of services* like *Twitter, Salesforce* and *Facebook* - with the ecosystem of connected people and third-party applications created around them - can be seen as current examples of software that represent this new socially-oriented era. Today, every product is a platform (SEMMELHACK, 2013) and we cannot build a product in this day and age without focusing on the open platform requirements, because lots of the best products are platforms by themselves; Google, YouTube, Facebook.

Twitter, for example, revealed that 75% of all their traffic is outside of twitter.com and comes from their platform's REST API (i.e., 3 billion calls every day, according to DuVander (DUVANDER, 2010)). This indicates that REST may continue to be the "foundational network protocol" of this era. However, there are other fundamental elements of these *social entities* that also have to be considered such as, e.g., the persistent *relationships* that exist among them (*Facebook's* platform, for example, establishes different relationships and interactions not only between its users, but also among its third-party applications). As a consequence, new mental models need to be created in order to better abstract these *social entities* and improve the practice of software development in this upcoming era that clearly needs to blend computational and social processes into single *socially connected units*.

2.3. Concluding remarks

The evolution of software in Computer Science is accomplished by building upon existing research to produce new and innovative solutions. During the history of software development there has been a continuous stream of innovations that have pushed forward the abstraction models used to improve software development practices. From *subroutines* in the 1970s to *objects* in the 1980s, *components* in the 1990s, *services* in the 2000s and *resources* in the 2010s, software development has been a story of continuous evolution and changing abstraction models used to better fit the current reality and deal with the increasing complexity of systems.

Nowadays, we are living in a "socially-oriented era" and there are no doubts that existing software development practices need to be revisited in preparation for the transition from the traditional (no-social) model to the social model of development. In this context, the Web has been influential in the way we develop software. It is becoming a more open, global, ubiquitous and pervasive platform for our society and world. Such "Social Web" requires considerable enhancements in software characteristics which challenge existing software paradigms, including abstraction models, architecture and engineering approaches. The expansion of software engineering research into the socially-oriented era is a natural evolution of the endeavor to understand how to wave social elements into software in order to systematically enable the development

of this new breed of social computing entities. In this thesis, *Social Machines* is presented as a possible option of these future directions. SMs são sistemas habilitados na Web e não devem ser comparados com ou vistos como uma fase independente da Web (Web 3.0, 4.0, etc). As diferentes características dessas fases têm habilitado diferentes SMs. As máquinas sociais emergentes, por exemplo, se beneficiam das novas formas de interação e relacionamentos para criar diferentes sistemas na Web que são governados por combinações de processos computacionais e sociais.

SMs are systems enabled on the Web and should not be compared with nor seen as a specific wave of the Web (i.e., Web 2.0, 3.0, etc). In practice, the different characteristics of these waves have enabled different SMs along the way. Emerging Social Machines, for example, benefit from novel forms of interactions and relationships to create new breed of systems on the Web which are driven by combinations of computational and social processes into software.

In the next Chapter, we present the results of a mapping study performed on efforts related to the topic of blending social and computational elements into software, having the aim of identifying existing approaches and understanding the main concepts that shape the *Social Machine* area.

3 Understanding Social Machines

"Life can only be understood backwards; but it must be lived forwards."

Søren Kierkegaard (1813 – 1855)
Danish philosopher

Blending computational and social elements into software has gained significant attention in key conferences and journals. In this context, "Social Machines" appears as a promising model for unifying both computational and social processes. However, it is a fresh topic, with concepts and definitions coming from different research fields, making a unified understanding of the concept a somewhat challenging endeavor.

This chapter aims to investigate efforts related to this topic and build a classification scheme to structure the science of Social Machines. We provide an overview of this research area through the identification of the main visions, concepts, and approaches; we additionally examine the result of the convergence of existing contributions. With the field still in its early stage, the first part of this work collaborates to the process of providing a more common and coherent conceptual basis for understanding Social Machines as a topic of research inquiry. Furthermore, this study helps detect important research issues and gaps in the area.

3.1. Introduction

We have discussed so far that the emergence of a new generation of Web-based technologies relying on social computing is changing the semantics of computation. Nowadays, more than ever, computing means connecting (ROUSH, 2005). In fact, the Social Web has fueled the growth of systems that not only make use of concepts from computing, but also are guided by social processes. As a consequence, novel breeds of applications are rapidly emerging and new computational models and paradigms are needed to deal with them.

Several studies (e.g., (SCHALL; TRUONG; DUSTDAR, 2008), (MAAMAR et al., 2009),(HENDLER; BERNERS-LEE, 2010), (MEIRA et al., 2011), (IAMNITCHI; BLACKBURN; KOURTELLIS, 2012), (THALER; SIMPERL; WÖLGER, 2012), (SHADBOLT, 2013),(KAJAN et al., 2014)) that adopt different visions have been conducted with the aim of creating innovative approaches to support the blending of computational and social elements into software. Consequently, these visions deal with the challenges of building this new generation of social systems. In this sense, the topic of "Social Machines" has been investigated as a way to address this challenge, appearing as a promising model for unifying both computational and social processes.

However, in spite of being a promising topic, the concepts behind Social Machines overlap different research fields and, consequently, have created confusion and raised several questions. For instance, we have found some researchers that have had difficulties in understanding the boundaries of this research topic and how it can contribute to their research fields.

Thus, with the intention of minimizing such problems, this chapter proposes to investigate existing efforts related to Social Machines and characterize such topic, systematically mapping foundational studies into a common and convergent classification scheme. As a result, we provide an initial overview of the research area, identifying the different visions of Social Machines as well as unifying them into a central idea within the field of computer science. Furthermore, this study provides a basis for the process of defining Social Machines as a paradigm.

The remainder of this chapter is organized as follows. Section 3.2 outlines the adopted research methodology. Section 3.3 shows the different visions of the "Social Machines" paradigm. Section 3.7 introduces some existing related reviews and, finally, Section 3.8 presents some concluding remarks and directions for the research.

3.2. Research Guidelines

By browsing the literature, an interested reader might experience difficulty in understanding what Social Machines really mean. In (HENDLER; BERNERS-LEE, 2010), Hendler and Berners-Lee suggest that social machines are systems that blend computational and social processes. Motivated by this position and some related works, the initial stage of this research adopts *Mapping Study* (BUDGEN et al., 2007) (PETERSEN et al., 2007) as the main research method to identify how existing efforts have been blending computational and social elements into software. Table 3.1 summarizes the research guidelines adopted in this first stage, which we refer to as “Understanding Social Machines”, as defined in our roadmap presented in Chapter 1 (Figure 1.2).

Table 3.1 - Research Guide for Stage 01: Understanding Social Machines

Stage 01: Understanding Social Machines	
Goal	Define a common conceptual basis of understanding
Question	How have the existing approaches been blending computational and social elements into software?
Research Method	Mapping Study;
Evidences	Categorization of existing systems; Description of different views, concepts and approaches.

Mapping Study is a helpful method used by the software engineering community with the aim of building a classification scheme and structure a software engineering field of interest (PETERSEN et al., 2007). In this chapter we present the results of an adaptation of this process whose main goal is to provide an overview of the Social Machines research area, focusing on the

different visions we identified during the mapping process. Next, some details of the adopted process are presented.

3.2.1. The adopted mapping process in a nutshell

The experimental software engineering community has been working towards the definition of standard processes for conducting mapping studies (MS). An example of this processes can be seen in (PETERSEN et al., 2007), which describes how to conduct mapping studies in software engineering. Petersen et al.'s process steps include *definition of research question, conducting the search for relevant papers, screening of papers, keywording of abstracts, data extraction and mapping*.

In our case, we merged the process defined by Petersen et al. (PETERSEN et al., 2007) with a *word cloud-based approach* (MCNAUGHT; LAM, 2010) that was used to automate the step of *keywording* (see Figure 3.1). Beyond that, in order to improve the final obtained mapping, we gathered experts' opinion though conducting, for example, *live online polls* during presentations at conferences and specific forums of discussion. Figure 3.1 shows the outcome of each process step as well as the adaptations we made.

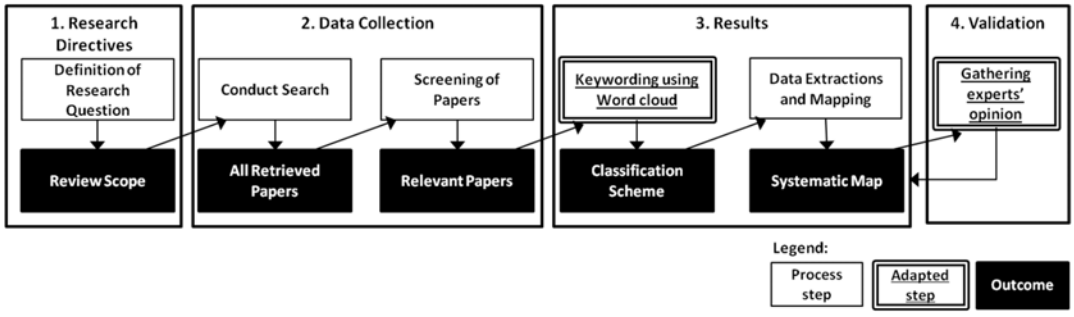

Figure 3.1 - The Mapping Process (adapted from (PETERSEN et al., 2007)**).**

3.2.2. Research directives (research scope)

An essential step of any mapping study is the definition of its research question. This question is responsible for scoping the review and driving all the subsequent steps of the process. Therefore, this mapping study is focus on answering the following question:

How have the existing approaches been blending computational and social elements into software?

This question aims to identify the main social machine-related topic and how representative such topics are, in order to outline the area and form the basis to build a classification scheme of the main existing visions, types of systems and approaches.

3.2.3. Data Collection

Search Strategy

We defined a search strings to be used for searching articles on scientific databases. The search strings were derived from the aforementioned research question and constructed using Boolean ANDs and ORs operators in combination with relevant keywords such as the following instance:

("Social" OR "Human" OR "People" OR "Crowd")
AND
("Computing" OR "Computation" OR "Computational")
AND
("Machine" OR "Unit" OR "Service" OR "Process" OR "System")
AND
"Web"

The possible sources of articles included journals, conferences proceedings, books, theses and technical reports. Furthermore, some initiatives in the industry and practical tools were also considered in the review, and relevant researchers were also directly contacted, as a way to get more information or request some specific material of interest.

Screening of papers for Inclusion and Exclusion

Inclusion and exclusion criteria need to be used in order to discard studies that are not relevant to answer the research question. Accordingly to Budgen et al. (BUDGEN et al., 2007), it is helpful to exclude articles that, for example, only point out the focus of the mapping study in the abstracts without developing it through the document, i.e., misleading abstracts. In this study, we adopted the following selection criteria:

1) Inclusion criteria: The abstract explicitly mentions the blending of social and computing elements in the context of Web-based systems. From the abstract, the researcher is able to infer that the focus of the paper contributes to the social machine area.

2) Exclusion criteria: The article lies outside the software engineering and/or Web domain. Social elements are not part of the contributions of the paper and social related terms are only mentioned in the general introductory sentences of the abstracts.

Based on these criteria and the initial focus of our mapping study, we screened the studies and after the screening process, a total of 64 references remained. After that, we started the mapping process with the creation of our own classification scheme following the keywording strategy described next.

3.2.4. Results

Keywording, data extraction and mapping

Word cloud is an efficient visual representation to depict keywords or tags based on the frequency of words in a given text. In such representation the more frequently used words are highlighted by occupying more prominence in the obtained diagram. In (MCNAUGHT; LAM, 2010), a word-cloud analysis is used as a supplementary research tool to improve researchers' understanding of data from different research projects. Hence, in order to automate part of the process of creating a classification scheme, we made use of a keywording approach supported by the free online tool of word cloud called *Wordle*[1]. Figure 3.2 shows some examples of word clouds extracted from content of different articles/webpages considered by this mapping study. The obtained word clouds guided us in the preliminary stage of defining our classification schema, once they helped to elicit the initial keywords we used in search strings.

Thus, this adopted strategy not only reduced the time needed to develop the classification scheme for the social machine area but also contributed to refine search strings used to gather relevant papers. The process consists of two steps. First, we create individual word cloud using paper content with the aim of identifying the main topics that reflect the contribution of the paper. After that, the set of generated word cloud diagrams of different papers are analyzed and possibly combined together to develop a "research cluster" in our mapping study. Such "research clusters" represent categories with a high level understanding about the nature and contribution of a specific research topic.

[1] http://www.wordle.net/create

Figure 3.2 - Examples of word cloud generated using Wordle

Data extraction and mapping start when the first version of the classification scheme is defined. Then, the considered relevant studies are mapped into the scheme. It is important to note that the classification scheme changes while doing the data extraction and mapping. New categories can be added to the scheme and the existing ones can be merged or spitted as well.

3.2.5. Validation

In order to validate the obtained results we also incorporated to the process a step of "gathering experts' opinion". In this step we made use of *online live polls*[2] which were launched during presentations in conferences and/or specific forums of discussion.

Particularly in this case, different live polls (which we refer to as "social slides") were launched in the open session of the *First International Workshop of the Theory and Practice of Social Machines*[3], when the results of this mapping study were presented. Such live polls were answered by an audience of more than 40 researchers and practitioners. In such pools we presented our general classification scheme and asked the audience, for example, how their solutions (i.e., already implemented SMs) better fit into our classification. These pools helped us in the process of characterizing Social Machines through the identification of the most chosen visions/keywords of our classification scheme

[2] http://www.polleverywhere.com/
[3] http://sociam.org/www2013/

as well as new terms to be considered. In general, this strategy has shown to be an excellent option to get faster feedback from experts in the field and a practical way to improve the obtained results. The next section discusses the obtained classification scheme for characterizing social machines.

3.3. A Classification Scheme for Social Machines

By performing the aforementioned mapping process, we characterize the "Social Machines" paradigm as a result of the convergence of three different visions: *i) Social Software*; *ii) People as Computational Units* and *iii) Software as Sociable Entities.* To better visualize this convergence, we use a similar diagram illustrating approach presented in (ATZORI; IERA; MORABITO, 2010). In this way, it is possible to clearly highlight and classify the main concepts, technologies and standards with reference to the various visions of Social Machines that are best characterized by this mapping. Figure 3.3 shows the result of this process of convergence.

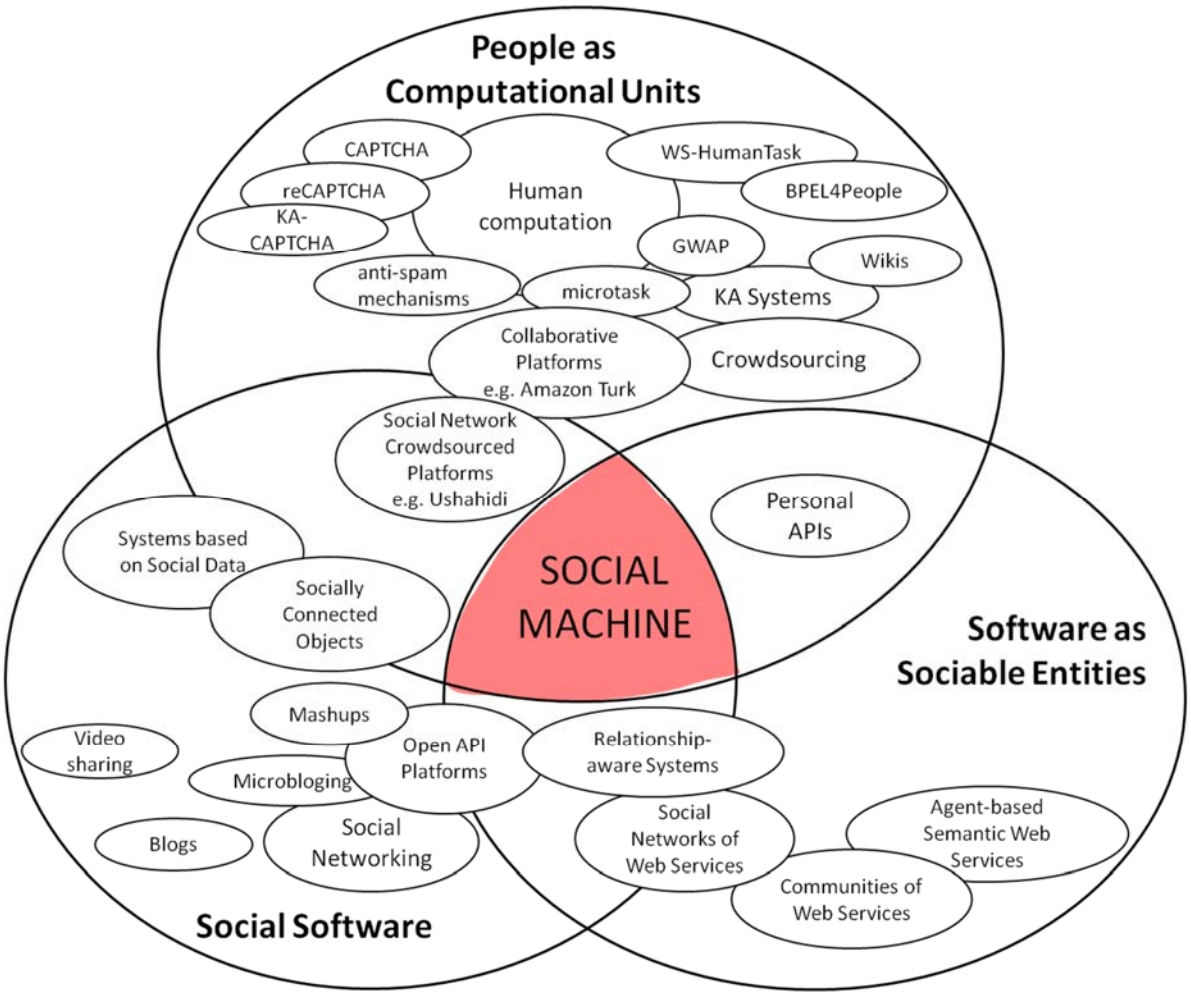

Figure 3.3 - Converging diagram of the different research visions of social machines

3.4. Social Software

This vision refers to the foundations of Social Machines. It includes initial research efforts and solutions that can be used to enable the creation of other socio-technical systems, which is the case for example of “Open API Platforms”. Systems built on top of these platforms and evolutions of current social network of people are also considered in this category.

3.4.1. Early Social Machines

Social Machines has its origins on social computing (ROUSH, 2005). Thus, some initial generation of Web-based social software (collectively called “Web 2.0” which consists of *blogs*, *social networking websites*, *video sharing*, etc.) can be seen as early versions of Social Machines. These technologies have allowed users to interact and collaborate with each other by storing and sharing various types of content, including messages, photos and videos. In fact, social media such as *Twitter* and *Facebook* have substantially changed the way we communicate and engage with others. They are now an important resource of everyday life and work, and one of the main enablers for creating an ecosystem of people, enterprises, and possibly things that collaborate in extraordinary innovative ways. In this sense, a recent work by Judith Donath (DONATH, 2014) – titled “*The Social Machine: Designs for Living Online*” – makes an analysis of social machines under the perspective of designing innovative interfaces for social online environments that explore new ways of interacting and communicating.

3.4.2. Open API Platforms

Besides transforming the manner we communicate, these systems have also been changing the way we develop software. This is because some of them, mainly social networking sites (e.g., *Twitter*, *Facebook*), have started a movement to expose their internal capabilities as Web Services in the form of open online application programming interfaces (*Open API Platforms*). Indeed, such concept of platform of services has completely transformed industry and society (JACOBS; JAFFE; HEGARET, 2012) and, as a consequence, it has been especially influential in the way we develop software (MAXIMILIEN; RANABAHU; GOMADAM, 2008). The Open API Platforms 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 (KO et al., 2010). Figure 3.4 uses data from *ProgrammableWeb*[4] and shows the dramatic increase in the number of available APIs from 2005 to 2012. According to DuVander (DUVANDER, 2012), in 2012, almost 15% of all APIs were added in only three months, with almost one per day added during the month of May.

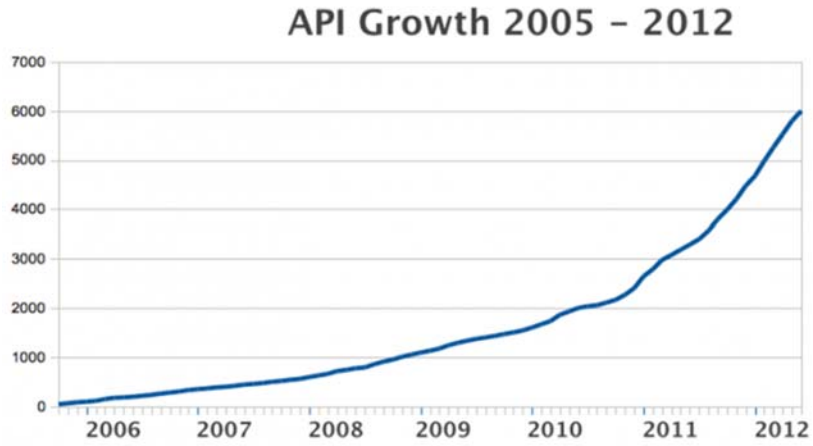

Figure 3.4 - API Growth 2005 - 2012

3.4.3. Systems based on Social Data

In practice, the direct consequence of the API growth was the rapid development of a *mashup ecosystem* (YU; WOODARD, 2009) in which Web-based mashups are created by integrating data from one or more sources to build new applications. *ProgrammableWeb*, the largest online repository of information about mashups and APIs, is concrete proof. Clearly, the combination of social information from multiple sources has enabled the creation of a novel breed of applications and service based on social data. In (ANDERSON et al., 2010), Anderson et al. present systems that take advantage of social data to infer preferences, trust between individuals, and incentives for resource sharing. Based on the results of their social inference functions, such systems can provide social knowledge to support other applications in their decision making processes, as per presented in (IAMNITCHI; BLACKBURN;

[4] www.programmableweb.com/

KOURTELLIS, 2012). In this sense, we can also find some initiatives on "*social search*" (GHADERI; YAZDANI; MOSHIRI, 2010; JIANDONG CAO; YANG TANG; BINBIN LOU, 2010; ZHANG; LI; XING, 2012), in which a combination of *social network services* (SNS) and *search engines* is made to improve web search results. These *social search engines* take into account the users' social graph and other data generated by existing social applications such as the Google+, Facebook and Twitter with the aim of helping users to quickly get the information they need.

3.4.4. Socially Connected Objects

Other examples of systems based on social data (in this case, using physical objects) have indeed been created by a digital agency called iStrategyLabs[5], which transforms real-world objects into machines controlled by social data. This combination of physical objects and social data is referred to as "Social Machines"[6], machines that turn a *Facebook* like, a *Tweet* or a *FourSquare check-in* into events to trigger actions on physical objects. Figure 3.5 is a picture of the "social cooler", one of the social machines created by iStrategyLabs. It is "hacked" to open according to certain rules as, for example, when a group of friends (using a location-based social network like Foursquare) do *check in* at a specific place. As can be seen, personal relationships and social interactions have also been evolving to include inanimate objects.

Figure 3.5 - The Social Cooler

[5] http://istrategylabs.com/
[6] www.facebook.com/socialmachines

Also inspired by this idea that almost everything will get socially connected, Peter Semmelhack (SEMMELHACK, 2013) defines social machines as an extension of the current social networks, in a way that includes not just other people but Internet-connected machines and all kinds of products and eletronic devices. According to Semmelhack, social machines represent an evolution of the relationships we have been building with machines for a long time. In practice, they are defined by Semmelhack as products that combine useful features with social networking as, for example, the *Nike+ product line*[7], which if formed by socially connected products that share performance metrics and other fitness data of their users.

Still in this context, and with the aim of supporting companies in the development of next-generation products, Salesforce makes available "*The Social Machine*"[8]- a cloud-based platform that provides products with a "voice" and enables organizations to "listen" their connected products. *The Social Machine* allows businesses to connect, control and engage machines into core business process via the *Salesforce Platform*. Using such platform makes possible to integrate machines as part of the business conversation and to be automatically notified by a connected machine before it breaks down, for example.

3.5. People as Computational Units

This vision refers to research efforts that integrate people, in the form of human-based computing, and software into one composite system. In this vision, the relevant computing (or even part of it) is performed by people.

3.5.1. Human Computation

The centerpiece of this vision is the idea of *Human Computation* which relies on systems that makes use of human abilities for computation to solve problems that are trivial for humans, but complex for machines (YUEN; CHEN; KING, 2009).

[7] http://nikeplus.nike.com
[8] http://www.etherios.com/products/the_social_machine/

Adopting this vision, CAPTCHA (LUIS VON AHN et al., 2003) and its extensions (i.e. reCAPTCHA[9] (VON AHN et al., 2008), KA-CAPTCHA (DA SILVA; CRISTINA; GARCIA, 2007)) can be considered kinds of Social Machines that use human computation to solve a challenge response test in order to make a distinction between humans and computers. The reCAPTCHA System (Figure 3.6), for example, has been successfully used to help to digitize old printed books and old editions of newspapers. It improves the process of digitizing books by displaying words (from scanned texts that cannot be read by computers) to the Web in the form of CAPTCHAs for humans to decipher. It is estimated that about 200 million CAPTCHAs are solved by humans around the world every day[10].

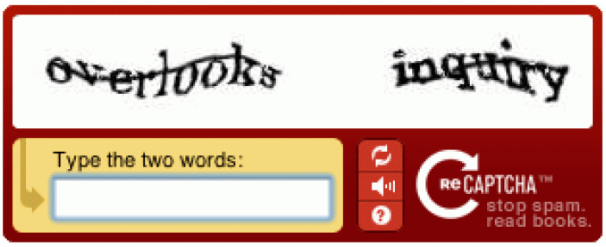

Figure 3.6 - reCAPTCHA

Duolingo[11], a free language education tool, is another example of system that uses the same principle from CAPTACHA. Its users create value by translating real-world documents while they are learning. Here is a summary of the *Duolingo's business model*[12]:

> *"Somebody who needs a webpage translated uploads it to Duolingo. That document then gets presented to Duolingo students who can translate it in order to practice the language they are learning. When the document is fully translated, Duolingo returns it to the original content owner who, depending on the type of document they uploaded, pays for the translation."*

Standards such as *WS-HumanTask* (AGRAWAL et al., 2007a) and *BPEL4People* (AGRAWAL et al., 2007b) have also introduced specifications that consider human interaction in the compositions of services in Service-oriented Architecture (SOA) environments (SKOPIK et al., 2011). In the same

[9] http://recaptcha.net/
[10] http://www.google.com/recaptcha/learnmore
[11] http://www.duolingo.com
[12] Extracted from http://www.duolingo.com

context, other studies (DUSTDAR; BHATTACHARYA, 2011)(SCHALL; TRUONG; DUSTDAR, 2008)(SCHALL; DUSTDAR; BLAKE, 2010) also propose models, such as the *Social Compute Unit* (SCU) (DUSTDAR; BHATTACHARYA, 2011) and *Human-Provided Services* (HPSs) (SCHALL; DUSTDAR; BLAKE, 2010), in conjunction with frameworks to deal with the seamless integration of human capabilities into a cross-organizational collaboration. In general, we can see these kinds of *collaborative computing systems* as Social Machines, since they incorporate the vision of people as computational units that make *collaborations,* which typically involve both humans and software as computational units.

3.5.2. Crowdsourcing and Collaborative Platforms

Other examples of systems that consider people as computational units can be seen in practice, such as the *games with a purpose* (GWAP) (VON AHN; DABBISH, 2008). GWAP are systems in which a computational process transforms some of its tasks into an enjoyable game and delegates them to human game players. In (THALER; SIMPERL; WÖLGER, 2012), Thaler et al. evaluate such human-computation techniques and argue that:

> *"Human computation lets organizations outsource tasks traditionally performed by specific individuals or experts teams to an undefined group of remote workers over the internet."*

This is the case of *microtask* (another human-computation technique) which is the basis of some *crowdsoursing* and *collaborative Web-based platforms* such as *Amazon Mechanical Turk*[13]. According to Shadbolt (SHADBOLT, 2013), crowdsourcing and collaborative Web-based platforms can be seen, in a general way, as knowledge acquisition systems in the age of Social Machines.

3.5.3. Knowledge Acquisition Systems

In Shadbolt's review of Knowledge Acquiring Systems (SHADBOLT, 2013), he concludes that:

[13] https://www.mturk.com/

"These social machines are knowledge acquisition systems at scale and machines that are socially contextualized."

Therefore wikis, which also are knowledge acquisition systems, can be considered Social Machines that make use of human computation, through the distributed co-creation of content. According to (YUEN; CHEN; KING, 2009), other examples of *distributed human computation* can be found in some anti-spam mechanisms (e.g. Vipul's Razor[14]) and systems with the aim of eliminating optical character recognition errors, such as Proofreader[15] used in the Project Gutenberg[16].

Furthermore, in terms of complexity, Shadbolt suggests that the result of combining different social computation approaches (e.g., crowd sourcing, co-creation and social network) might create real Social Machines with relatively unsophisticated software (i.e., comparatively lower compute complexity), but with a stronger social engagement (i.e., higher social complexity). Relying on this idea, he highlights Ushahidi[17], an open crowdsourcing platform for mapping crisis situations, as an example of a more sophisticated Social Machine, in terms of social complexity.

Ushahidi is a Social Machine that combines social networking, crowdsourcing and co-creation to build a unique open source platform on the web for changing the way information flows in the world. Figure 3.7 shows some examples of major events around the globe in which Ushahidi engine was used. This includes the "Swine Flu" incident around the world, the 2009 Indian general elections, the war on Gaza in 2009 and xenophobic attacks mapping on South Africa.

[14] http://sourceforge.net/projects/razor
[15] http://www.pgdp.net/c/
[16] http://www.gutenberg.net/
[17] http://www.ushahidi.com/

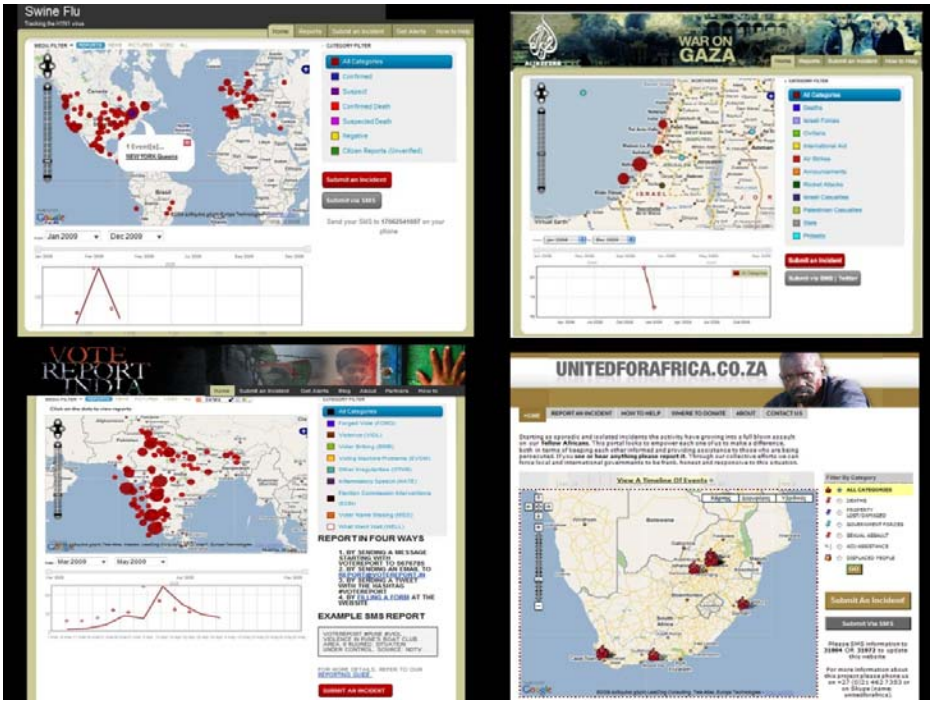

Figure 3.7 - Ushahidi[18]

3.5.4. Personal APIs

Recently, some practical initiatives have been wrapping people (their information and/or human capabilities) to provide a set of services through simple, clean, stable APIs on the Web. We can call these kinds of solutions as *Personal APIs*. The term *Personal APIs*[19] 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 "**API**zing" people is also a way of "**PERSON**ifying" software. *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 (BURÉGIO et al., 2014a). Based on this fact, Figure 3.3 presents *Personal APIs* in the intersection of the vision of "*People as Computational Units*" and "*Software as Sociable Entities*".

[18] bit.ly/19Gjj2B

[19] This 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.

Despite it is 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). In the following, we briefly present some existing practical initiatives built upon the concept of Personal APIs, namely: *The Human API*, *Naveen Selvadurai's API*, *Personal assistants* and *VoiceBunny*.

The *Human API*[20] 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 Human 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.

Naveen Selvadurai's *API*[21] is another example of *Personal API* that tracks health-related data. 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.

Personal Assistants - as a way to improve their communication and engagement with others, some professionals, such as the journalist Brian Proffitt[22] and the technologist Jay Cousins[23] have started specifying their own personal interfaces as a set of APIs on the Web. Under some communication

[20] Human API: A platform for human health data, available at http://humanapi.co.
[21] Naveen Selvadurai's API available at http://api.naveen.com/
[22] http://readwrite.com/2013/08/23/building-personal-api
[23] http://jaycousins.wordpress.com/about/personal-api-personality-interface/

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.

As aforementioned, beyond allowing access to personal data, there also have been APIs to trigger human capabilities. In this context, we can highlight *VoiceBunny*[24] whose 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.

3.6. Software as Sociable Entities

This vision is focused on works that try to weave social elements into software in order to enable their "socialization", mainly in terms of having "social" relationships with other software and interacting with each other. As a preliminary result, it is important to highlight that we are only considering the Web context. Other topics such as affective intelligent Social Machines (DUFFY, 2008), which refer to machines that speak our language and perceive our emotions, were not considered here.

3.6.1. Agent-based Web Services

Agent-based semantic Web Services (GIBBINS; HARRIS; SHADBOLT, 2004) is a research effort in this vision, since it represents an approach in which semantic Web technologies are used to improve the meaning of Web Services' descriptions and, consequently, to facilitate the interactions of loosely-coupled Web Services (at least in terms of discovery, reuse and composition (YAHYAOUI, 2005)). Some ideas regarding the use of a social unit to facilitate and improve the discovery of Web Services in an open environment like the Internet can be found in the research efforts of Benatallah et al. (BENTAHAR et al., 2007). In that work it is suggested to gather similar Web Services (WS) into groups known as *communities*.

[24] Voice actors and professional voice over recordings, available at http://voicebunny.com

3.6.2. Communities of Web Services

Maamar et al. (MAAMAR et al., 2007), motivated by the idea of communities, present the concepts and operations to specify and manage communities of Web Services. Hence, the involved Web Services interact with each other, in communities, to decide who will be responsible for treating a specific request. Under this Social Machines' perspective, these WSs represent services as sociable entities that are related in communities and interact with each other. Agent-based Web Services and the concept of communities formed the basis for the definition of reputation and trust models (e.g., (DUSTDAR; BHATTACHARYA, 2011) and (SCHALL; TRUONG; DUSTDAR, 2008)) that drive the discovery and composition processes of Web Services. More recently, the metaphor of "social networks" has been considered as an alternative to the use of communities of Web Services. (MAAMAR et al., 2010).

3.6.3. Social Network (SN) of Web Services

In order to support the process of discovery and composition, some works (e.g., (MAAMAR et al., 2011), (MAARADJI; HACID; DAIGREMONT, 2010), (MAAMAR et al., 2010), (MAAMAR; HACID; HUHNS, 2011)) suggest the use of historical records of Web Services interactions, in a SOA composition environment, as basis for extracting *Social Networks of Web Services*. Actually, Maamar et al. (MAAMAR et al., 2009) are one of the first researchers to advocate for the notion of *social Web services*. Different types of SNs (having Web Services as nodes) are captured, and the basic idea is to make a service recognize the relationships it participates in, and make recommendations about relevant peers. A service's peers include those that it can collaborate with, those that could substitute for it in case of failure, and those that it competes against (in the case of a selective environment). These approaches represent an important aspect for this vision of Social Machines. Once, such approaches turn Web Services into nodes of different social networks (e.g., *similarity-based SN*, *collaboration-based SN*) and make them aware of their relations with other peers, in this case, to support the process of discovery, composition and other collaborative processes.

3.6.4. Relationship-aware Systems

in this vision. In (MEIRA et al., 2011), the idea of Social Machine - as a unifying mental model for understanding, describing and designing each and every entity connected to the Web - points relationship as a fundamental element of such model. In fact, turning software into services on the Web means allowing it to interact with a huge number of other independently owned (and sometimes unknown) applications and services, and possibly establishing a plethora of "social" *relationships* with them. In this sense, a system can be viewed as a sociable entity whose interactions with each other are determined by their "social" relationships, just like people. In a more general sense, it inspires the idea of what we refer to as *Relationship-aware Systems*, which are an option for describing possibly related and interacting Social Machines that make use of notions from *computing*, *communication* (in the form of relationships and interactions) and *control.*

Figure 3.8 summarizes the timeline of part of the research adopting this vision of software as sociable entity. We mark (with "X") the milestones of research in the area. As can be seen, the research on relationship-aware systems started to gain momentum recently with the first edition of the *International Workshop on the Theory & Practice of Social Machines*, realized at the WWW Conference in May 2013.

3.7. Related Reviews

Although the concept of Social Machines overlaps other research fields and issues currently studied such as *SaaS*, *Cloud Computing*, *SOA* and *Social Networks*, we have not found any research that deals with the concept as we do propose herein. Some authors had already mentioned the term 'Social Machines' (ROUSH, 2006). However, the expression has been used with a different meaning, representing human operated machines responsible for socializing information among communities, that is, an intersection of the areas and studies of social behavior and computational systems.

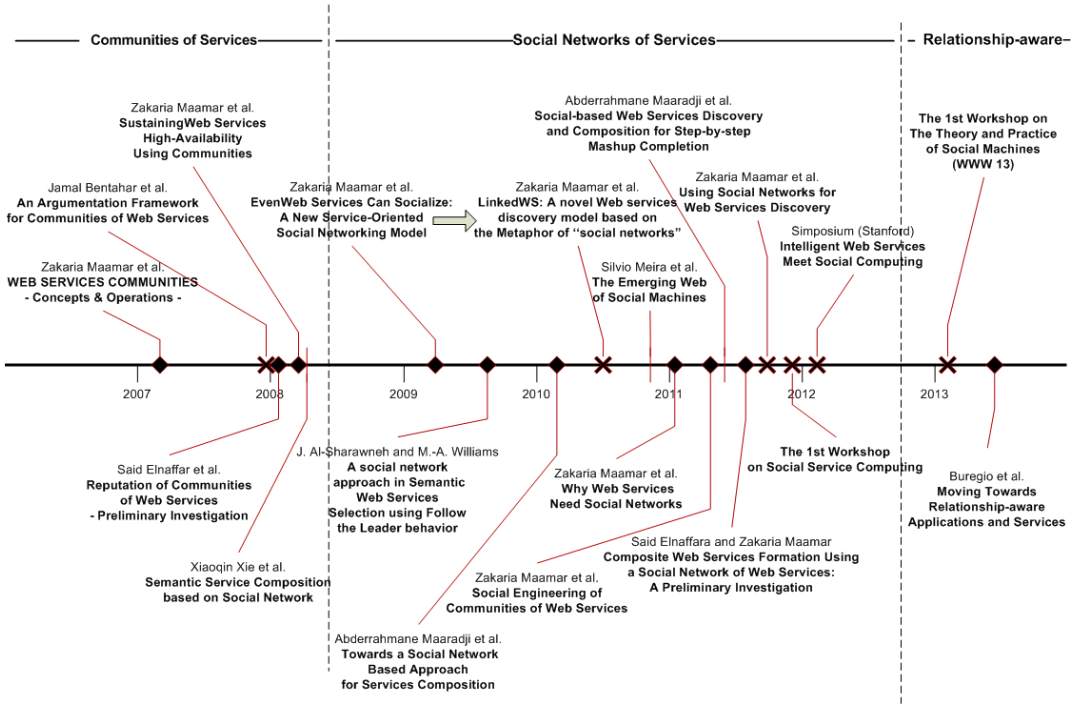

Figure 3.8 - Timeline of part of the research on software as sociable entities (2007 - 2013)

Initial ideas of Social Machines are also presented in (HENDLER; BERNERS-LEE, 2010), but currently there is no mapping study characterizing the Social Machine area as a whole. However, there are some studies that analyze and categorize specific aspects of related topics such as human computation (YUEN; CHEN; KING, 2009)(THALER; SIMPERL; WÖLGER, 2012) and knowledge acquiring systems (SHADBOLT, 2013). Yuen et al. (YUEN; CHEN; KING, 2009) give a survey on various human computational systems, defining the categories and their characteristics. They also present a discussion on performance aspects of human computation systems. In order to answer which technique is better in terms of costs and benefits, Thaler et al. (THALER; SIMPERL; WÖLGER, 2012) evaluate two prominent human-computation techniques: GWAP and microtask. Shadbolt (SHADBOLT, 2013) provides a comprehensive review of Knowledge Acquiring (KA) Systems and characterizes new kinds of emergent and collective problem solving. In this context, he presents a vision of Social Machines as KA Systems.

3.8. Concluding remarks

In this chapter, we have set the scenario to discuss Social Machine as a proper research area, including scientific inquiries and possible and different views of research topics. We characterized the Social Machine area through a mapping study on a set of existing work, outlined our adopted research methodology and made a discussion about the outcomes.

From our literature review, it clearly appears that the Social Machine paradigm relies on social computing and shall be the result of the convergence of the three main visions: *i) Social Software* (as its foundations), *ii) People as Computational Units* and *iii) Software as Sociable Entities.* In practice, Social machines are all about the emerging Web-enabled systems capable of blending computational and social elements into a variety of different *socially connected computing units* (e.g., software systems, hardware, people, physical objects and so on).

However, in the context of software engineering, the science, technology and implementations of Social Machines are in a very early stage, and there still is a need to provide a unified conceptual basis to describe, design and implement such emerging social systems.

According to Jifeng and Hoare's *unifying theories* (HOARE; JIFENG, 1998), understanding and defining a common conceptual model enables experience gained in the successful practice of software engineering to be rapidly generalized to new applications and to new developments in technology. Currently, we have different visions of social machine, however, based on our converging diagram, one question raises: *would it be possible to define a unified mental model of Social Machine capable of commonly describing, designing and implementing each and every socially connected entity that compose such emerging social systems?*

From a software engineering point of view, this unified perspective of Social Machines has meaningful implications on our concepts of what kinds of building blocks it might be possible to work with in the near future. While we do not have a unified model that converges the existing visions of social machines, questions still arise in this context, such as: *What are the elements that better*

represent a Social Machine? Which principles and constraints should be considered in order to build such kind of systems? What are the guidelines to describe and design Social Machines?

Thus, to start with, the next chapter proposes a common abstraction model that we have used as basis for establishing social machines as a unified paradigm to describe the emerging Web around us.

Describing Social Machines

"The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking."

Albert Einstein (1879 – 1955)
Theoretical Physicist

In the previous chapter we explored different visions of social machines and also motivated the need for a unified perspective capable of describing, designing and implementing social machines through a common abstraction model. Under the point of view of software engineering, defining a common model to describe systems involves thinking in high-level abstractions that can be used as building blocks to generalize both the system as a whole as well as the parts that compose it.

In the case of social machines, such parts can be viewed as the different *socially connected entities* (e.g., people, software, physical objects) which together compose this new breed of social systems. The current social machines are indeed all about this emerging kinds of Web-enabled systems that incorporate novel forms of *relationships* and *interactions* among a plurality of *socially connected computing units.*

Adopting a software engineering perspective, this chapter weaves "social" *relationships* into software, leading to the notion of "relationship-aware" systems. Relationship-aware systems use the metaphor of human social relationships and, at the simplest level, is software whose behavior takes into account other software it interacts with. Here, we revisit the concept of relationship and, based on that, we define a common abstraction model that is used as a foundation to establish social machines as a unified paradigm to describe each and every entity socially connected to this emerging Web around us.

4.1. Introduction

As aforementioned in Chapter 2, the Web has spread from a collection of documents to a platform for open, interactive, distributed applications and services. Such concept of platform of services has completely grounded up and increasingly transformed industry and society (JACOBS; JAFFE; HEGARET, 2012). Today, the Web is the dominant platform – an open programmable one - and, as a consequence, it has been especially influential in the way we develop software (MAXIMILIEN; RANABAHU; GOMADAM, 2008) (YU; WOODARD, 2009). In fact, since its conception the Web was open and decentralized, thus anyone could write new Web software without needing permission. This open approach is the main responsible for fostering widespread creativity, contributing to the current proliferation of distributed applications and services on the Web. Nowadays, we are facing a transition from "*siloed software*" to what we call "*sociable software*". Actually, we are writing such a transition, albeit in an ad-hoc way.

The term "*sociable*" does not necessarily refer to social applications or social networks. Here, it has a broader meaning. "*Sociable software*" means software designed and built from the ground up to (1) publish its internal capabilities (i.e., core functionalities) to its environment, (2) allowing the easy creation of other applications on top of its externalized capabilities and (3) being aware about its relationships with other software, changing its behavior accordingly.

The opposite implementation is considered "*siloed software*", a term used in (PATTERSON; FOX, 2012) to designate software that does not expose its internal components to other applications in the outside world. Leading companies like *Facebook*, *Google, Salesforce, Amazon*, *Twitter* can be considered seminal contributors to such a transition from siloed to sociable software. They recognized the world had changed and started a movement to externalize their systems' functionalities as platforms of services (KORIS; HODDINOTT, 2008)(KO et al., 2010). Indeed, they provide interfaces to many of their services at little or no cost, which allows individuals and other

businesses to create a multitude of other applications by [re]using and combining their provided services.

However, although the open Web platform offers many opportunities, publishing internal capabilities of a system on the Web presents considerable challenges(SIMOES; WAHLE, 2011)(LANTHALER; GÜTL, 2010)(ISSARNY et al., 2011). One of these challenges is related to a change in the semantics of *relationship*, which is the centerpiece of the modern Web. In fact, the Web of today can be interpreted as a "*dynamic set of relationships*" (TAYLOR; MEDVIDOVIC; DASHOFY, 2009) among collections of information, services, people and so on.

Hence, turning software into a "*sociable*" platform of services on the Web means making it able to interact with a huge number of other independently owned and sometimes unknown applications and services, and consequently establishes a plethora of *relationships* with them. In practice, it is increasing the number of applications on the Web that take into account their relationships with others. In our investigation, we discovered that Facebook is capable of establishing 2^{82} different *kinds of relationships* with other applications, changing its interactions with them accordingly.

This transition on the practice of software development leads to discussions in the field of software engineering including, among other things, the necessity of creating new mental models to describe and engineer this emerging sociable systems and their relationship-awareness features. Given that, one problem, in the context of software engineering, is how we could incorporate the notion of "*social relationship*" into the development of software, taking into account its implications on the way they will interact with each other as well as other *socially connected computing units* (e.g., people, physical objects).

In this chapter we discuss the weaving of "*social*" relationships into software, leading to the notion of "relationship-aware" applications and services which are a type of sociable software whose behavior takes into account other software it interacts with. Initially, we outline the adopted research guide and then we revisit the concept of *relationship* with the aim of establishing a unifying abstraction model for social machines in order to be used as basis for

specifying relationship-aware systems. The Social Machines' building blocks are defined as kinds of interacting *social service components*, and we set up their main elements, characteristics, types and guidelines for their analysis.

4.2. Research Guidelines

In this stage of research the main goal was to specify a unified abstraction model for describing social machines. *Brainstorming* and *focus groups* were used as the main techniques during this stage of our research. In the context of empirical software engineering, such techniques are often used to bring groups of people together in order to discuss specific topics. In this case, brainstorming is considered to generate ideas to answer a given question and focus group is similar to a brainstorming but – as its name says – it focuses on just one of the brainstorm particular ideas. According to Singer et al. (SINGER; SIM; LETHBRIDGE, 2008) they are excellent data collection techniques to use when one is new to a domain and seeking ideas for improvements. In our case, we collected experiences from focus group studies we conducted during three different postgraduate subjects on social machines that we offered along this research effort. Our studies also used these techniques for gathering experience from practitioners of companies like, for example, SODET[25]. Beyond *brainstorming* and *focus group* sessions, we also performed some *hands-on activities* to describe existing systems under the perspective of our proposed model. In general, all of these activities resulted in relevant and usable findings that supported us in the process of describing social machines.

[25] *Shifting Business into Social Machines*: http://sodet.biz/en/

Table 4.1 summarizes the research guideline adopted in this stage of research, as per defined in our research roadmap (Figure 1.1).

Table 4.1 - Research Guide for Stage 02: Describing Social Machines

Stage 02: Describing Social Machines	
Goal	Specify a unified abstraction model to describe social machines
Questions	How to merge both social and computational aspects into a unified abstraction model? How to describe existing applications with such unified model?
Evidences	Establishment of Basic Concepts; Definition of a common building block; Analysis Guideline; Description of existing applications (e.g., *Facebook*)
Main Methods	Brainstorming and focus group; Hands-on activities

4.3. Basic Concepts

As can be concluded from the previous chapter, the emerging social machines are Web-enabled systems that blend both computational and social elements, and possibly use a combination of different *socially connected computing units* (e.g., applications, services, people, and objects). In order to facilitate unified understanding, what about we abstract each of these different *socially connected computing units* into a high-level building block capable of describing the common features of such units? This question led us to think in a mental model for *social machine* that could be used to not only abstractly describe a *social machine* as a whole but also each of its *socially connected entities*.

We assume the context of software development to start conducting this process of abstraction. Then, prior to defining and detailing the elements of the social machine abstraction model, it is important to outline some basic concepts (e.g., “sociable” and “siloed” software) that we adopted in this process. For now, in order to understand the example used in the next section to illustrate the notion of “sociable” software, it is enough to just consider a social machine as an abstraction of a *socially connected entity*.

4.3.1. "Sociable" and "Siloed" Software

"Sociable software" is a term we have used to refer to a kind software that embraces the principles of service-oriented computing (ERL, 2005) and also, at the simplest level, was designed and built from the ground up to interact with other software, satisfying the following properties:

i) publishing its internal capabilities (i.e. core functionalities) to its environment;

ii) allowing the easy creation of other applications by [re]using and combining its provided services;

iii) being aware about its relationships with other software, changing its behavior accordingly.

We call "siloed software" – a term also used in (PATTERSON; FOX, 2012) - the opposite implementation of sociable software. To make the notion of "*sociable*" and "*siloed*" more concrete, Figure 4.1 illustrates a hypothetical online store system designed in both ways: Figure 4.1 (a) - as "*siloed software*", and Figure 4.1 (b) – as "*sociable software*". As we can see in Figure 4.1 (a), the siloed version is a monolithic piece of software with all the components built inside, behind a "*wall*". All its components have strong connections and can share and collect data together, which are what makes them all mutually coupled. Often, siloed software has only one access point to its functionalities and rarely exposes its internal components to other applications in the outside world. Thus, it is not designed early to be "*sociable*" much less to "*know*" its relationships with others.

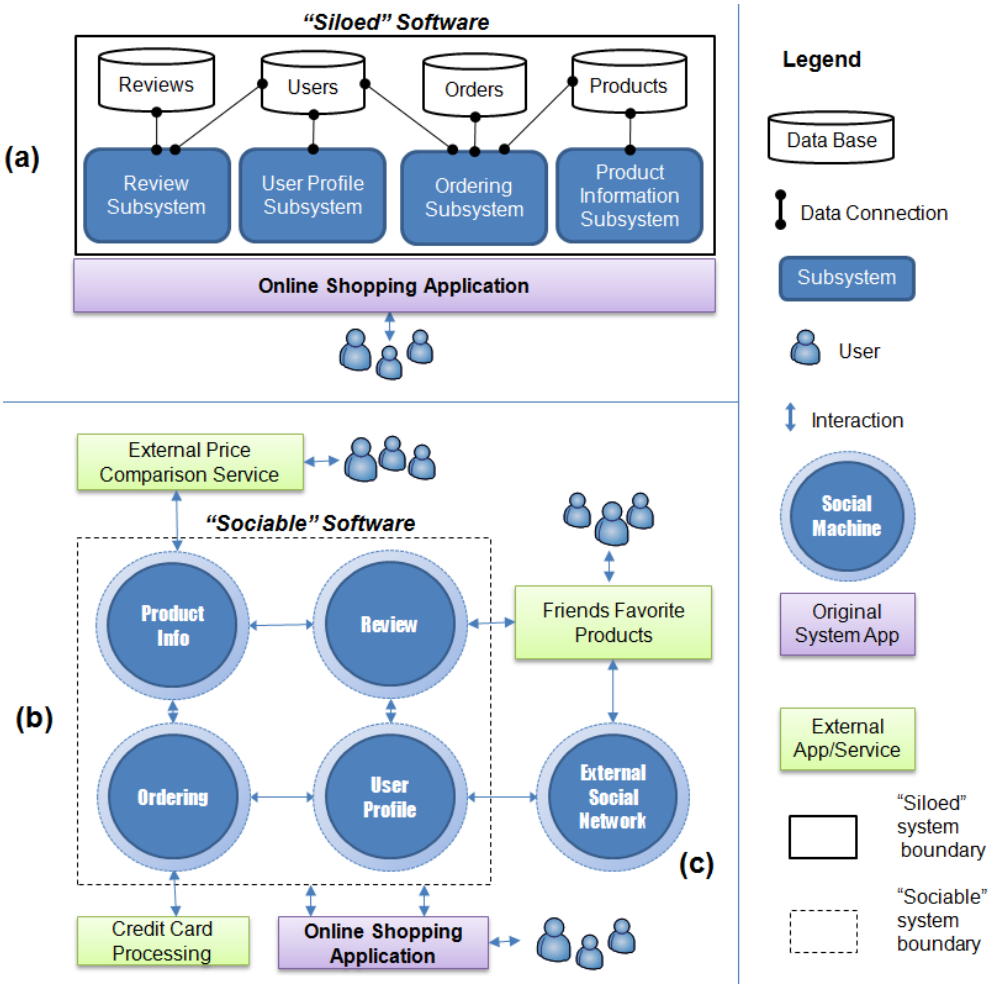

Figure 4.1 - Example of an online store system designed as: (a) "siloed software" and (b) "sociable software" which interacts with (c) - an external existing social network

On the other hand, the sociable version –Figure 4.1 (b) - is composed by connectable entities (Social Machines[26]) that interact with each other and implement the same original *Online Shopping Application*. Beyond that, they facilitate the creation of other external applications and services by allowing the reuse of the system's functionalities externalized to the outside world. In Figure 4.1 (c), for example, an external Social Machine (wrapping an existing social network) is being used in combination with the "*Customer Review SM*" to create a "*Friends Favorite Products*" service - a new service made by combining existing ones.

As one of the ideas behind Social Machines is taking advantage of the networked environment to facilitate the combination and reuse of existing

[26] We will detail the elements of a Social Machine; for now, just consider them as networked application building blocks.

services provided by other SMs, the SM's model has to embrace the principles of *service-oriented computing* as well as *component-based software development* to support the properties (*i*) and (*ii*) established for sociable software. Furthermore, some extensions are required into the SM's model to better incorporate property (*iii*) - *being aware about its relationships with other software, changing its behavior accordingly*. Figure 4.1 does not show clearly the property *(iii)*. Such a property is associated to the notion of *relationship* and *relationship-aware* software, which we discuss next.

4.3.2. Relationship

Relationship is an essential element in the Social Machine model. However, after some preliminary practical experiments (BRITO et al., 2012; MEIRA et al., 2011), we identified a need to extend its notion in order to better weave *relationship* into the SM's model, and consequently better satisfy the property (*iii*) of sociable software. In this sense, the first step was to give a clear definition for the relationship element. For that reason, with the purpose of providing subsidies needed to [re]define the idea of *relationship* to be adopted here, we analyzed its existing common definitions and different views along software engineering evolution.

Firstly, *relationship*, in general, can be defined as "*the way things are connected*" and, in this sense, it is often used interchangeable with terms such as "*connection*", "*association*", "*link*", "*relation*", and so on. On the other hand, focusing on a software engineering evolution's perspective, we recognized at least five different views of *relationship:*

Data-oriented view: relationship is a tuple of entities. This is the classic, relational algebra-based definition adopted by Peter Chen on the E-R Data Model in 1976(CHEN, 1976);

Object-oriented view: here, relationships represent different strengths of dependencies among classes of objects. UML offers five different types of class relationship for object oriented analysis and design, ranging from weaker to stronger dependencies (MILES; HAMILTON, 2006);

Architecture-oriented view: in the seminal Perry and Wolf's paper on software architecture (PERRY; WOLF, 1992), relationships are introduced as

basis for restrictions on software structure and the formal arrangements of its design elements. Formal relationships can be used to define different topologies of network architectures, components and data element associations;

User-oriented view: in social networks and applications, relationships correspond to connections between users. Actually, they form graphs of relations among people, organizations, states and other units. In this context, different applications describe user relationships using different terminologies such as “contact”, “friend”, “circle”, “follower”, “co-worker, and so on;

Service-oriented view: with regard to distributed and service-oriented systems, relationship underlies reasoning on trustworthiness. In these systems, trust relationships are used to infer reputation and control access to services and resources (SURYANARAYANA et al., 2004).

Figure 4.2 sums up the different views of *relationship* along software engineering evolution. As we can see, most of the views present a static aspect and have little or no focus on how software behaves and interacts with others. Therefore, after this analysis, we introduce the following general definition of *relationship* in the context of Social Machines:

> *“A **relationship** is 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.”*

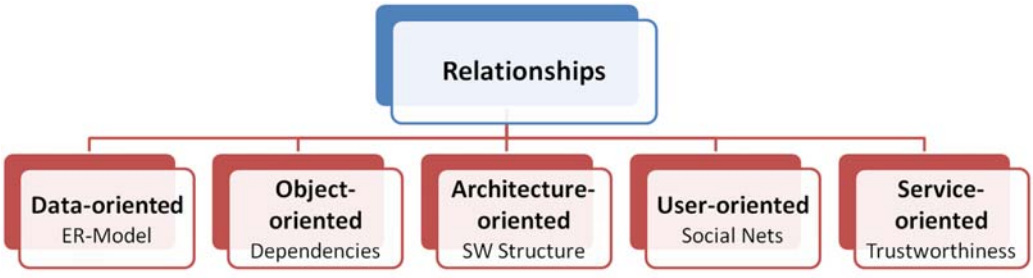

Figure 4.2- Different views of "relationships"

As aforementioned, our main objective is to weave this concept of relationship into the development of sociable software. In order to do this, we created the notion of *relationship-aware* application and services, which is a

kind of sociable software that satisfies its property (*iii*) – *being aware about its relationships with others* - and also establishes other relevant capabilities. During our definition process, we identified some challenges and issues that should be highlighted: (1) how applications can benefit from maintaining these relationships; (2) which is really the main idea behind being aware of relationships; (3) how to define a unified mental model that incorporates this notion; (4) which are the steps for analysis of relationship-aware software; (5) how existing applications can be described under this new perspective. We discuss all these issues along the next Sections.

4.4. "Relationship-aware" Applications and Services

4.4.1. Benefits from maintaining relationships

Based on our investigations of the concept of *relationship* and existing approaches (BUREGIO; MEIRA; ALMEIDA, 2010)(LEE; KOTONYA; ROBINSON, 2012)(MAAMAR et al., 2011)(MAARADJI; HACID; DAIGREMONT, 2010)(MAAMAR et al., 2010) (ANDERSON et al., 2010), we believe that giving preeminence to "*social*" relationships can enable new classes of systems and offer the potential for creating "*relationship-aware*" applications and services capable of: (1) *determining dynamic interaction views*, (2) *supporting the process of discovery and composition*, and (3) *reasoning on trustworthiness and privacy*.

1) *Determining dynamic interactions views:* inspired by the idea of dynamically providing different views (products) of the same system (BUREGIO; MEIRA; ALMEIDA, 2010)(LEE; KOTONYA; ROBINSON, 2012), and based on observations on existing open Web platforms (e.g., *Twitter*, *Facebook*), we recognized that establishing and being aware about different relationships with other applications and services is key for an application to determine its different interaction views (e.g., set of services under specific constraints) to be made available to its client applications. This is the main general aspect explored in this chapter and Facebook is a practical example of an application that deals with that;
2) *Supporting the process of discovery and composition*: extracting Social Networks (SNs) from the historical records of Web services interactions

in a SOA composition environment has been suggested in some work (e.g., (MAAMAR et al., 2011), (MAARADJI; HACID; DAIGREMONT, 2010) and (MAAMAR et al., 2010)) as an useful way to facilitate the process of discovery and composition of services. Different types of SNs (having Web services as nodes) are captured, and the basic idea is to make a service recognizes its relationships in the social networks it participates in, and makes recommendations about the peers with whom it would like to collaborate in case of compositions; the peers that can substitute for it in case of failure; and finally be aware of the peers that compete against it in case of selection. In conclusion, these approaches turn Web services into nodes of different social networks (e.g. similarity-based SN, collaboration-based SN) and make them aware of their relationships with others to support the process of discovery and composition. For this reason, they also represent an important aspect of "relationship-aware" software.

3) *Reasoning on trustworthy and privacy*: nowadays, we have much social information available connecting people or other entities through relationships. Hence, another capability of relationship-aware applications could be to consider the connections between their owners with the aim of assessing trustworthiness and privacy, as well as supporting other decision making processes. The idea is to enable the easy implementation of scenarios like: "*Allow access to functionalities provided by my research agenda service only to client applications that belong to professors defined as partners of mine in any private collaborating group on Mendeley*[27]". Some research efforts aiming at modeling user's social data can be found in the field of semantic Web technologies (e.g., SIOC[28], FOAF[29]) and social network (ANDERSON et al., 2010). They can be considered options to enable the kinds of scenarios we raised, and consequently achieve another benefit from making software aware of social relationships. In (ROMERO et al., 2013), the notion of Social Machines is used to propose a novel architecture for networked information systems that makes easier for users to control

[27] Open platform/social network for researchers http://www.mendeley.com
[28] SIOC: http://sioc-project.org/
[29] FOAF: http://www.foaf-project.org/

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.

It is important to note that the capabilities described above do not represent an exhaustive list of all features that can be enabled in relationship-aware applications. Actually, such capabilities represent different general aspects of the concept of *relationship-awareness* and examples of how applications can benefit from maintaining their social relationships with others. We know that each aspect may involve a wide field of research. For this reason, the initial focus of this chapter is to address only the first aspect, in order to understand, in a general way, how software's relationships can determine its dynamic interaction views. In the next Section we explain the idea behind such aspect with a simple analogy.

4.4.2. Analogy with Human Relationships

In Chapter 2, we realized that each software paradigm has its own abstractions and sometimes they rely on real-world metaphors. This section introduces the central idea of the "relationship-aware" aspect of sociable software, i.e., *determining dynamic interactions views*. It does so by first exploring a simple analogy between social relationships among people and relationships among software. We believe that understanding human social relationships offers a simple base of concepts from which Social Machines, and more generally, software interactions as a whole has performed. This analogy is useful to show that almost every software interactions can be explained under the perspective of social relationships. It is also valuable to introduce the elements that help understanding other concepts explored along this thesis.

The idea is as follows: in human society, the different *kinds of relationships* between people are keys to determine the different *sets of interactions* between them. For example, the possible *set of interactions* between a man and his daughter is for sure different from the expected *set of interactions* between him and his boss. This analogy is quite aligned with our definition for "*relationship*", once it constraints the way of how two or more people have interactions with each other. Due to that reality, a same person can

exhibit several *sets of interactions*, depending on the existing *relationship* with whom he/she is interacting with. Thus, how could we map this into software?

An easy way to put the same idea in the context of software is by mapping the different *sets of interactions* of a person to the different *interaction views* (i.e. set of services/functionalities) that an application can make available. Thus, just as with people, different *interaction views* can be provided by software according to the *types of relationships* it establishes with applications that use its services. This is a very common behavior in the Web today, especially regarding open Web platforms. Figure 4.3 illustrates a sociable application providing different *interaction views* (V_1 to V_n), whose properties (e.g., set of services, rate limiting, performance) are determined according to the *relationship* between it and its client applications.

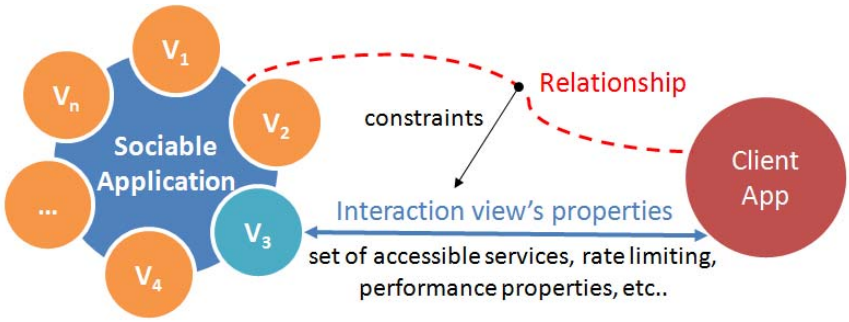

Figure 4.3 - Relationship-aware application: relationship determining interaction views

It is important to note that, in this context, two different *interaction views* mean either *i)* two *sets of different services* or even *ii)* the same *set of services* being provided under different quality constraints. Twitter.com, for example, implements the latter by establishing *feature-based rate limit* which enforces a rate limit for third party applications through restricting the number of Twitter API requests that can be performed within the application. This rate limit varies based upon the type of operation being performed as well as the third party application. On the other hand, Facebook provides different options of "relationships" to its client applications, determining the set of accessible services (different interaction views) accordingly.

Another aspect to be noticed here is that different *interaction views* can be seen as different dynamically provided "*products*", composed by different sets of services and properties. Thus, a sociable application, under this

perspective, can also be viewed as a system that incorporates somehow the concepts of *Dynamic Software Product Lines* (DSPL) (BUREGIO; MEIRA; ALMEIDA, 2010), more specifically, it can be considered as a dynamic service-oriented product line.

In practice, we have been writing different kinds of "*relationship-aware*" software, albeit in most cases in an ad-hoc way. The spreading of the Web as a software development platform along with the increased interactivity and connectivity of applications and services has changed our understanding of the nature of computing. Many computational processes are nowadays Web-based, autonomous and concurrent. The status of the digital universe cannot be properly accounted for, from the engineering point of view, by the use of the computing metaphor alone. In today's highly interconnected software architectures we should consider interactions, relationships and their constraints on software behavior.

However, in the context of the new emerging Web developments, there has not been a clear, precise description that incorporates these concepts into each and every entity used to compose applications and services. For this reason, we believe it is necessary to provide new mental models capable of representing such aspects as well as providing a common and coherent conceptual basis for the understanding of this young, upcoming and possibly highly innovative phase of software development.

Thus, inspired by the notion of *Social Machines* and based on some practical developments with it [Burégio et al. 2014; Burégio et al. 2013b; Meira et al. 2013; Brito et al. 2012; Nascimento et al. 2012], we extended its initial model as well as its core elements to create a common abstraction that has the potential to describe any existing application or service through a unifying building block that makes use of concepts from **computing**, **communication** and **control** in order to specify possibly related and interacting entities.

4.5. The Social Machine Model

We understand the Social Machine model as a high-level abstraction that provides the elements for transforming any computing unit (e.g., a piece of software, hardware or even a person) into a *socially connected information*

processing system. In this sense, our abstraction model blends the principles of *component-based development* (HEINEMAN; COUNCILL, 2001)(LAU; WANG, 2007)(MCGOWAN, 1975)(SZYPERSKI, 2002) with those of *service-oriented design* (ERL, 2007)(PATTERSON; FOX, 2012) through a new software abstraction that considers social aspects in the form of *relationships* and *interactions*. This abstraction model is defines 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."*

Because of the fact that this model weaves a social aspect into a single unit that blends principles of *software componentization* with principles of *service-oriented development*, we often refer to this social machine's abstraction model as a *Social Service Component* (SSC), i.e., a software building block (*component*) that provides a set of *services* which can vary according to its "*social*" relationships with others. Together, such building blocks interact to compose new social systems, as illustrated in Figure 4.4.

The social machine's abstraction model is built on three key concepts: **computation**; **communication** and **control**. Understanding the role each plays is fundamental to describe the model as a whole. In this section, we provide an overview of these concepts relating them to the elements presented in Figure 4.4. After that, we proceed to a more detailed look at how applications can be analyzed using the model.

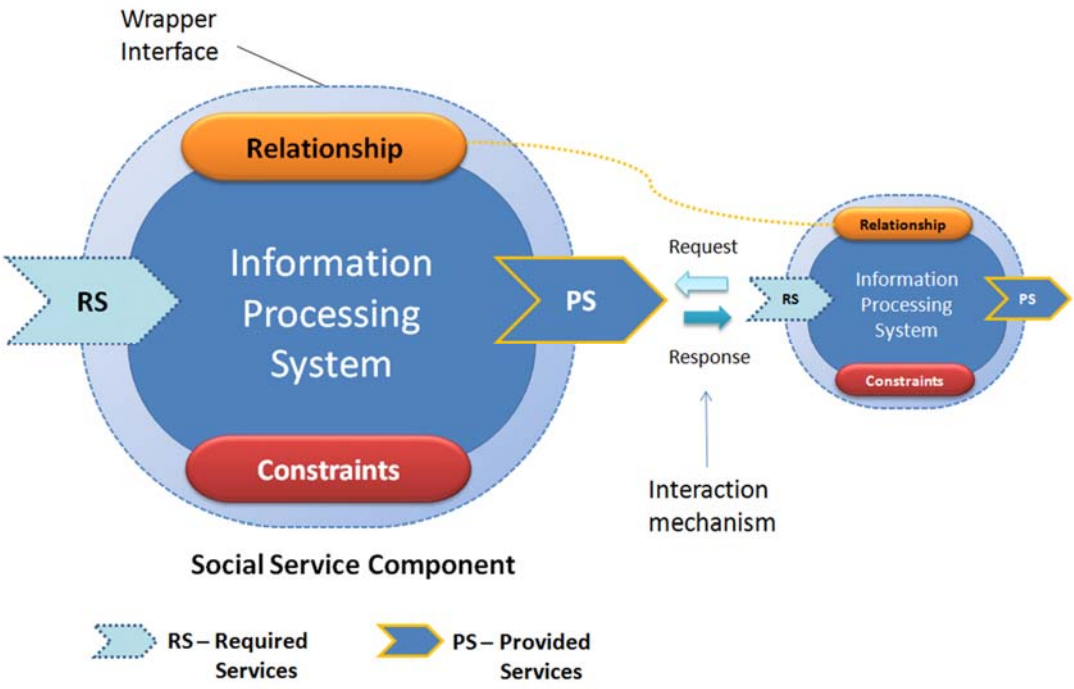

Figure 4.4 – Conceptual view of the Social Machine's abstraction model

4.5.1. Computation

The daily practice tells us that higher levels of abstraction are needed to properly deal with real-life situations. Therefore, the traditional notion of algorithmic Turing Machine was generalized and represented in Figure 4.4 by a single element: the *Information Processing System*.

Information Processing System *(IPS)* abstracts any computational unit whose behavior is defined by the functional relationship between inputs and outputs. It can be either a piece of hardware or software, or even a person. To better understand the wide scope of this element we can adopt the three components of information processing systems defined by Burgin (BURGIN, 2005): *hardware* (physical devices), *software* (programs that control its operation) and *infoware* that represents information processed by the system. (DODIG-CRNKOVIC, 2011). Hence, an IPS can be represented, for example, by an algorithm, a Web service or even a network of computer processes at different scales or levels of granularity. As seen, in general, any mechanism which ensures definability of the Social Machine's business logic.

4.5.2. Communication

Bohan Broderick (BOHAN BRODERICK, 2004) distinguishes computation and communication by considering that the former is limited to actions within a system, and the latter is an interaction between a system and its environment. Hence, wrapping an Information Processing System, a Social Machine goes beyond computation and incorporates the notion of communication, having relationship as its fundamental element. Figure 4.4 illustrates the common set of abstractions related to this notion of communication in the Social Machine model, which are: *Relationship*, *Wrapper Interface*, *Provided* and *Required Services*.

Relationship *(Rel)* is the centerpiece of the Social Machine model. Dealing with relationships enables the establishment of a multitude of different kinds of interactions between the computing process and its environment and, as a result, provides a vision of engineering software which involves communication and information processing. In practice, a relationship between two Social Machines can be obtained by prior establishing a true persistent relation between them. For example, to have specific kinds of interactions with applications, such as *Twiiter, Facebook, Dropbox*, a client application needs to be registered before calling their provided services, and, in most cases, different constraints are associated to these relationships in order to determine specific interaction views. Other types of relationships can also be considered. Thus, the characterization of relationships can be made in several dimensions by classification into orthogonal types: persistent/temporary, directed/undirected, and explicit/implicit. However, regardless of the types, the main idea to be highlighted here is the notion of relationship as key to determine the different sets of interaction views, as we have already explained before. The concept of relationships between SMs is similar to that of relationships between people; we can view them as trusted relations between different SMs, satisfying established constraints. In Section 4.7 we show how Facebook implement such behavior.

Wrapper Interface *(WI)* encapsulates the SMs computational unit (i.e., *Information Processing* System) and provides an interface of functionalities to be used by the SM's services. It also comprises a *mapping layer* which is responsible for dealing with data (converting, formatting, etc.)

that flow from the SM's services to the wrapped computational unit, and vice versa. Then, it maps requests to the *IFP's* inputs, and the *IFP*'s outputs to the corresponding responses.

Provided Services *(PS)* represents the SM's business logic that is exposed as a dynamic set of services. For example, considering Twitter as a SM, the API it provides could be considered a kind of provided service. Through Twitter's API it is possible a client application to interact with its main services (e.g., search, tweet, direct messaging, retweet). In general, the Provided Services can be categorized into two main types:

- *Open Common Services*: represent publicly available services whose access to them does not require the prior establishment of a specific relationship between the provider Social Machine and its client applications. In practice, open services accept requests from "unknown" and unauthenticated applications. Yahoo and Google Maps are good examples of providers that offer fairly open services accessible by any application through their public APIs. Taking our analogy with human relationships, open services could represent, for example, the general set of interactions we set up for unfamiliar or unknown people. In the case of Social Machines, this set of services forms a single and common interaction view whose access is limited by a general set of constraints. As an example, a Social Machine can limit the number of requests from a specific IP address to its open services. Google, for example, impose rate limiting on its public APIs, as well as Twitter, Yahoo, Facebook and others.

- *Relationship-driven Services*: beyond the open common services, a Social Machine can make available a group of services whose access to them are constrained according to the relationships established between the SM and the client applications interacting with it. This is the case of a SM dynamically provides different interaction views in accordance with its relationships with others. Twitter's API, for example, is open for looking up public information about a user, but other operations and conditions require a prior establishment of a

relationship between Twitter and the application intended to call its services. Each set of different services represents a specific interaction view, but different interaction views can also be created by the same set of services when they are provided under different quality attributes. This behavior follows the aforementioned interaction pattern, illustrated in Figure 4.3.

Required Services *(RS)* is an optional element defined by the proposed model. It represents the set of services that a Social Machine needs to invoke in order to work properly. It has the same semantics of required interfaces which is a generally accepted element adopted in most software component models (LAU; WANG, 2007). Assuming this concept is very useful because it facilitates the specification of functional and structural dependencies between SMs.

4.5.3. Control

In the Social Machine model, the semantics of control is related to any restrictions or rules that a given SM establishes in order to influence or determine its interaction views with others it relates to. Such restrictions are represented by the element Constraints.

Constraints *(C)* specify the rules or limitations that take place during the establishment of relationships and definition of the interaction views among different SMs. We consider two main types of constraints: *Visibility* and *Quality*.

- *Visibility Constraints*: this kind of constraints is related to the visibility restrictions on the services provided by a Social Machine. Often they specify different types of access modifiers or permissions that determine the different sets of services (i.e., *interaction views*) to be accessed by others (according to the relationship established between them).
- *Quality Constraints*: this type of constraints encompasses the restrictions that influence the quality attributes of the services provided by an interaction view. They can specify, for instance, authorization protocols (for security), number of requests per hour

(for performance) or any additional properties that can influence any other quality attribute. The Twitter API platform, for example, provides different options of rate limits, according to the kind of relationship established with the applications that access its services.

Table 4.2 summarizes the basic elements of a Social Machine model.

Table 4.2 - Basic Elements of the Social Machine Model

Element	Description
Information Processing System (IPS)	Represents an abstraction of any computing unit (e.g., a piece of hardware, software, person, things).
Relationship (Rel)	A persistent type of connection between two Social Machines. Relationships are responsible for determining the degree of interactions between SMs. They can be *registered* or *inferred* relationships.
Wrapper Interface (WI)	The interface of functionalities that encapsulates the information processing system and forms the basis for the SM's services.
Provided Services (PS)	Dynamic set of services provided by the SM to the external world. They can be categorized into *relationship-driven* or *open common* services.
Required Services (RS)	The optional set of services required by the SM to work properly. In some SMs these required services are used internally and are not exposed like the provided services.
Constraints (C)	Rules or limitations that constraints the interactions with the Social Machine's services.

4.5.4. Discussion

The conceptual view of social machine depicted in Figure 4.4 helps in describing and designing both an entire system as a *relationship-aware social machine* (BURÉGIO et al., 2013b) (e.g., the facebook platform as per presented in Section 4.7) as well as each of its possible *socially connected parts* (e.g., modules, subsystems, users).

In practice, any computing unit that makes sense to establish relationships with others and interact according to such relationships, can be socially wrapped up into a set of dynamic and specialized services, and so described as a *relationship-aware social machine*, following our SM abstraction model. These entities include, for example, a source of information, a stateless service, a collection of other socially wrapped entities, and people with their information/behaviors.

In a meta-level architecture, our model can also be used to compose existing Social Machines into new ones. In this way, the obtained system is the result of a set of SMs working together.

4.6. Analysis Guideline

The main steps we use to analyze a system as a social machine are grouped in Figure 4.5.

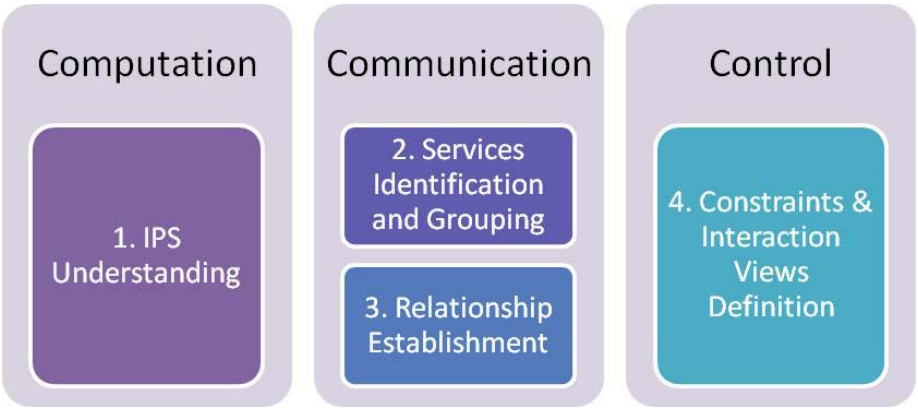

Figure 4.5 - Analysis Guideline

Each step can be summarized as following:

1. *IPS Understanding:* The first step is to identify the kinds of information the IPS deals with and, when viable, how such information are internally represented and which are the possible operations over them. Understanding the *Information Processing System*, the information it deals with and its operations, helps the task of grouping the services that expose to the outside world its internal functionalities. An acceptable result of this step can be a description of the different types of identified information and, if possible, a representation of how such information is internally structured.
2. *Services Identification and Grouping*: The objectives of this step are both analyzing how the IPS's internal functionalities are exposed (i.e., which kind of *Wrapper Interface* should be considered) and identifying and grouping the set of *Provided Services* and, when applicable, the set of *Required* ones. At the end of this step, an abstract model, containing the groups of related services and the operations they perform, should be provided.
3. *Relationship Establishment:* This step involves describing how *relationships* among Social Machines are established. In practice, we have realized two types of relationships:
    - *Registered relationships*: represents the relationships that are established through a manual (in opposite to inferred) registration process. In this case, before two applications start to interact, the owner of one of them should initiate a process (provided by the other application) to register his application as a valid pair to interact with. It is the typical case of the registration processes offered, for example, by *Facebook, Google APIs, Twitter* and other Open API Platforms (BURÉGIO et al., 2013a) in order to allow third part application to interact with their provided services.
    - *Inferred relationships*: are those relationships automatically established between SMs based on existing data of the owners of the involved applications. Systems based on Social Data (BURÉGIO et al., 2013a) are an example of systems that take advantage of social data of their users to infer preferences, trust between individuals, and incentives for resource sharing. Based on the results of their social

inference functions (IAMNITCHI; BLACKBURN; KOURTELLIS, 2012), such systems can provide social knowledge to support the automatic establishment of relationships among different applications.

4. *Constraints & Interaction Views Definition*: This step corresponds to the definition of the constraints that somehow influence the formation of the different interaction views provided by a Social Machine. Optionally, they can also be classified into the types aforementioned (i.e., *Feature-based*, *Visibility* and *Quality*).

Based on this analysis guideline, an experience report describing an existing system (i.e., *Facebook*) as a Relationship-aware Social Machine with 2^{82} interaction views is showed in the next Section.

4.7. Describing Social Machines In-Action

The Social Machine building block is indeed a way of modeling the social Web. As it establishes a new perspective, one question is how to describe existing social systems as Social Machines. Motivated by this question and prior to starting some implementation cases, we made an effort to describe some existing Web-enabled social systems under the perspective of our Social Machine model. This included systems like *Twitter*, *Dropbox*, *Facebook* and others. The next section shows part of the analysis of *Facebook* as a SM.

4.7.1. Facebook: A Social Machine with 2^{82} interaction views

A primary example of the power of Social Machines as a model for describing the emerging social systems can be seen in an application most of us are familiar with: *Facebook*. *Facebook* is one of the most complete Social Machines in the Web of today. It knows not just the relationships among its users, but it also "*knows*" its relationships with other applications, what make it a true *relationship-aware* Social Machine with 2^{82} interaction views. Because of that, we adopted it as an example of how we could describe an existing application under the perspective of our Social Machine model. Thus, in order to do that, we performed the steps defined in our proposed analysis guideline, described in Section 4.6, which includes the following steps:

1. *IPS Understanding*;

2. *Services Identification and Grouping;*
3. *Relationship Establishment and*
4. *Constraints & Interaction Views Definition.*

1. ***IPS Understanding***: The Facebook's Information Processing System deals with the relationships among its users and other objects. To do so, it uses the notion of *Open Graph* as its main core concept. The *Open Graph* models *User*'s activities based on *Actions* (the interactions users can perform), and *Objects* (the target for actions taken by users). In this sense, it defines an open model for relating users to objects. In this model any user's action on an object can be mapped as a relationship in the graph. For example, suppose a user is using a Facebook integrated app about cooking recipes, which allows users to publish on Facebook when they cook something. Once a story like that is published by the app, a relationship of type 'cook' between the *user* and the recipe cooked (the *object*) is created on the *Open Graph*'s structure. A general representation of Facebook's *Open Graph* and its abstractions (user, action and object) are shown in Figure 4.6.

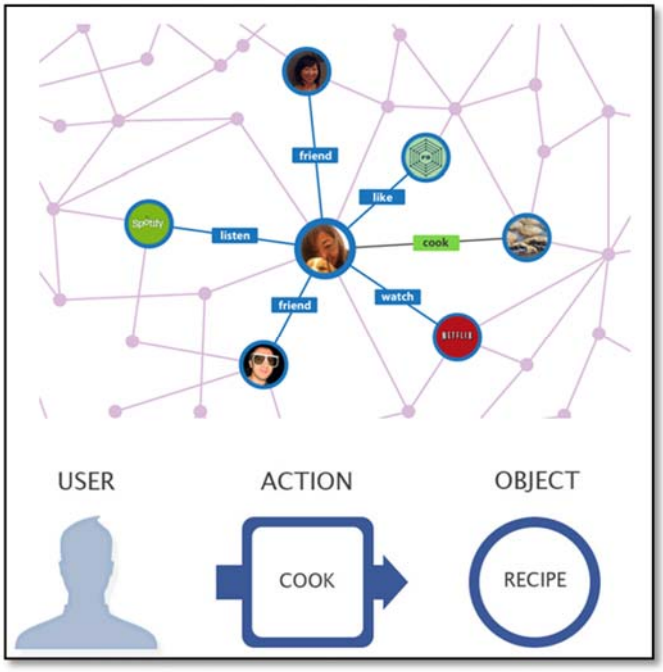

Figure 4.6- Facebook's Open Graph[30]

2. ***Services Identification and Grouping***: Facebook externalizes its functionalities through a communication layer composed by a set of *Application Programming Interfaces* (APIs). Such APIs forms the *Provided*

[30] Adapted from http://developers.facebook.com

Services that enables client applications to interact with Facebook programmatically via HTTP requests. It is also possible to use several methods in the *JavaScript* or mobile *SDKs* to build lots of other applications on top of this communication layer. This is what makes Facebook an open platform for development of interacting sociable applications on the Web. Regarding its services, in this analysis, we focused on the *Provided Services*.

Based on the kinds of information that Facebook deals with (step 1) and the functionalities it offers, we identified and grouped its *Provided Services*, and built a *mind map* with them, as showed in Figure 4.8.

5. ***Relationship Establishment***: In this step we analyzed how applications establish *relationships* with Facebook. Relationships between Facebook and other applications are of the type *registered relationship*. Hence, prior to accessing Facebook's services, developers need to perform a registration process in order to create the desired relationship between Facebook and his/her application. Thus, an application to interact with Facebook needs, at least:

   (I) ***To be registered in Facebook***: developers should fill out a form in the Facebook's website which requests basic information about the application, such as *App Display Name*, *App Namespace*, *App Domain*, *Category* and so on. Figure 4.7 shows an example of Facebook's App registration form.

Figure 4.7 – Facebook's app registration form

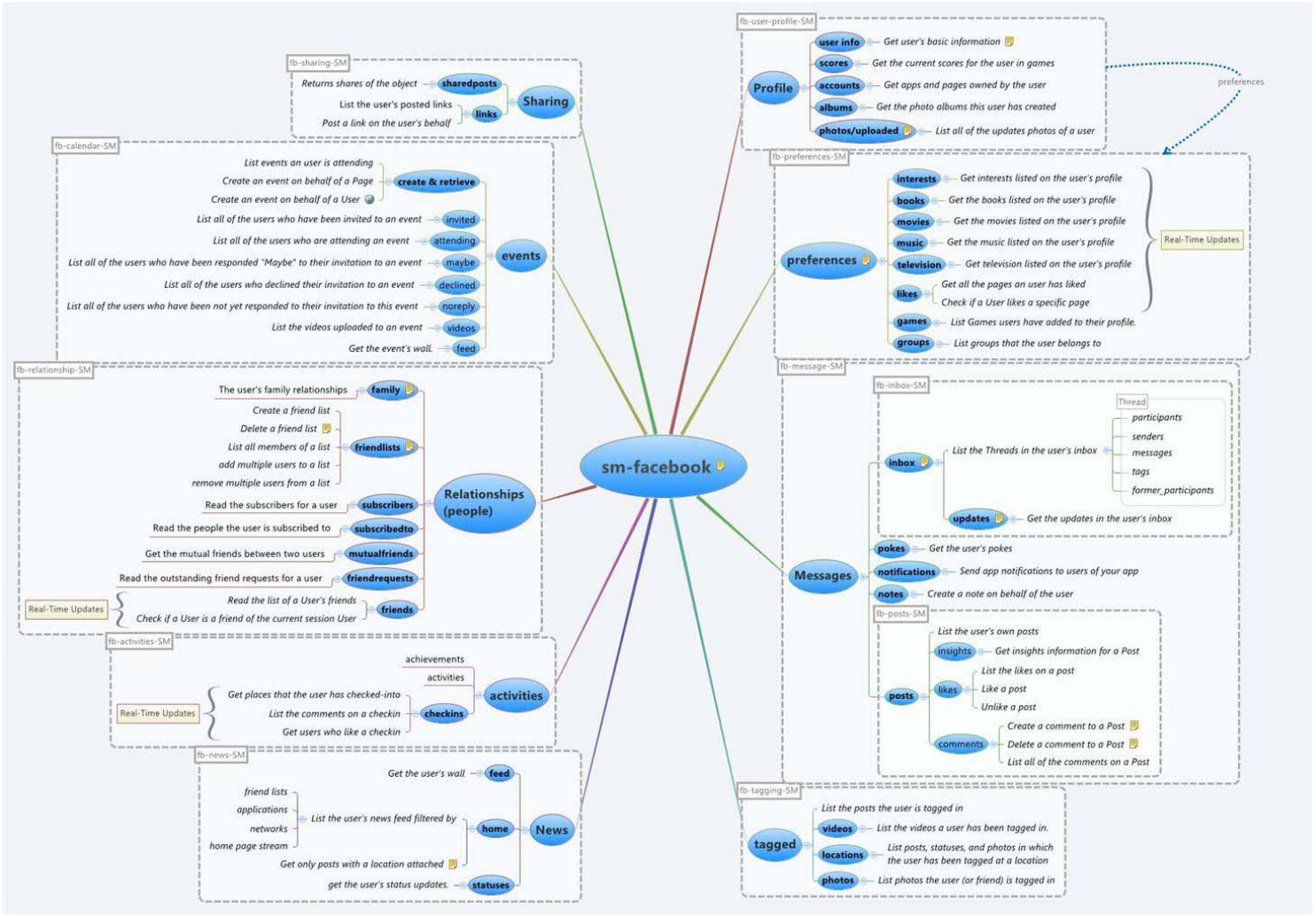

Figure 4.8 - Facebook's Services Analysis

(II) ***To have its permissions set up:*** besides register the application, developers should choose its permissions. *Basic permissions* are already established by default, i.e., the application will have access to only the user's basic information. Beyond that, any combination of *additional permissions* can also be set (Figure 4.9).

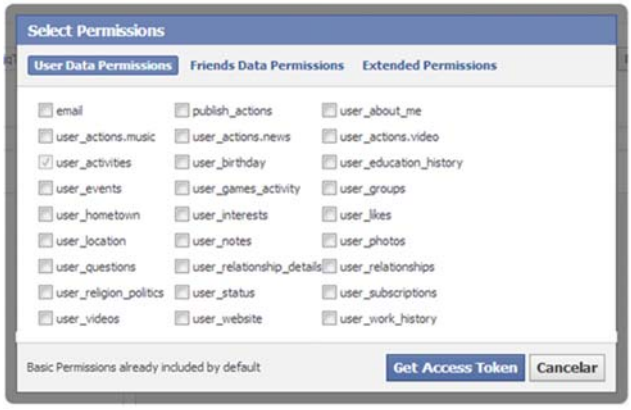

Figure 4.9 - Facebook's 'Select Permissions' screen

Doing these two tasks above a *relationship* is established between Facebook and an application. But, how such *registered relationships* determine 2^{82} interaction views?

6. ***Constraints & Interaction Views Definition****:* In Facebook, most of the restrictions used to control its interaction views (i.e. different sets of provided services) are of the type '*Visibility Constraints*'. These visibility constraints, in the case of Facebook, represent the different types of permissions that Facebook offers to characterize its *relationships* with other applications. The set of permissions is one of the properties of a *relationship* established between Facebook and third-party applications. Table 4.3 summarizes our analysis on all the permissions defined by Facebook, including the total number of available permissions grouped by their types.

Table 4.3 - Facebook's Permissions[31]

Type	Permission	Quantity
Default	Basic Information	1
Additional (total of 82)	User Data	27
	Friends Data	24
	Extended	25
	Open Graph	6

[31] For more details, access: http://bit.ly/fb_perms

Figure 4.10 illustrates how Facebook's relationships are used to determine the different interaction views accessed by third-party apps.

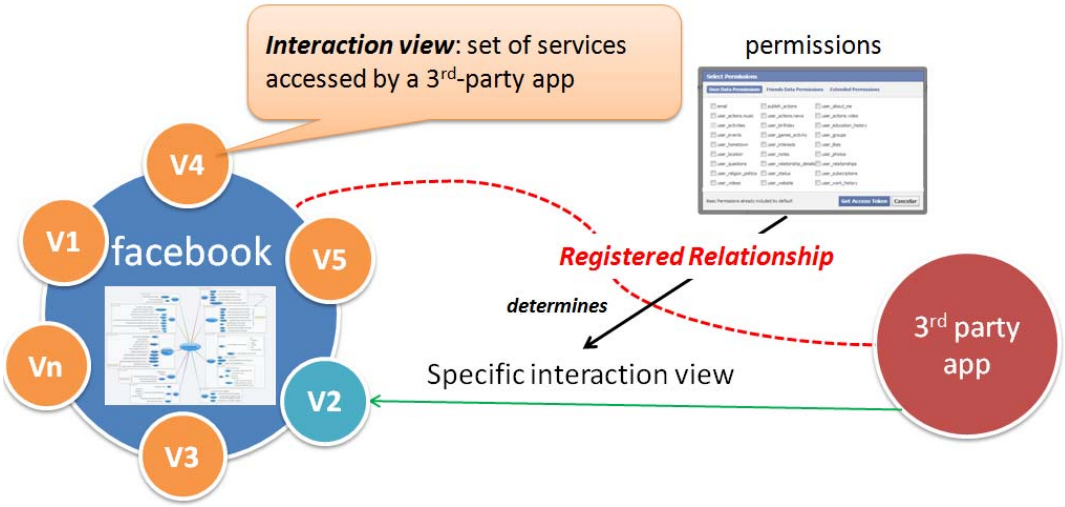

Figure 4.10 - Facebook's interaction views

In practice, beyond the default permission (basic information), each application can select any combination of additional permissions. Therefore, the total number of possible interaction views is given by the sum combinations presented in Figure 4.11.

$$v = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} \ldots + \binom{n}{n} + 1$$

Possible combinations of additional permissions

Default permission (basic information)

Where:

- v: number of possible views
- n: number of additional permissions

Figure 4.11 -Formula to calculate the total number of Facebook's possible interaction views

As can be seen in Table 4.3, the total number (n) of Facebook's additional permissions is 82 (n=82). Hence, using the combinatorial interpretation presented in Figure 4.11, the total number of Facebook's possible interaction views can be given by:

$$v = 2^{82}$$

Considerations

The analysis of *Facebook* as a Social Machine was very important in the process of characterizing Social Machines on the Web. It helped us to better understand the social aspects of a system that well represents this new generation of relationship-aware software. *Facebook* is not only a useful social network for its users, but it is also a platform for developers. Actually, *Facebook* is one of the most important general-purpose Social Machines (SHADBOLT et al., 2013) in the Web today, because it enables the formation of other Social Machines in the Web's ecosystem. The result of this analysis points not only to the relationship-awareness and "sociability" of the whole software, but also to the degree of sophistication and complexity that can be enabled by a certain family of Social Machines. In Table 4.4, the main abstractions of the social machine model are mapped to the Facebook's elements.

Table 4.4 - Social Machine's abstractions mapped to Facebook

Social Machine abstractions	Facebook
Information Processing System (IPS)	It consists of the Facebook's *Open Graph* (including operations on it) and the Facebook's internal functionalities.
Wrapper Interface	It is represented by *Facebook's* communication layer responsible for mapping requests and responses to the internal *Open Graph*' operations.
Relationship	*Facebook* knows the relationships among its users and also its relationships with third-party applications. The latter is obtained through a registration process and characterizes Facebook as a *relationship-aware* system.
Constraints	*Visibility Constraints* (i.e., permissions) are used to determine Facebook's interaction views with third-party applications.
Required Services	Not analyzed, because they are not exposed and we do not have access to Facebook internal code.
Provided Services	Set of *Facebook's* provided services (Figure 4.8).

4.8. Concluding Remarks

In this chapter, we established a unifying abstraction model for Social Machines that can be used for specifying what we refer to as *relationship-aware* applications and services. We outlined the notion of sociable software and [re]defined the semantics of *relationship* as a way to weave "sociability" aspects into software. Furthermore, we presented an engineering guideline for supporting the analysis of existing systems under the perspective of Social Machines.

Our abstraction model generalizes the traditional algorithmic Turing Machine model of computation (i.e., the element IPS) and provides a new conceptualization of computational phenomena which involve possibly related and interacting building blocks that make use of notions from *computing*, *communication* (in the form of *relationships* and *interactions*) and *control*.

In more than one sense, what we discussed in this chapter contributes to the process of blending computational and social elements into software, and further an attempt to give some foundations to systematically engineering the emerging social systems. Next chapter establishes the basis for the creation of SoMAr – an architectural style for the design of Social Machine.

5 Desigining Social Machines

"Every great architect is - necessarily - a great poet.
He must be a great original interpreter of his time, his day, his age."

Frank Lloyd Wright (1867– 1959)
American Architect, writer & educator

The previous chapters have laid the foundations for social machine. However, we have not explicitly presented *how* architectures of social machines should be designed. What are the constraints, principles and properties of such architectures? Which guidelines and patterns should be considered to derive systems as social machines?

Motivated by these questions, this chapter presents the third stage of our research which involves the design of systems as social machines. This chapter introduces SoMAr (***So****cial* ***M****achine-oriented* ***Ar****chitecture*) and characterizes its *constraints*, *principles* and *properties*. Furthermore, it also presents a lightweight design guideline in conjunction with a reference architecture and patterns as a way to support the process of deriving social machine-oriented architectures.

5.1. Research Guidelines

As shown in Chapter 1, our proposal includes four main stages and this chapter presents stage 3, which is referred to as "Designing Social Machines", as per depicted in our research roadmap. In this stage we began our study by looking in existing works for a common definition to the concept of *architectural style*. Then, in the first step we adopt a basic framework to guide our definition of the elements that compose an architectural style for social machines. *Brainstorming and focus groups* were performed during this stage. Furthermore, we used *personal opinion surveys* as an instrument for gathering information about projects built with our model of social machines. This includes projects developed during Graduate subjects and industry projects such as *Futweet*[32]. A system called *[YOU]* was built as a case study just after defining the guiding principles of social machines. Table 5.1 summarizes the research guidelines adopted in this stage of designing Social Machines.

Table 5.1 - Research Guidelines for Designing Social Machines

Stage 03: Designing Social Machines	
Goal	Design systems as social machines
Questions	How to design systems with the proposed paradigm?
Evidences	Architectural style (principles, constraints, properties); Design guidelines; Practical example (e.g., *[YOU]* is described in Chapter 6)
Research Methodology	Brainstorming and focus group; personal opinion surveys; Case Studies

[32] https://twitter.com/futweet

5.2. A Basic Framework for Defining Architectural Styles

One of the most challenging aspects of writing about software architecture is the adoption a common terminology. Often, many design-related terms suffer from wide-spread ambiguity, which sometimes makes difficult the use of a common vocabulary.

As one of the main goals of this chapter is to define an *architectural style* that guides the design of systems as Social Machines, we need to establish a common definition of the notion of *architectural style* prior to exploring the details of the style we refer to as Social Machine-oriented Architecture (SoMAr). Given that, we have to answer at least two questions: "*What exactly do we mean by an architecture style?*" and "*What are its key elements?*"

Based on our discussions and investigations on software architecture and architecture styles (GARLAN; SHAW, 1994)(SHAW; CLEMENTS, 1997)(TAYLOR; MEDVIDOVIC; DASHOFY, 2009)(FIELDING; TAYLOR, 2000)(BASS; CLEMENTS; KAZMAN, 2012), we can generalize the concept of an architectural style into the following common definition:

> *"An **architectural style** comprises a set of **constraints and principles** imposed on the design of a product to obtain **desired properties**"*

Figure 5.1 illustrates this definition and reinforces that a style limits/guides the development of a product with the aim of obtaining beneficial properties.

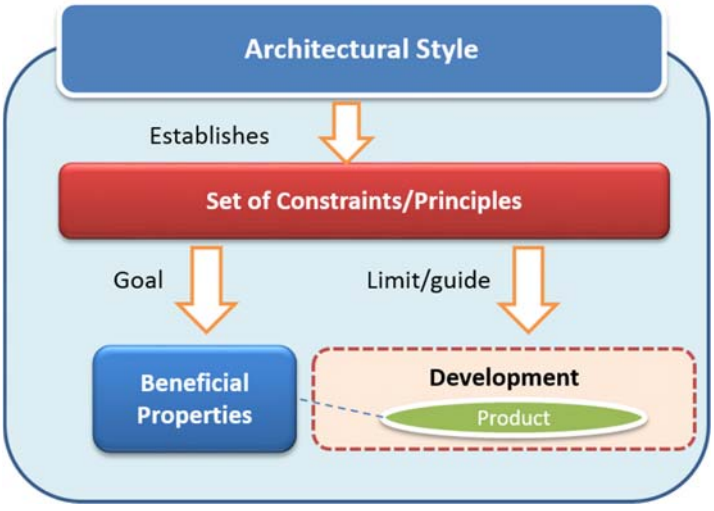

Figure 5.1 –Generic Definition of an Architectural Style

The aforementioned definition can be applied to several contexts, including building architecture, for example. To support this claim, Figure 5.2(a) illustrates the elements of this definition in the context of a *Swiss Chalet style*, which is an analogy from building architecture, used by a UC Irvine's software architectures group[33]:

"...buildings constructed in the Swiss Chalet style are constrained to have steep roofs. This constraint elicits a particular beneficial property: snow will slide off the roof, rather than building up crushing the structure."

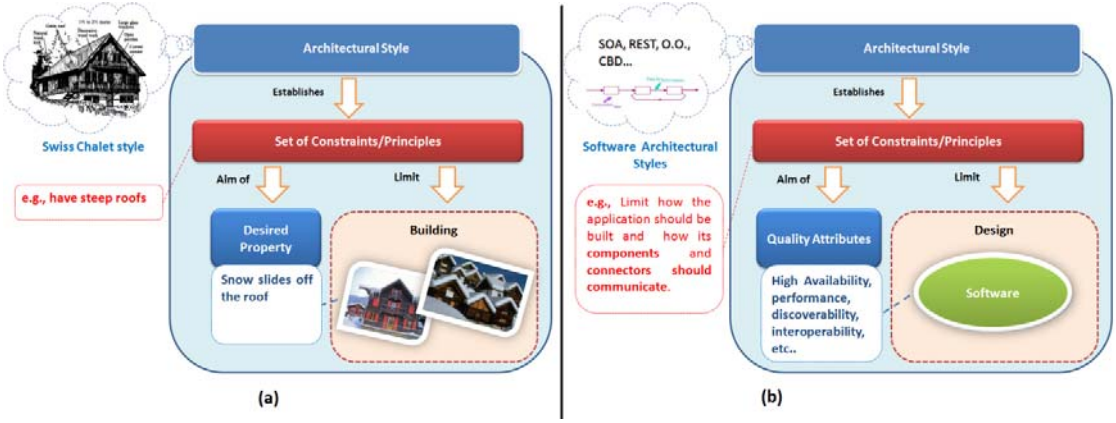

Figure 5.2 - Architectural style in different contexts: (a) – building architecture; (b) software architecture

As can be seen in Figure 5.2(a), the Swiss Chalet style has *steep roofs* as its basic *constraint*. This constraint *limits* the design of house architectures guided by this style with the aim of allowing snow slide off the root (*desired property*). In this way, it is worth noting that it is possible to have several distinct products (i.e., house architectures), yet following the same style.

This is exactly what happens in the context of software. Software architectural styles (Figure 5.2 (b)) define a set of *constraints* and *design principles* that limit the design of software with the aim of satisfying different *quality attributes* (*desired properties* such as high availability, performance, security and so on). A software architecture style limits/guides how a software system should be built and how its components and connectors should communicate. However it is also possible to have different (specific) system's architectures following the same software architecture style.

[33] http://isr.uci.edu/projects/archstudio/myx.html

As architectural styles define high level patterns and principles commonly used to develop applications, they include, for example, the different styles derived from software paradigms, such as SOA (*service-oriented architecture*), *component-based architectures*, *REST* and so on (see Chapter 2). Each of them with its own set of constraints/principles that imposed on the design of software obtains desired quality attributes. Table 5.2 lists some examples of what we consider architectural styles and organizes them by category, namely communication, deployment and structure.

Table 5.2 - Examples of architectural styles

Category	Architecture Style	Description
Communication	***SOA*** (ERL, 2005)	Refers to the architecture style of systems that expose and consume functionalities as services using contracts and messages.
	REST (FIELDING; TAYLOR, 2000)	Architectural style that uses existing HTTP methods as application protocol to apply CRUD operations (*Create*, *Read*, *Update* and *Delete*) to information resources defined by their URI.
Deployment	***Client/Server*** (SHAW; CLEMENTS, 1997)	It segregates the system into two applications, where the *client* makes requests to the *server*. It is possible to have variations on the client/server style including, for example, *Peer-to-Peer style* that allows the *client* and *server* to exchange their roles to share information.
Structure	***Pipes & Filters*** (SHAW; CLEMENTS, 1997)	It designs a system as a set of independent and stateless filters that transform input data stream into an output data stream and the pipes conduct such streams.
	Layered Architecture (SAVOLAINEN; MYLLARNIEMI, 2009)	It partitions the concerns of applications into stacked groups (layers). Each of them with its own responsibilities. *MVC (Model-View-Control)* is the most common layered architecture.
	Object-Oriented (TSAI; ZUALKERNAN, 1988)	An architectural style based on the division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.
	Component-Based Architecture (SZYPERSKI, 2002).	It decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.

As we adopted this notion of architectural style for defining SoMAr , next section provides an overview of this style, presenting some of its *constraints*, *guiding principles* and *desired properties*.

5.3. The SoMAr Architectural Style

The motivation for developing SoMAr (***So****cial* ***M****achine-oriented* ***Ar****chitecture*) is to create an architectural style that could serve as an abstracting framework for guiding the design of Social Machines as a proper family of systems. This section makes clearer the *constraints*, *principles* and *desired properties* considered by SoMAr.

First of all, it is important to note that SoMAr is a *hybrid style,* which means it combines principles from other existing styles as a way of yielding more powerful design. Actually, some software paradigms such as *object-oriented, service-oriented and component-based paradigms* provide us with substantial practices that can successfully support the identification and description of appropriate abstractions within architecture. However, according to Taylor et al. (TAYLOR; MEDVIDOVIC; DASHOFY, 2009) existing experiences show that these practices are rarely applied independent from each other, mainly when more complex systems are considered.

Supporting this thought, J.D. Meyer (MEYER et al., 2009) states :

> *"...the architecture of a software system is almost never limited to a single architectural style, but is often a combination of architectural styles that make up the complete system. For example, you might have a SOA design composed of services developed using a layered architecture approach and an object-oriented architecture style."*

In fact, there is a great number of architectural styles, whilst a high-quality software design inevitably includes more than one style. As a consequence, during the definition of *SoMar* we investigated suitable elements from OO, SOA, REST and other existing styles in order to support our motivation for creating a hybrid one. The resulting, style has successfully

facilitated the design of social machines. As illustrated in Figure 5.3, ***SoMAr[34] is the combination of different principles from existing styles, constrained by the unified vision of Social Machines***.

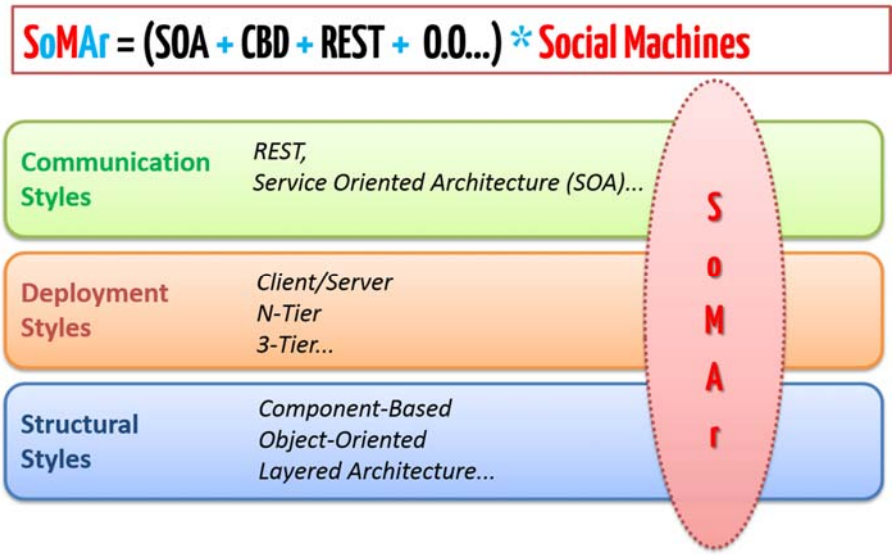

Figure 5.3 - SoMAr: Social Machine-oriented Architecture

The *unified aspect* of the Social Machine paradigm is accomplished by both the SoMAr's combination of principles from existing styles and the unified arrangement of elements of its building blocks (i.e., the Social Machine model discussed in the previous chapter), once such building blocks also embody elements from services, components, REST, and other paradigms to wrap any kind of *socially connected computing unit*.

5.3.1. Constraints

The Social Machine model, introduced in Chapter 4, is the architectural building block native to our proposed style. It is important to have its elements in mind in order to better understand the SoMAr's constraints. Social Machine is not only a possible foundation for describing the emerging *socially connected computing units* but it also should provide some general constraints to guide the design of such units.

Hence, systems should be designed as networks of related Social Machines in SoMAr as follows:

[34] SoMAr lives up to its name, once the word "somar" in Portuguese means "to sum", what makes sense to think about a hybrid style that combines different principles from other existing ones.

- **Sociable**. By the very nature of the concept we are proposing, SMs are sociable stuffs and, in nearly all cases, each one should provide means to make *relationships* (see Chapter 4) and interact with one another. The isolated Social Machine is an exception.
- **Identified & accessible.** Any Social Machine on the Web is an entity at some URI and can have its services accessed in a standard way (e.g., via HTTP operations).
- **Autonomous.** Each Social Machine can be maintained, developed, deployed, and versioned independently.
- **Composable.** Social Machines can be composed by other Social Machines. This can be obtained by integrating data from different Social Machines or by connecting their services together, i.e., connecting the provided services of one Social Machine to the required services of other.
- **Loose coupled.** Each Social Machine should be independent from each other, and can be replaced or updated without breaking applications that use it as long as its "provided services" are still compatible.
- **Encapsulated.** Social Machines expose services that allow the caller to use its functionality, but such services should not reveal details of the wrapped computational unit's processes, internal variables or state.
- **Extensible.** A Social Machine can be extended from existing Social Machines to provide new behavior.
- **Highly cohesive.** Well-defined responsibility boundaries for each Social Machine, and to ensure that each SM contains functionality directly related to the tasks it is responsible for. This fact helps to maximize cohesion within the Social Machine.
- **Not context specific.** Social Machines are designed to operate in different environments and contexts. Specific information, such as state data, should be passed to the Social Machine's services instead of being included in or directly accessed by the Social Machine.
- **Reusable.** Social Machines should be designed to be reused in different contexts by different systems.

Figure 5.4 presents the constraints defined by the *SoMAr* architectural style and illustrates how they are mapped into some popular existing architectural styles, i.e., styles derived from the paradigms discussed in Chapter 2. It is worth noting that a constraint can simultaneously belong to more than one architectural style and in Figure 5.4 they are positioned in the place they better fit into. For example, "*loose coupled*" is a typical constraint from SOA and CBD, and "*reusable*" is a constraint almost equally considered on the development of building blocks of all the listed architectural styles.

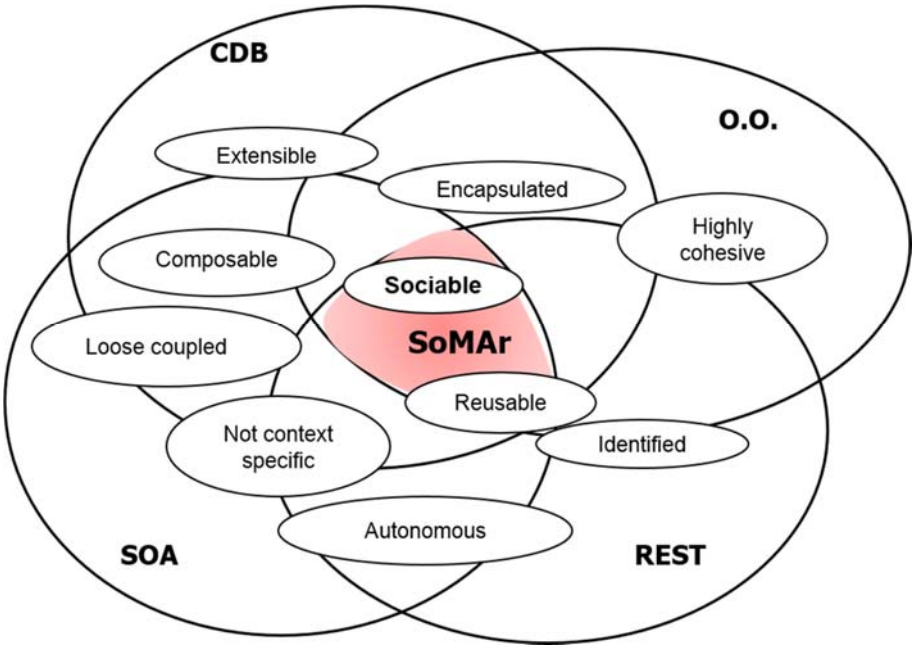

Figure 5.4 – SoMAr's constraints

Another important point is that such list of general constraints serves as guideline and it is not fixed or fully comprehensive. It means that variations of constraints from other styles can also be added to such list in order to instantiate different software architectures. For example, considering the *client/server* deployment style (Table 5.1), it is possible to build a *Social Machine-oriented architecture* in which the socially connected entities can act as servers (*providers*), clients (*consumers*) or even both (*prosumers*[35]). Prosumer SMs support the peer-to-peer architectural style, which is a variation of the client/server style, to enable the deployment and management of multiple

[35] "Prosumer" is a term originally proposed by Alvin Toffler (TOFFLER, 1980) to designate someone who blends the roles of "consumer" and "producer". Thus, this term is used herein to refer to connectable entities that are capable of both consuming as well providing service.

federations of Social Machines. A multi-layered style can also be used to support the definition of a *reference architecture* (introduced in Section 5.5) for Social Machine through the combination of further styles. Next, some guiding principles are provided.

5.3.2. Guiding Principles

Social Machines presents a vision of a world in which systems are cleanly partitioned and consistently represented as connectable and sociable entities, and the following principles should also be considered for guiding the design of such entities:

Embracing relationships

Social Machines is about embracing different kinds of *relationships*. For many years, connected entities have been accommodated inside information processing systems. Relationships do exist in the core of such systems, but often only as a means of defining dependencies among them. In our discussion about '*relationship-aware*' software in chapter 4, we mentioned the need to disambiguate the semantics of relationships that connect entities and to qualify their capacity to determine different levels of interactions between them. Like users, we can understand semantic dependencies between entities, but software—the building blocks themselves— are "blind" to these relations, i.e., they are not aware about them. Social Machine-oriented systems must be aware of their relationships with others and create a way to deal with them in order to infer the different sets of interactions that should take place among a plurality of socially connected entities.

Neutrality

A Social Machine should provide its services in a way that SMs using such services do not take care about how they were implemented, whilst it should define services with interfaces that also abstract away application-specific details. It means that beyond implementation neutrality the SM's services should emphasize *application neutrality*. Application neutrality enables *shareability* by providing a generic and application-neutral protocol. It is desirable that SMs use well defined and de facto standard protocols in such way that the communication between them is as simple as possible. REST is an

example of an existing style that implicitly makes use of such concern by providing an *application protocol* build upon the core HTTP methods which, by their turn, are used with associated semantics. *Atom Publishing Protocol* (APP)(SAYRE, 2005) and *Google's gdata Protocol*[36] are examples of generic, application-neutral protocols. By designing Social Machines we should not only put emphasis on implementation neutrality, but also on the generic interface constraint of application neutrality which should be the main goal of the definition of a service interface. Social Machines' *provided services* should be legible, neutral and consequently easy to use; the clearer a SM is in providing access to its services, the easier will be for developers use their services.

Transparent Blending

As aforementioned, Social Machines integrate both social and computational processes. However, to be considered a Social Machine this blending should be transparent as much as possible. We could realize such fact during the *First International Workshop on the Theory and Practice of Social Machines* (SOCM2013). At that time, we presented the concept of "*social slide*" - a slide connected to the Web that blended two existing systems, i.e., a slide show presentation program and a polling service. Such "*social slide*" had an URL that allows users to submit votes/comments to the slide and change its content on the fly. The "*social slide*" was indeed a transparent way of blending a *computational process* (i.e., the process of computing users' input to automatically update the slide's content) and a *social process* (i.e., the process of directly "co-authoring" a shared content). During the discussion, 80% of the audience (more than 40 experts) answered "Yes", when asked whether the "*social slide*" could be considered a kind of Social Machine or not. The main reason for that was the transparent blending between the two existing systems, i.e., the *slide show presentation program* and *polling service*. Such transparent blending gave the impression that the slide was a single *socially computing unit* that updated itself in response to the interactions of the audience. Based on this result, we concluded that sometimes the degree of novelty (innovation) of something relies (almost exclusively) on the way it blends existing thing. Maybe it is the case of both our proposed unified model and architectural style that in

[36] https://developers.google.com/gdata/

more than one sense blend concepts from existing approaches in order to establish a new one.

5.3.3. Obtained Properties

As stated in Section 5.2, an architectural style comprises a set of *constraints* and *principles* imposed on the design of a product to obtain *desired properties*. We discussed the general set of *constraints* (Subsection 5.3.1) and guiding principles (Subsection 5.3.2). However, which are the obtained *properties* when applying such constraints? In order to answer this question, we realized some practical experiments in which developers used our proposed approach and answered an opinion survey (as presented in Chapter 6). We collected a set of benefits that could be observed by developers during the development of the different case studies, and we can highlight the following cited properties:

- ***"Sociablility"* as a quality attribute of software**. By creating a Social Machine implies much more than just connecting software to the Web, it allows the creation of software that accounts for the fact that it will exist in a world of multiples peers. Then, a Social Machine design places the idea of *sociability* as it core. By "sociability" we mean the ability of a system to "socialize" with others, which could involve four things:

    1. Ability to establish (different) relationships;
    2. Ability to recognize its relationships with others;
    3. Ability to provide different interactions according to the relationship with the peer it interacts with; and
    4. Ability to disengage from a relationship (i.e., break it up).

  In our exercise to describe *Facebook* as a Social Machine with 2^{82} *interaction views* (Chapter 4), it was possible to realize how the different types of relationships established with third-party applications can impact the number of interaction views. We believe that to define "*sociability*" as a software quality attribute such as performance, security and others, is important to the success of any social system on the Web. In practical terms,

sociability can be a *composed quality attribute* of software, which can be broken down into other attributes like *reusability* and *adaptability*.

- **Abstraction.** SoMAr can abstract the view of a system as whole while providing enough detail to understand the services and responsibilities of individual Social Machines and the relationships between them and other entities. This allows a reduction of complexity into a generalization that retains the base characteristics of its core functionalities.
- **Understandable.** It maps the application more closely to the socially connected real world entities that make relationships and interact, making it more understandable.
- **Interoperability.** Given their neutrality and the adoption of standard protocols and data formats, the provider and consumer of Social Machines' services can be built and deployed on different platforms.
- **Ease of deployment.** As new compatible versions become available, one can replace existing versions without impacting on neither the other Social Machines nor the whole system.
- **Ease of development.** Social Machines implement well-established interfaces to provide defined services by allowing the development without impacting other parts of the system.
- **Reusability.** The idea behind Social Machines is to take advantage of the networked environment where they are to make easier to combine and reuse exiting services from different SMs and use them to implement new ones. The use of reusable Social Machines means that they can be [re]used to spread the development and maintenance cost across different systems or applications.

Having the aforementioned principles, constraints and properties in mind, next section presents a guideline for supporting the design of Social Machines.

5.4. Design Guideline

As a software development guideline for designing Social Machine-oriented architectures, we have considered the steps illustrated in Figure 5.5 as the basis for the design exercise of Social Machines.

Figure 5.5 - Design Guideline

1. Define building blocks

The 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 designing the whole Social Machine as a single social service component and parts of the Social Machine (e.g., its modules, subsystems and participants) as relationship-aware entities. In practice, any entity that has the potential to establish relationships with others and interact according to such relationships can be wrapped up into a set of specialized APIs and be defined as a relationship-aware entity. These entities include, for example, a source of information, a stateless service, a collection of socially wrapped entities and 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 indeed make sense to involve with a "social layer". This layer should allow the creation of an independent and autonomous entity capable of establishing relationships with others to define its different interaction views. During the design process, each SM should have a unique identifier, often a URI, which is used as basis for accessing its provided services/APIs.

2. Specify services

During this step, the set of services to be provided by each Social Machine (identified in the Step 1) needs to be designed. High-level design as well as significant parts of the detailed design of the services provided by each Social Machine is included in this step. The Social Machine's services can be specified in terms of *endpoints*, its relative URIs, type of request (*GET, POST*, etc), possible parameters and description. Often a common URI syntax is adopted to identify Social Machines and its services.

Table 5.3 shows an example of the set of services provided by a Social Machine called *CalendarYOU* which wraps users' agendas on *Google Calendar*[37].

Table 5.3 - Simple example of sevice specification

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

3. Design integrations

Generally, the process of composing Social Machines deals with both structured and unstructured data from multiple 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 Social Machine manipulation.

In this step, a common view of relevant data - that occurs when designing Social Machine-oriented applications - should be defined. Often, a diagram containing the set of abstract data type is enough to define the kind of

[37] www.google.com/calendar

data to be manipulated by each Social Machine. Once the abstraction 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 architecture patterns to deal with integrations issues, as will be shown in the reference architecture depicted in Section 5.5.

4. Design interaction models

To define how the components and actions that make up a system interrelate 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, designing 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. For example, it includes the definition of communication protocols and how such parts establish *relationships* to interact with each other.

Social Machines define different interaction models, but some patterns of interactions can be observed and are worth highlighting. One of these patterns is what we call *the "love triangle"* interaction model, which is accomplished by an authentication model involving the Social Machine, its users and third-party applications.

The "love triangle" model

The *"love triangle"* model is a generalization of the OAuth model (LEIBA, 2012) and defines three roles, namely: *API Provider* (the Social Machine), *Data Owner* (Users) and *API Consumer* (Third-party Applications), as illustrated in Figure 5.6.

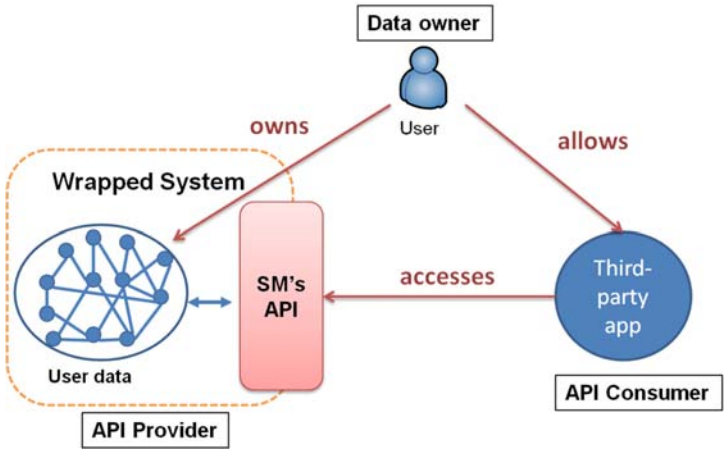

Figure 5.6 - The "love triangle" interaction model

As can be seen in Figure 5.6, the Social Machine 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 Machines and some credentials (e.g., username and a password) should also be established in advance.

In this model, the same registered application can be used by different users. Actually, 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, instead of using its credentials, an application should use the data owner's credentials to make requests - pretending to be the user.

The "*love triangle*" interaction model is commonly used when a Social Machine exposes users' data in the form of a platform of *Application Programming Interfaces* (*APIs*) to be accessed by other applications. In this case, 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* (LEIBA, 2012), which recently have gained wide popularity on the Web. Next, we describe a reference architecture that combines different design patterns in order to support the design of systems as Social Machines.

5.5. Reference Architecture and Patterns

A *reference architecture* can be extracted from practical developments on applying architectural styles and acts as a template solution for the design of

architectures of related applications that possibly belong to a certain domain or family of systems. In practice, a reference architecture has substantial knowledge, acquired through the use of a set of design decisions and patterns to structure different applications within a particular domain (TAYLOR; MEDVIDOVIC; DASHOFY, 2009).

Hence, based on practical experiences and case studies in the context of social machines, it is possible to generalize the set of adopted solutions with the aim of defining a higher level architecture that can be used as reference for the design of social machines. Motivated by this fact, we took into account our different development experiences with social machines (BRITO et al., 2012; BURÉGIO et al., 2014a, 2014b, 2013b; MEIRA et al., 2013, 2011; NASCIMENTO; GARCIA; MEIRA, 2012) as well as the practical use of architecture integration patterns (LIU et al., 2009), with the aim of defining a reference architecture for social machine that integrates the common set of patterns we have been used in a number of successful implementations.

Figure 5.7 illustrates the obtained high-level architecture and shows how it integrates *Model-View-Control (MVC)*, *Pipes & Filters* and *Data Federation* patterns. This conceptual level architecture is a technology-independent reference that can be used as a template for instantiating concrete architectures of both single and composite social machines.

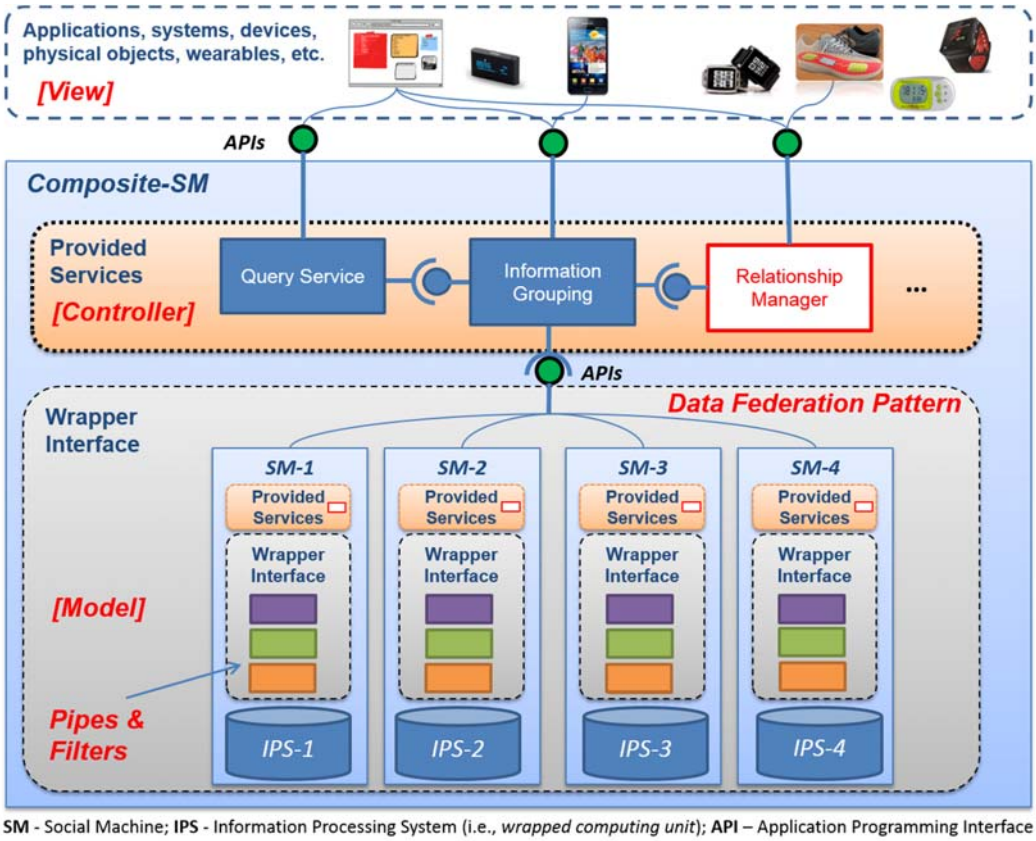

Figure 5.7 - Reference Architecture for Social Machine

5.5.1. Model-View-Control

The reference architecture for social machines depicted in Figure 5.7 follows the *Model-View-Control* (*MVC*) pattern, which divides the system into three interconnected layers, namely *Model*, *Controller* and *View*. The first two are part of the social machine's structure and the latter is a layer that groups the applications that use (or are built on top of) the social machine's provided services.

5.5.1.1. Model

The *Model* uses the element *Wrapper Interface* to getting access to data obtained from the wrapped *computing units* (IPS in Figure 5.7) and making these data available to the SM's provided services. It is worth mention that in the case of *composite social machines*, the wrapped *computing units* can also be represented by a set of individual SMs, such as SM-1, SM-2, SM-3 and SM-4 (see Figure 5.7). In this case, the *Data Federation Pattern* is used to aggregate data and make them available to the composite-SM's *controller*.

5.5.1.2. Controller

The *Controller* represents the layer in which the SM's provided services are placed. It uses the data delivered by the model layer and provides a set of software components that, among other things, are responsible for implementing both the SM's business logic and the intrinsic features of a social machine like its *relationship-awareness* capability.

Components in charge of the SM's business logic depend on the business domain the social machine belongs to. For the sake of simplicity, Figure 5.7 shows only two examples of these components, namely *Query Service* and *Information Grouping*, which are often implemented when developing composite SMs that deal with a huge amount of data extracted from multiple different sources.

Relationship Manager

Among the components in charge of implementing the intrinsic features of a SM, we can highlight the *Relationship Manager*. As aforementioned in Chapter 4, *relationship* is the key element of the social machine model. In fact, a SM can establish a lots of different *relationships* with a multitude of applications, systems, services, people, physical objects and other *socially connected entities* that [re]use the SM's provided services. The *Relationship Manager* is an important component of this reference architecture as it is responsible for realizing part of the *relationship-awareness* aspect (and consequently "sociability") of a SM. This component should provide mechanisms to mediate the establishments of relationships between the SM and others.

As presented in Chapter 4, *relationships* can be inferred (automatically established) or *registered* (manually established through a registration process). In practice, most of the current social machine platforms use the latter type to allow third-party applications to interact with their services (BURÉGIO et al., 2013b). It means that prior to accessing the SM's services, developers need to perform a registration process in order to create the desired relationship between the SM and his/her application. In terms of design, the steps of a possible registration process to be implemented by the *Relationship Manager* are illustrated by the sequence diagram in Figure 5.8.

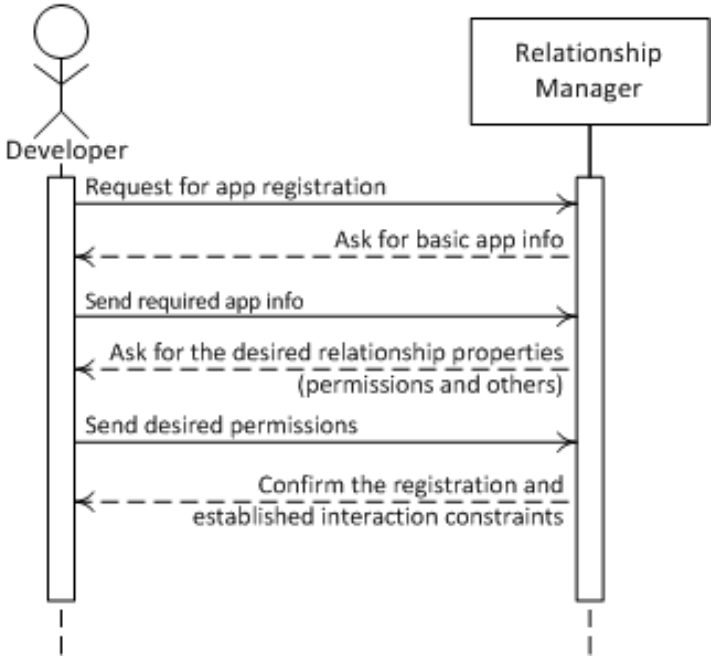

Figure 5.8 - Process of establishing a relationship

As can be seen in this figure, during the registration process, developers should provide information about his/her application and the properties that will characterize the relationship to be established between the application and SM. These properties will define how should be the interactions between the two parts, and they can include, for example, desired permissions, rate limiting, payment terms (when applicable) and others.

5.5.1.3. View

Finally, the *Viewer* layer represents the plurality of *socially connected entities* (including other social machines) that make use of the SM's provided services or even are built atop of them. As can be seen in Figure 5.8, these entities comprise different applications, systems, wearables, physical objects and so on. Each of them with a specific kind of relationship to determine *what* and *how* they interact with a given SM.

5.5.2. Pipes & Filters

In our design guideline (Section 5.4), we mentioned that the process of wrapping an *Information Processing System* (IPS) possibly involves a need to deal with a variety of data and formats manipulated and provided by such IPS. Given that, it is worth noting that to support the set of SM's provided services requires the development of *Wrapper Interface* (WI) that are in charge of

collecting and converting data provided by the *Information Processing System* (IPS) to be wrapped. Figure 5.9 illustrates how the WI can be designed.

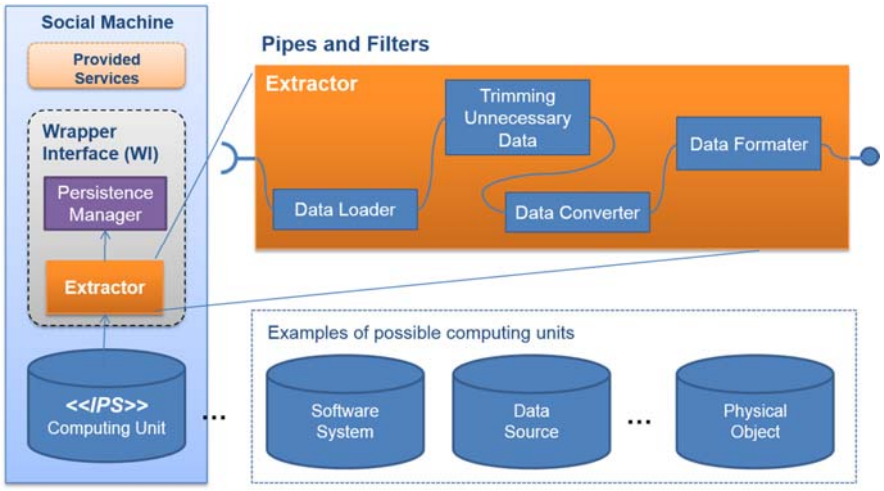

Figure 5.9 – *Wrapper Interface* designed as a set of Pipes & Filters

As can be seen in this figure, the element *WI* uses *Pipes and Filters* as an integration pattern to create the logic for both data conversion and combination. A WI should consist of a set of interconnected components that perform specific tasks such as loading data from the wrapped IPS, filtering unnecessary data, and formatting them to the desired representation.

The *extractor* component is responsible for converting the wrapped datasets into the format required by the component that encapsulates the persistence logic to be provided to the SM's controller layer, namely *Persistence Manager* (PM). PM is used to support the SM's provided services by retrieving and updating specific data types to be handled by the SM's business components. It is important to note that such set of internal components serves as an initial reference and can be customized, according to the specific needs of the concrete architecture to be instantiated.

5.5.3. Data Federation

The Data Federation style is used in the social machine's reference architecture as a way of aggregating and correlating the necessary data from multiple sources. As can be seen in Figure 5.8, in the case of composite social machines, the data federation should be realized by combining the set of parallels *pipes and filters* defined by each individual SM. In this pattern, a single source of data (e.g., IPS) remains under control of an individual social machine that can

asynchronously pull data on demand for federated access. This process of federating data is also supported by a specific component within the *controller layer*. In Figure 5.8 this component is represented by the *Information Grouping* component.

5.6. Concluding remarks

In this chapter, we provided a synopsis of the key constraints, principles and properties pertaining to social machine-oriented architectures. We introduced SoMAr as a social machine architectural style that combines different aspects from existing styles as a way of guiding the design of systems under the perspective of our social machine model.

The design guidelines used to develop systems as social machines were outlined in this chapter. Furthermore, we presented a high-level conceptual architecture that combines different patterns and serves as a reference architecture to derive both single and composite social machines.

To sum up, the central idea of SoMAr revolves around the notion of social machines as a unified way to design the wrapping of any computing unit that can be engaged in relationships and interactions with others. In practice, a Social Machine enables such relationships and interactions with a variety of applications, systems, people, physical objects, wearables and other *socially connected entities*.

The next chapter presents a discussion through the experience and lessons learned from applying our approach to the implementation of practical cases social machines in different contexts.

6 Experience & Evaluation

"Practice is the frequent and continued contemplation of the mode of executing any given work..."

Marcus Vitruvius Pollio (1642 – 1727)
Roman author & architect

We have been used Social Machines as a paradigm to guide the analysis, design and implementation of several emerging Web-enabled social systems. This chapter describes some of our implementation experiences and lessons learned from applying Social Machines paradigm in different contexts. First of all, we present the main stages of our evaluation approach, outlining their main goals, context and methodology. After that, for each stage, we present details of the different experiences through a qualitative discussion about the obtained results.

6.1. The Evaluation Process in a Nutshell

We used a four-stage approach to implement and evaluate the Social Machine paradigm proposed herein. Our approach can be divided into the following main stages: *i) Preliminary Experience*, *ii) Case Studies* and *iii) Opinion Survey*. Figure 6.1 summarizes the *goal*, *methodology* and *context* considered in each stage.

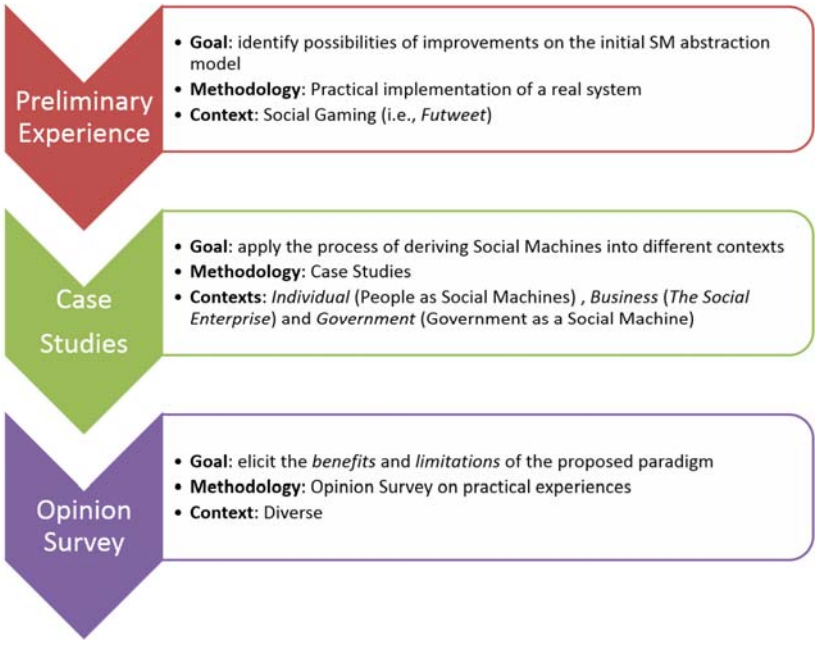

Figure 6.1 – 3-stage approach to evaluate the Social Machine paradigm

Next Sections look deeply at each stage and present details of the obtained results.

6.2. Preliminary Experience

In this section we describe the implementation of a seminal system, namely *Futweet*, developed using the initial ideas of Social Machines. Although there has been a number of developments on the model of Social Machines used at that time, *Futweet* still represents a relevant starting point to identify some preliminary lessons learned about the use of the Social Machines paradigm in the implementation of a real systems from scratch.

6.2.1. *Futweet*

Futweet is both a social network and a guessing game about football (soccer) results. Initially developed for Twitter users, Futweet was subsequently connected with other online social networks, e.g. *Facebook* and *Orkut*, making it a good initial case study for illustrating the development of an application that uses the concept of Social Machines.

Futweet is a social game originated from the idea of developing a Social Machine using the features provided by *Twitter*, which is a paradigmatic example of a Social Machine. The game illustrates the development of a real Social Machine, since it was designed and built to be networked with other applications and be itself a connection point of other applications and services. The social game was designed as a network of the related machines *Twitter*, *Orkut*, *Facebook*, *Gmail* and *MSN*. Figure 6.2 shows all Social Machines that comprise the Futweet system and their relationships.

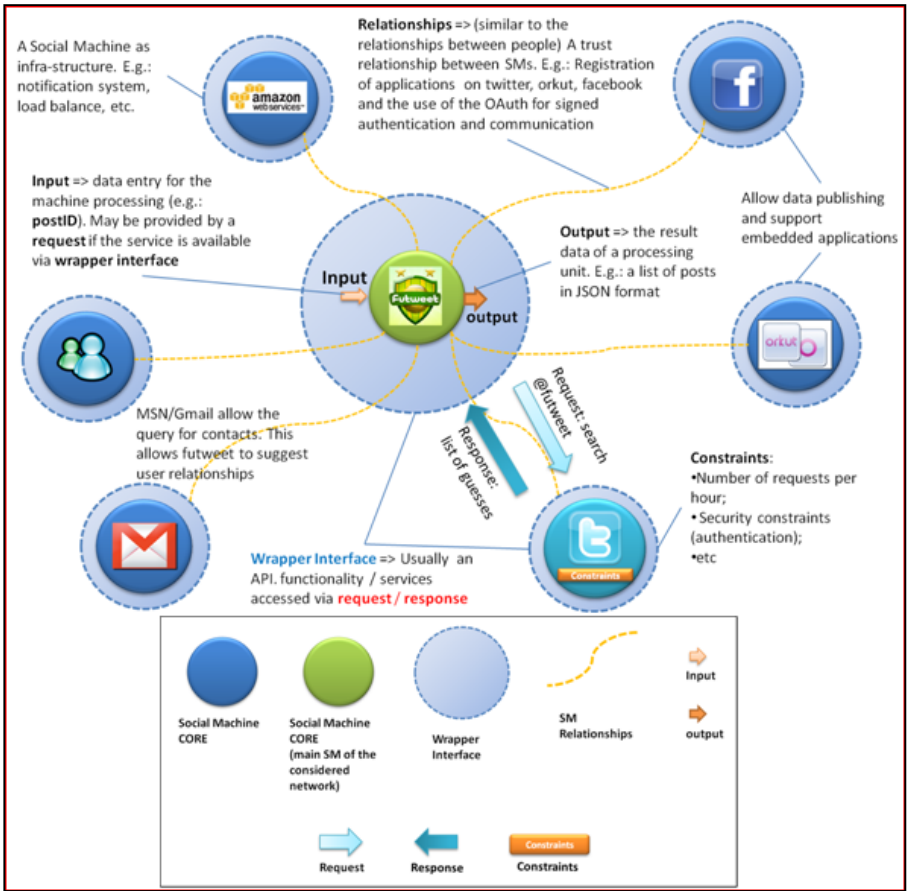

Figure 6.2 – Futweet as a network of Social Machines

Futweet is, of course, part of a network; in it, we can map the main abstractions of the Social Machine model (see Chapter 4) into the elements presented in the Futweet system, as shown in Table 6.1:

Table 6.1 – Social Machine abstractions mapped to Futweet

Social Machine abstractions	Futweet
Information Processing System (IPS)	It consists of the Futweet's business rules, i.e., its core and internal functionalities which are wrapped.
Wrapper Interface	It is represented by *Futweet's* communication layer responsible for mapping requests and responses to the IPS's inputs and outputs, respectively. It deals with data from interactions with Twitter, Gmail and MSN, for example.
Relationship	*Futweet* has relationships with other online social networks. These relationships define *persistent connections* with components or services that can be considered part of the social game network. If any of these SMs are unavailable, *Futweet* as a whole may be affected.
Constraints	Futweet has what we call "*quality constraints*" (see Section 4.5.3). Some of them are similar to the Twitter API rate limiting. So, when using an application built on the Futweet API, it may run into a situation where it is rate limited, i.e., unauthenticated calls are permitted 100 requests per hour. Futweet also limits request per account and IP.
Required Services	Set of services the Futweet needs to work properly, e.g., the search service provided by Twitter.
Provided Services	*Futweet's* specialized API, which encapsulates the main features of the game available on the Web.

Mechanism of Communication

Sending guesses

The basic mechanism of the game is to send guesses on soccer matches in a given league; such guesses are processed and compared with a set of pre-

established scoring rules and the winner is one who gets more points at the end of a specified period of time, which generally coincides with the end of the league championship. In the case of Twitter, to send the guess of a match follows a pre-defined syntax that has the team's acronyms and predicted scores as follows.

@futweet < TEAM 1 Acronym> <Score for TEAM1> X <Score for TEAM2> <TEAM2 Acronym>

Searching for guesses

Futweet has also an engine that periodically searches for *tweets* matching the pattern, extracts the information that represents the guess of a user and then recalculates the overall rank. Since *Futweet* also exists as embedded applications in *Facebook*, a user can request data (e.g., a ranking list) to the *Futweet's* apps on top of the *Facebook's* platform. *Futweet* is a Social Machine of class *prosumer* is an example of how Social Machines can work together to *receive*, *compute* and *present* information.

Infrastructure

Futweet does not own the servers it runs upon and its infrastructure is provided by Twitter (hunches in the form of tweets) and *Amazon EC2*. Hence, the social game is an application fully provided, designed, implemented and available in the cloud. This reinforces the assertion that the fundamental component of a Social Machine (its computational unit) and its (possibly many) other components can be supported by other, existing, Social Machines, resulting in a network which is, by itself, the desired application.

The functionalities of this network are encapsulated by a *wrapper interface* and a set of APIs (*provided services*) that make the main features of the Futweet available to other applications. It is worth observing that the *Futweet* serves as the "glue code" between different Social Machines.

Design Issues

During the design phase of Futweet as a Social Machine, it was necessary to consider a set of questions in which the answers had influences in the development of the social game itself:

- *Are there any available Social Machines on the Web that could be (re)used by the project?* Building a Web application as a SM should consider the existence of other machines to be (re)used. In the case of *Futweet*, already existing machines considered were*: Twitter, Amazon AWS, Gmail, MSN* and, thereafter, the online social networks *Facebook* and *Orkut*.

- *What does the Social Machine provide for its environment (Web)? Futweet* is one of several implementations of a soccer guessing game. However, it provides mechanisms through APIs to allow users to use its platform to create their own applications of guessing game, and allows the entertainment of Twitter users by extending the capabilities of Twitter through the addition of a new service.

- *What are the (read/write) operations provided by the application?* Social machines may have different social levels that vary according to *i)* the connection they have with other machines and *ii)* the type of operations enabled by these connections. As mentioned before, *Futweet* is a *prosumer* Social Machine. It has connections to read/write on Twitter (read and put data in the social network) and allows the same operations through its own API (users as well as third-party applications can remotely post on and read data from *Futweet*).

By answering these questions, the implementation of *Futweet* consisted of designing a set of interfaces to access various Social Machines, governed by business rules (from the social game) that implied the functionalities and design of an API, on the top of which the application (website) was also built. This simplistic view of *Futweet* was important for understanding the concept of Social Machines.

6.2.2. Discussion

With this initial development of the *Futweet* system we identified several factors that should be taken into account when developing a Social Machine; One has to bear in mind that the complexity of a given system's development is directly related to the properties, power, limitations and restrictions of other Social Machines considered in the project. Non-functional requirements such as

response time can be affected by quality attributes of SMs being used as a basis for the design and implementation, such as availability, limitations or restrictions of third-party APIs, changes in the mechanisms for accessing Social Machines, and so on.

We were aware that *Futweet* was a "toy" project compared against actual corporate projects. However, by comparing the effort needed to design it from scratch against one using our approach, it is clear that Futweet is not small project anymore. Futweet puts together a lot of stuffs already provided by existing Social Machines available in the Web.

Improvements on the initial SM abstraction model

The *Futweet* experiment is a milestone of the transition from the initially adopted Social Machine model (MEIRA et al., 2010) – without the semantics of relationship-aware entities – to the unified model of Social Machine we describes in Section 4.5, which is capable of representing a Social Machine as a whole as well as the composition of its related *socially connected units*. The following are some improvements we obtained in this transition, marked by the implementation of *Futweet*:

- The initial model was simplified with the creation of a general abstraction that incorporates the well-known elements of computing systems (e.g., *input, output, states, processing unit*), into a single computational unit (i.e., IPS);
- The semantics of the *relationship* was made explicit and, besides representing static connections and dependencies, established constraints that are influent in the way Social Machines dynamically interact with each other;
- The sets of required and provided services characterized the classical composition mechanisms from the notion of software component (LAU; WANG, 2007) and facilitated the implementation of the concept of composability as a design principle of software architecture. In more than one sense, after the preliminary Futweet experience, the Social Machine's building blocks indeed blended the principles from other abstraction models, such as component and

service, but adding *relationships* and *interactions* as key social elements;

- *Requests* and *responses:* instead of considering requests and responses as fixed element of the model as initially presented in (MEIRA et al., 2011)), they started to be considered as a kind of *interaction mechanism* used by the SM's services. This fact makes more sense because the sets of [required/provided] *services* are elements of the SM building block, and, in this case, request-response represents a kind of message exchange pattern already brought from the concept of services.

Besides *Futweet*, other projects and case studies have been implemented, such as the *WhereHere* Social Machine (BRITO et al., 2012). However, in all cases, *relationships* had not yet the exact semantics of "social connections", as explained in Chapter 4; they basically were seen just as dependencies between the involved parts. In addition, such different parts - used to compose the whole system - were not individually encapsulated as real Social Machines. As a result of these observations, some improvements were made in the Social Machine abstraction model (BURÉGIO et al., 2013b) and other implementations were developed to better explore the concept of *relationship* as the centerpiece of the proposed paradigm.

6.3. Applying SM to Different Contexts

In order to evaluate how the Social Machines paradigm can be broadly applied, we implemented case studies in different contexts. These case studies use the concepts we have discussed so far with the aim of deriving Social Machines that have, or may have in near future, a significant impact on the lives of *individuals*, *business, governments,* and possibly the society as a whole. Given that, we conducted this practical experiences as follows:

- **Case 1:** People as Social Machines (for individuals)
- **Case 2:** Social Enterprise (for businesses)
- **Case 3:** Government as a Social Machine (for governments)

Next Sections look deeply at this practical experiences, structuring each case study in terms of *i) motivation, ii) scenario, iii) proposed Social Machine, iv) realization of the proposed SM* and *v) discussion.*

6.4. People as Social Machines

Based on the concept of *Personal APIs* (BURÉGIO et al., 2014a) (see Section 3.5.4), this initiative aims to develop a Social Machine that "encapsulates" people in order to provide a set of specialized open APIs on the Web. These APIs should allow third-parties to programmatically access information about a person (e.g., health-related statistics, busy data) and/or trigger his/her human capabilities (i.e., order to perform a certain task) in a standardized way. The initial result of this initiative is the Social Machine called *[YOU]*, which was implemented as a *personal information retrieval platform* in which “you” (the information related to you) is wrapped as a composite Social Machine.

6.4.1. Motivation

[YOU] is a Social Machine that wraps “you” (i.e., your information). One of the main goals of implementing [YOU] was to apply the discussed design guidelines to create an example which embraced the notion of *relationship* and sociability, discussed so far. The central motivation of *[YOU]* is based on the fact that nowadays we have to deal with a large number of information about (and related to) us. In general, this information spreads across multiple sources and there is a huge effort to connect related things that matter to us. Figure 6.3 illustrates this *information deluge*, with each little square representing a different piece of information, and each color meaning a different type of them. Indeed, often there are few things that really matter to us, but sometimes, it becomes hard to connect them (Figure 6.3b).

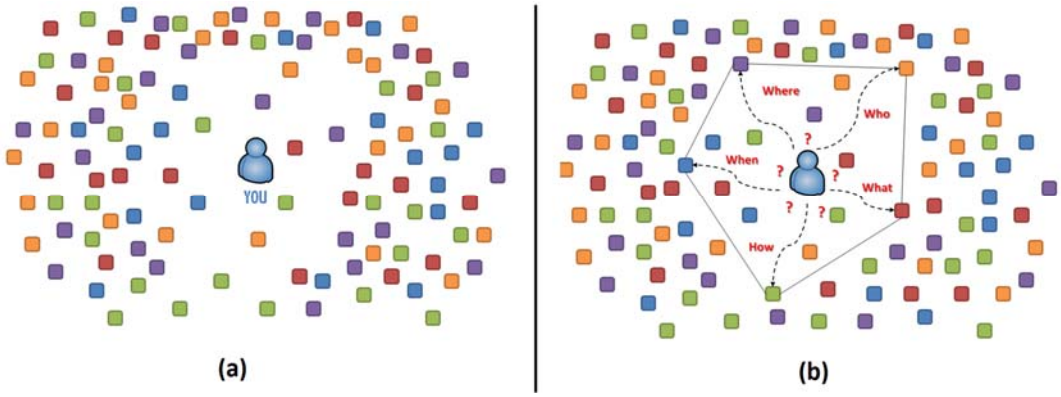

Figure 6.3 – Information deluge: (a) large number and types of information about you; (b) huge effort to connect related things.

6.4.2. Scenario

The following scenario illustrates the aforementioned information deluge issue, by presenting a example that involves different pieces of related information spread out through different systems in the Web, such as *Gmail*, *GCalendar*, *Dropbox* and *Facebook:*

Suppose you are a PhD student and your advisor used his Gmail account to send an email to you and other students with some guidelines about what should be considered during the next meeting with him. In the email, he informs that there will be a discussion about an article he's shared with all of you, through his Dropbox account. He also says that an appointment was booked in the GCalendar, with information about local and time. Furthemore, he's created an event in Facebook to invite other people to attend the discussion.

In this scenario, there are at least four pieces of information related to the same event: *(1)* the ***email*** with guidelines; *(2)* the shared ***article*** to be discussed; *(3)* the ***appointment*** in the *GCalendar* with local and time; and *(4)* the ***event*** in *Facebook* with a list of additional participants. Thus, if we want to have a complete overview of the set of information related to the meeting, we are supposed to manually gather these pieces of information, by using the different sources in which they are stored in (i.e., *Gmail*, *Dropbox*, *GCalendar* and *Facebook*).

6.4.3. [YOU]: The Social Machine that wraps "you"

Inspired by this illustrative scenario, we defined the Social Machine called *[YOU]*. One of the goals behind the *[YOU]* Social Machine is to provide a single access point to your information with the possibility of connecting and share them in a useful way, as illustrated in Figure 6.4.

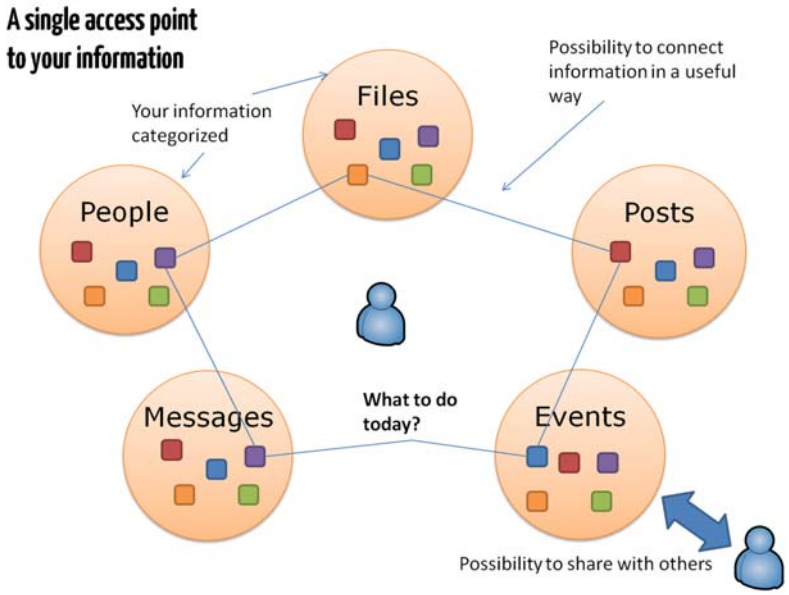

Figure 6.4 - [YOU]: A Single Access point to your information

This idea sets up the context to think about possible scenarios involving the use of our Social Machine model (Section 4.5) in conjunction with *Personal APIs*. For 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 6.5 illustrates this conceptual scenario by using our SM model to represent people as interacting Social Machines. As can be seen in Figure 6.5, the SM's *Provided Services* (PS) are exposed as *Personal APIs*. Such APIs allow both access to personal data (illustrated in the figure as little multicolor squares) and the execution of human-based activities.

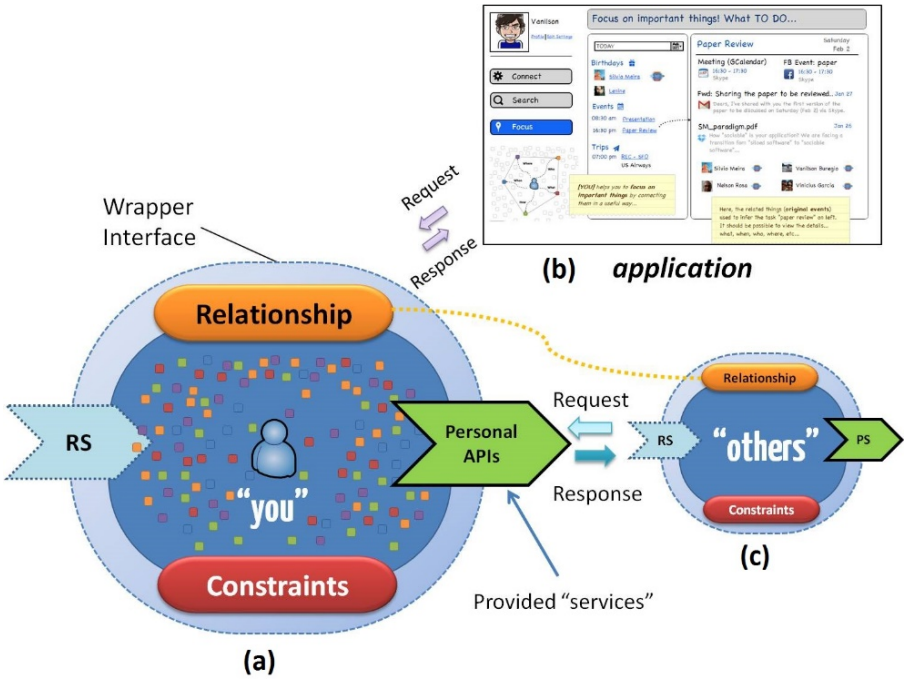

Figure 6.5 - Conceptual view of People as "relationship-aware" Social Machines: (a) [YOU] Social Machine; (b) application built on top of [YOU]; (c) other [YOU]-like Social Machine

The *[YOU]* Social Machine (Figure 6.5a) was designed as a *personal information retrieval platform* in which "you" (the information related to you) is wrapped as a Social Machine. On top of the *[YOU]* Social Machine we built the *[YOU] application* (Figure 6.5b) - a Web-based interface that uses the *[YOU]-SM's APIs*. In practice, it is important to note that *[YOU]*-SM should represent "you" as a Social Machine and, like you, it also provides ways to establish different *relationships* and *interaction views* with "others", i.e., other people wrapped as *[YOU]*-like Social Machines (Figure 6.5c). In this case, 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.

Such SMs are autonomous and can be independently deployed. So, it is possible to consume the services of the *[YOU]* Social Machine with third-party apps and in this way to enable the creation of an ecosystem of applications built on top of its services.

As aforementioned, the *[YOU] application* was built on top of the *[YOU]*'s *Personal APIs* to provide three core functionalities, namely: *connect*, *search* and *focus*. Table 6.2 presents a brief description of each of them. For

more details, the screenshots of the *[YOU] application* is presented in Appendix B.

Table 6.2 - Core functionalities of the [YOU] Application

Basic Functionalities	Description
Connect	It provides a way to connect the [YOU] application to the main user's sources of information. Initially the sources mentioned in the motivational scenario were considered: *Dropbox*, *Facebook*, *GCalendar* and *Gmail*.
Search	It represents a single access point to search and combine related things from the set of connected sources of information.
Focus	It helps the users to focus on important things by connecting them in a useful way. It answers the question: "*What to do today?*", by grouping related information (events, people, files, etc...) from different sources to show to the users what he/she has to do in a specific date.

6.4.4. Realizing the [YOU]-SM

After setting up the general context of [YOU], one question is *how* to derive *[YOU]* as a Social Machine. As a software development guideline, we consider the steps we described in Chapter 5, namely: i) *Define building blocks, ii) Specify services, iii) Design data integration* and *iii) Design interaction models.* This section provides a general overview of how we implemented the *[YOU]-SM* by walking through these steps.

Define [YOU]'s building blocks

In the case of *[YOU],* we designed the whole system as a *composite Social Machine* internally formed by the combination of other SMs that wrap the user's different sources of information. Figure 6.6 shows a logical view of this composite SM.

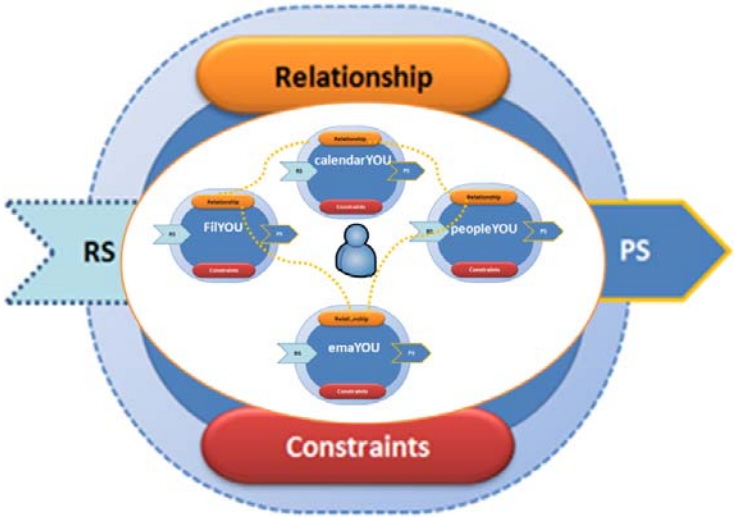

Figure 6.6 – Logical view of *[YOU]* as a composite Social Machine

As can be seen in this figure, the sources of information were wrapped as independent and autonomous Social Machines. Each of them with its own URL, used to access the provided services. In this way, it was possible to independently deploy each SM on a different provider. Table 6.3 shows each Social Machine considered by the *[YOU]*, their identifiers (URL), the wrapped sources of information and abstract data types. In the base URL, *{host}* represents the service provider in which the SM is deployed and *{user}* is the identifier of the user, whose information is wrapped by the whole system.

Table 6.3 - List of Social Machines that compose the [YOU]-SM

Social Machine	Base URL	Wrapped Source	Abstract Data Type
CalendarYou	*https://{host}/{user}/calendaryou*	*GCalendar*	*Event*
PeopleYou	*https://{host}/{user}/peopleyou*	*Facebook*	*People*
EmaYou	*https://{host}/{user}/emayou*	*Gmail*	*Message*
FileYou	*https://{host}/{user}/fileyou*	*Dropbox*	*File*

Define [YOU]'s Services

The services provided by *[YOU]* and its internal SMs were designed as endpoints of a REST API. A set of common services was defined for each Social Machine, including *search*, *list*, *detail*, *link* and *unlink*. For example, Table 6.4 shows how the set of services of the Social Machine *calendarYOU* was designed.

Table 6.4 - CalendarYOU's provided services

CalendarYOU		
Service	**Http request**	**Description**
URIs relative to https://{host}/{USER}		
List	GET /calendaryou/	Returns the lists of agendas that belongs to the user
Search	GET /calendaryou/search?q=*query*	Returns the events related to the search query
List by date	GET /calendaryou/date=*yyyyMMdd*	Returns the events in a specified date
Get details	GET /calendaryou/{id}	Returns the details of a specific agenda identified by {id}
Link	GET /calendaryou/link	Link to the user's google calendar account
Unlik	GET /calendaryou/unlink	Unlink to the user's google calendar account

These services are invoked by the [YOU] composite SM as a way to minimize the complexity to interact with each specific source of data. The *link* service, for example, abstracts out the whole authorization process necessary to access the user's data stored in an individual source, e.g., *GCalendar*.

Design [YOU]'s integrations

As [YOU]-SM deals with data from multiple sources, there is an evident need to provide ways of integrating them. In the [YOU] context, we categorized the user's information into four abstract data types, namely *People*, *File*, *Message* and *Event*. Their internal structures are shown in Figure 6.7.

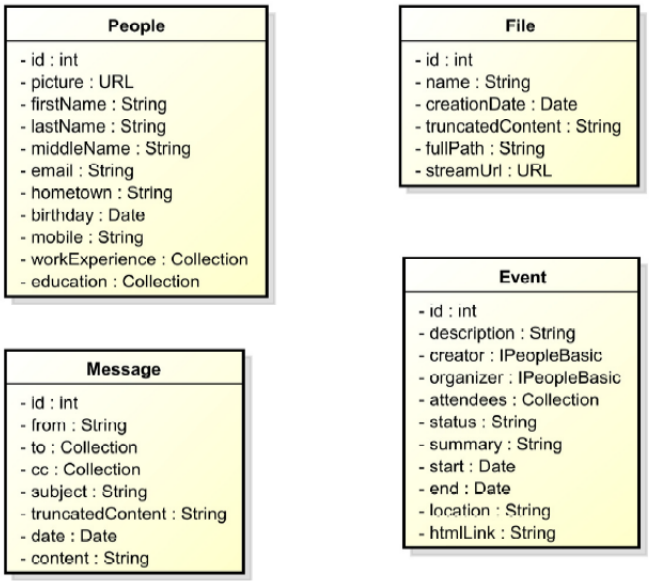

Figure 6.7- [YOU]: main abstract data types

Each Social Machine manipulates one of the abstract data types defined in Figure 6.7. As each source deals with its own specific formats and types, the element *Wrapper Interface* (*WI*) is fundamental for collecting and converting data from each specific wrapped *Information Processing System* (i.e., *Facebook, Gmail, Dropbox* and *GCalendar).* Following our *reference*

architecture (Section 5.5), WI indeed uses the *pipes and filters pattern* as illustrated in Figure 6.8. In the case of *[YOU]'s* Social Machines, the main format conversions take place in mapping from JSON (JavaScript Object Notation) into entity objects (i.e., the defined abstract data types) and vice-versa. The *format converter* is the component in charge of converting these data to the format required by the *Persistence Manager* (PM) which encapsulates the persistence logic provided by each internal Social Machine.

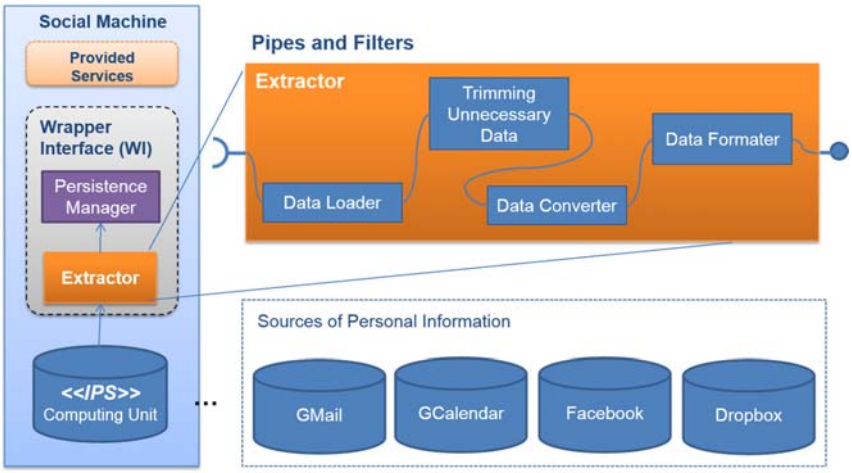

Figure 6.8 – The Wrapper Interface as a set of pipes and filters

Model-View-Control (MVC)

The *[YOU]-SM* follows exactly the *reference architecture* we defined for Social Machine (Section 5.5), having only as a *variation point* the set of wrapped IPSs (*Information Processing Systems*) which, in this case, is formed by the different sources of personal data available on the Web. The adopted *Model-View-Control* (MVC) pattern is very useful for the [YOU] application, once it manipulates data to present views according to user inputs. As can be seen in Figure 6.9, the *model* layer of the composite [YOU]-SM has access to data from the set of individual SMs, and groups them in a structure to be used by the [YOU]-SM's personal APIs.

Data Federation

The *Data Federation* pattern is also used to aggregate necessary data from the set of parallels *pipes and filters* defined by each individual Social Machine. Each source of data remains under control of an individual SM which asynchronously pulls data on demand for federated access. In this case, the *Information Grouping* component acts as an asynchronous data handler,

enabling the [YOU]-SM to start an "external process" while the handler continues processing. Then, the handler continues without waiting for the external process to finish. This design decision was very important to solve some performance issues faced during the process of data federation. The use of asynchronous data handler was very useful to compose, for example, the result of a search on multiple sources, which is one of the services provided by the [YOU]-SM.

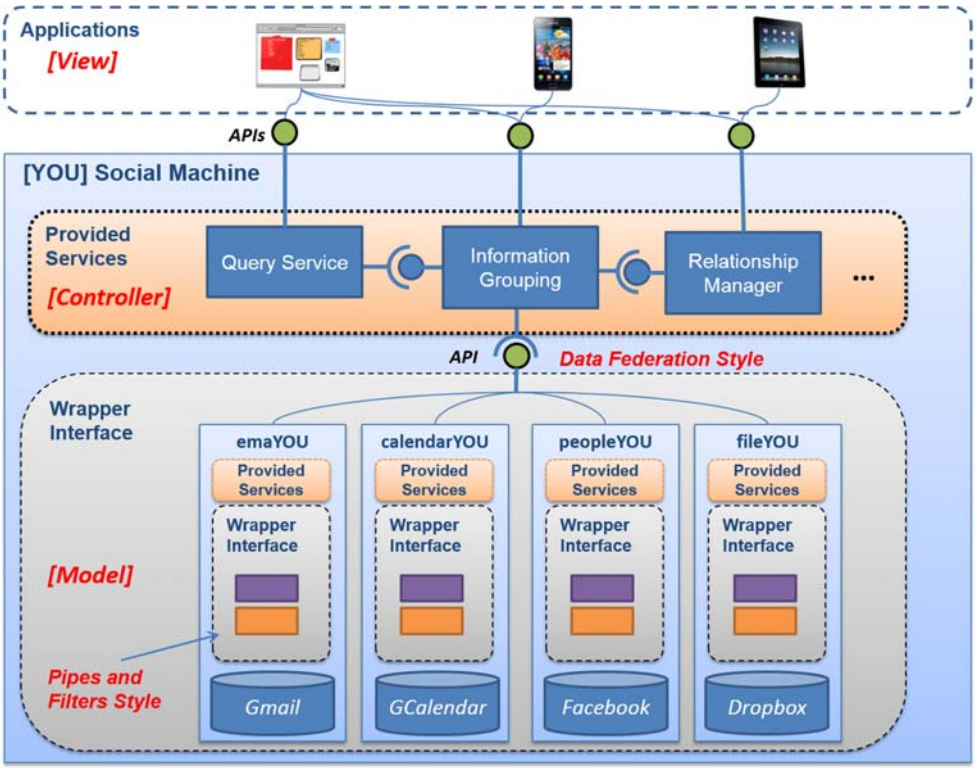

Figure 6.9 – Overview of the [YOU]-SM's architecture

Design [YOU]'s interaction models

In the [YOU]-SM, there are two interaction models that worth highlighting: *i) Interactions with the sources of data;* and *ii) Interactions between [YOU] and third-party apps.*

i) Interactions with the sources of data

These interactions take place in the internal Social Machines (*CalendarYOU, emaYOU* and so on). There is an authentication process that follows the "*love triangle*" model (Chapter 5). In this case, the source of data assumes the role of the *API Provider*.

The service "*link*" is responsible for abstracting the whole authentication process required to interact with the specific sources. Figure 6.10 shows a sequence diagram containing the steps of the authentication process implemented by the "link" service of the *CalendarYOU* Social Machine, which interacts with the *Google's servers*.

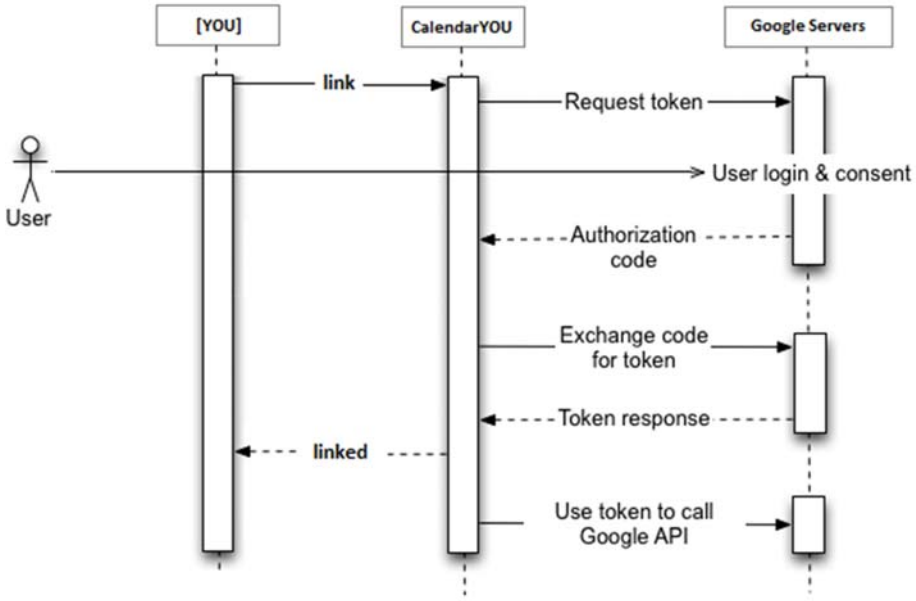

Figure 6.10 - Authentication process with a source of data

In this example, the authorization sequence begins when the internal SM (*CalendarYOU*) requests a *token* to the Google Server. Then, the *CalendarYOU* returns to the user and redirects a browser to a Google URL; the URL includes query parameters that indicate the kind of access being requested. Google Server handles the user authentication, session selection, and user consent. The result is an authorization code, which the internal Social Machine exchanges for an *access token* and a *refresh token*. The Social Machine then stores the *refresh token* for future use and gets the *access token* to access the *Google API* (GCalendar services, in this case). This process carried out during the first access. Once the *access token* expires, the Social Machine uses the *refresh token* to obtain a new one. This authentication process was implemented with OAuth 2.0 (RECORDON; HARDT; HAMMER-LAHAV, 2011) and the Social Machine can access the source API while the user is present at the *[YOU]* application or after the user has left the application.

ii) Interactions between [YOU] and third-party apps

As mentioned before, the [YOU] Social Machine allows the development of third-party applications that extend its provided services, as a way to leverage the [YOU] Social Machine's features. However, before interacting with the [YOU]'s services, applications need to establish a *registered relationship* with the *[YOU]* Social Machine. We divide these applications into two categories:

- *[YOU]-like apps* - applications that also represent or wrap a person in the real world (like done by [YOU]), and
- *General purpose apps*: applications intended to use the [YOU]'s services.

The kind of *relationships* were defined based on these categories. For general purpose apps, a relationship of type *public* was assumed. For [YOU]-like apps an *inferring process* (implemented by the *RelationshipManager* component) was used to infer the relationship to be adopted.

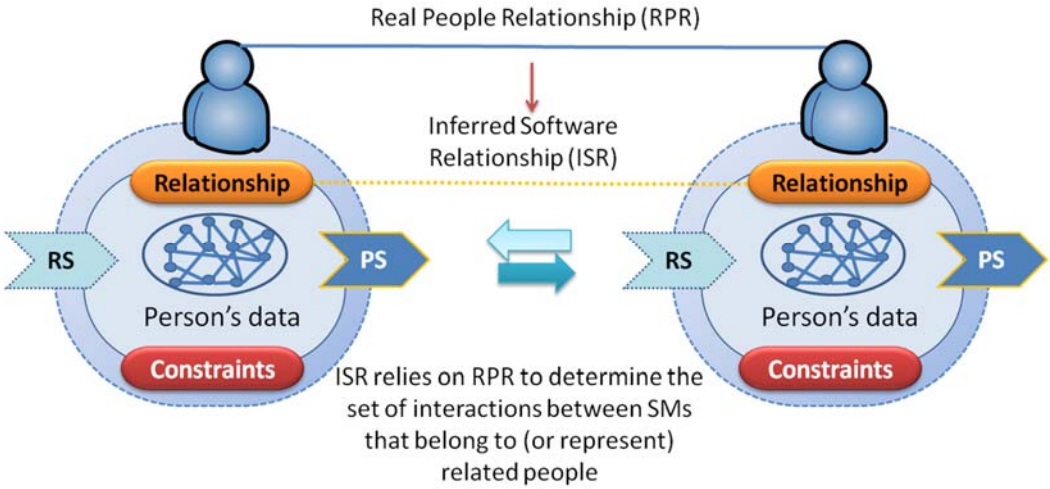

Figure 6.11 - Inferred Relationship

Such *inferring process* infers the type of relationship between the two involved people (i.e., the person wrapped by the [YOU] and the person represented by the third-party app) in order to determine which type of relationship should be considered by the involved software, as illustrated in Figure 6.11. Initially, *Facebook* was adopted as the main source to infer the relationships between people. To do that, *Facebook Query Language*[38] (FQL) was used to make queries to the Facebook's graph and the following relationship types were considered: *friend*, *family*, *work* and *education*. These types of

[38] https://developers.facebook.com/docs/reference/fql/

relationships are used to determine different interaction views between two related Social Machines. Table 6.5 shows how they establish what services of the *CalendarYOU* should or not be accessed. In this table, "*X*" means allowed by default and "*Under Approval*" means that the user has to approve the access in advance, which is similar to the authentication process used to interact with the sources of data.

Table 6.5 - Example of permissions based on relationships

Type of Application	Type of Relationship	CALENDARYOU									
		Search Events					Add Event				
		Public	Specific Calendar			All	Public	Specific Calendar			All
			Research	Personal	Work			Research	Personal	Work	
General-purpose	*public*	under approval					under approval				
[YOU]-like app	*Friend*	X					under approval				
	Family	X		under approval		under approval					
	Work	X	under approval		X					X	
	Education	X	X		under approval			under approval			

6.4.5. Discussion

[YOU]–SM is an example of Social Machine that really embraces the idea of *relationship*. With the implementation of [YOU] we learned other aspects that should be taken into account when designing a Social Machine as a platform of services:

- When implementing Social Machines that deal with different sources of data, it is very important to decide in advance how relevant data or function will be aggregated, correlated and corrected. This involves not only the definition of the main abstract data types but also the design patterns to be used to collect and format such data;
- Combining design patterns is in fact a way to minimize integration issues. In this case, we followed the reference architecture and combined *MVC*, *data federation* and *pipes&filters*, and confirmed that the data federation pattern can be accomplished by several parallel pipes and filters.
- The responsibilities of the wrapper interface became clearer with the introduction of converting and formatting operations; and

- By abstracting the basic flow of authorization and authentication between the Social Machines and different parts into a single service (i.e., link) greatly minimized the complexity of implementation of such process.

It is also important to highlight that by using *Personal Social Machines*, we are defining a software-to-software interface, not necessarily a user interface. In fact, the main goal of building autonomous and independently deployed personal SMs in the Web is to allow the creation of an ecosystem of applications built on top of the services provided by such SMs. These applications allow 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 **EVERY**one (in this case) and possibly any**THING**, creating the basis for a world in which **EVERYTHING** is going to be socially connected. In more than one sense, we can say that 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.

6.5. The Social Enterprise

Following our goal of deriving Social Machines that have a significant impact on individuals, enterprises and government, this second case study is focused on applying Social Machines into the enterprise context.

It was developed as a proof-of-concept of a joint-initiative that we refer to "*The Social Enterprise*" (MAAMAR; BUREGIO; MEIRA, 2014)(BURÉGIO; MAAMAR; MEIRA, 2015). The Social Enterprise is a kind of "*Enterprise 2.0*"[39] with a business model composed by two distinct worlds, known as the *business world* - associated with business process management platforms - and the *social world* - associated with Web 2.0 platforms. As both worlds need to be connected, the purpose of this case study is to bridge the gap between them through the use of Social Machines in a meet-in-the-middle environment.

[39] "Enterprise 2.0" is a term originally proposed by McAfee to designate "the use of emergent social software platforms within companies, or between companies and their partners or customers" (MCAFEE, 2006)

6.5.1. Motivation

Figure 6.12 illustrates our proposed business model for the social enterprise. As can be seen, on the one hand the *business world* hosts the enterprise's *business processes* (BPs), which consist of a set of *tasks* capable of manipulating *business artifacts* (BAs). On the other hand, the *social world* hosts *social processes* that take place by the execution of different *social actions* on the *social artifacts* (SAs) deployed on top of Web 2.0 platforms.

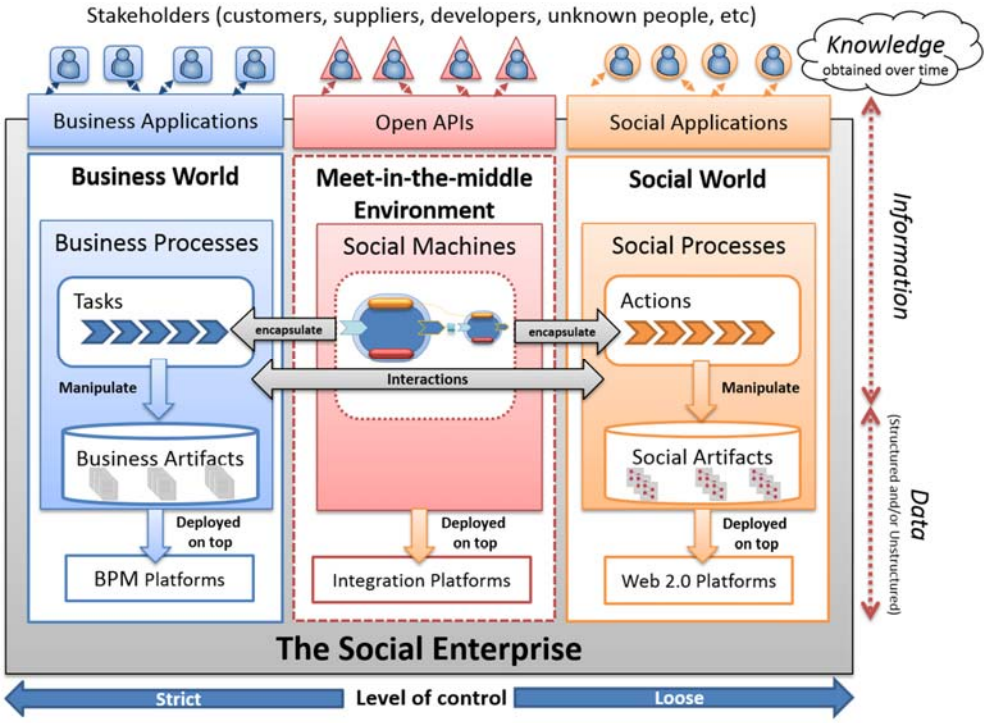

Figure 6.12 - Proposed business model for the social enterprise

In the business side, *Business Artifacts* (*BAs*) (NIGAM; CASWELL, 2003) represent identifiable pieces of information and/or collections of related data that are used by a person to actually "run a business". Examples of *BAs* include data abstractions such as an *order*, *customer*, *product*, and so on. On the other side, *Social Artifacts* (*SAs*) (MAAMAR; BUREGIO; MEIRA, 2014) abstract objects/events associated with Web 2.0 applications. Examples of *SAs* include a *post*, *invite*, *tag* and so on.

The level of control in this model indicates how much control an enterprise has over the operations it initiates. Thus, it ranges from *loose* in the social side to *strict* in the business side. In fact, dedicated business process

management platforms allow process engineers to design, develop, deploy, and track processes in a more controlled way. However, in the social world, processes are often performed in an unstructured and uncontrolled way, in response to online *social actions*[40] that Web 2.0 applications allow users to execute, e.g., to hare a file, post comments, launch a social event, co-author a text and invite friends.

This model for the social enterprise indeed requires an online presence tightly-coupled with a set of Web 2.0 applications which should be used to support the enterprise in the process of reaching out to its stakeholders, such as customers, suppliers, competitors and, more recently, third-party developers. It is worth noting that unknown people can also be treated as stakeholders and hence, can interact with the enterprise.

6.5.2. Scenario

This case study refers to Jones-Onslow Electric Membership Corporation[41], which is an electric distribution cooperative in the US providing utility service to more than 54,000 homes and businesses. To illustrate our work we assume that Jones-Onslow is about to launch an awareness campaign about renewable energy using social media like Facebook and Twitter.

Considering our two-side model for the social enterprise, we should mediate the interactions between the two worlds in a way that the business world can have an impact on the social side (e.g., marketing business process that launches a new campaign on Facebook) and vice-versa (e.g., online comments on the campaign are used to adjust the marketing business process).

6.5.3. The Meet-in-the-middle Social Machines

Figure 6.13 represents our architecture for the two-side enterprise supported by a meet-in-the-middle platform that acts as an integration tier connecting both sides through Social Machines.

[40] *Social actions* in Web 2.0 applications are counterpart of *tasks* in the business world
[41] http://www.perceptivesoftware.com/casestudies/jones-onslow-electric-membership-corporation.

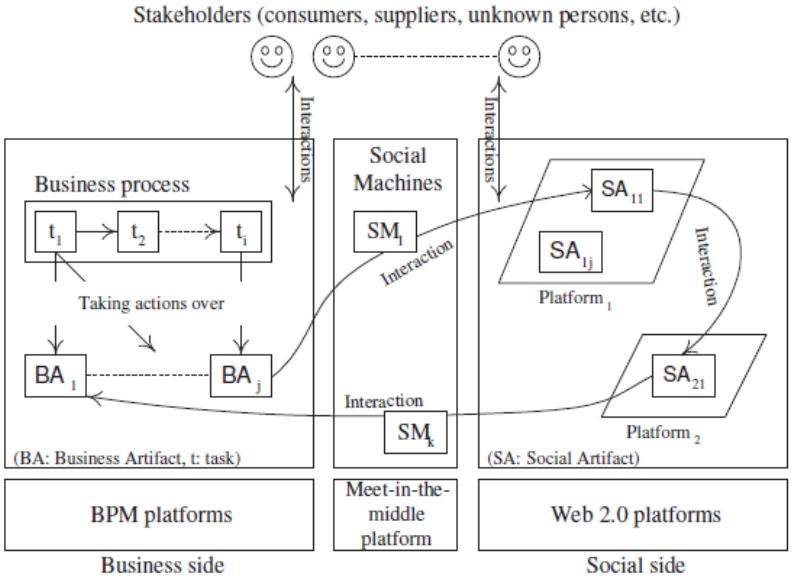

Figure 6.13 - Architecture of the two-side enterprise

The meet-in-the-middle platform comprises a set of SMs that support interactions between *BAs* and *SAs*. These SMs define a unified model to wrap Web 2.0 platforms like *Twitter*, *Facebook*, *Google*, and *Dropbox*, and provide specialized APIs that make easier the manipulation of SAs hosted on several platforms.

Indeed, the diversity of Web 2.0 platforms (in terms of APIs, protocols and data format) makes it difficult to consume their provided functionalities in a unified way. SMs hide this complexity of interacting with multiple platforms and dealing with the existing variety of data formats and types. To this end a set of common functionalities are defined per SM. These functionalities abstract out the messages that implement the interactions between the business and social sides.

Table 6.6 includes some common functionalities that the meet-in-the-middle SMs provide. The majority of these messages originate from the business side since it hosts the processes that drive the enterprise operation and hence, trigger social activities that should be performed. Additional social activities can be driven by the outcome of other activities in the Web 2.0 platform.

Table 6.6 - SM's common functionalities to abstract messages between Business (*B*) and Social (*S*) sides

Name	From:To	Description
link	$\mathcal{B}$:$\mathcal{S}$	connects the business side to a platform hosting $\mathcal{SA}$s in the social side.
unlink	$\mathcal{B}$:$\mathcal{S}$	coupled with *link*; disconnects the business side to a platform in the social side.
search	$\mathcal{B}$:$\mathcal{S}$	looks for $\mathcal{SA}$s according to a provided search query.
list	$\mathcal{B}$:$\mathcal{S}$	identifies $\mathcal{SA}$s according to specific filters (e.g., all users who declined an invitation to a social initiative).
create	$\mathcal{B}$:$\mathcal{S}$	requests the establishment of a new $\mathcal{SA}$.
delete	$\mathcal{B}$:$\mathcal{S}$	coupled with *create*; requests the removal of a $\mathcal{SA}$.
update	$\mathcal{B}$:$\mathcal{S}$	requests changes in the data/state of a $\mathcal{SA}$.
ping	$\mathcal{B}$:$\mathcal{S}$	requests details (e.g., current data values and state) on a $\mathcal{SA}$.
ack	$\mathcal{S}$:$\mathcal{B}$	coupled with *ping*; returns the current data values and state of a $\mathcal{SA}$.
subscribe	$\mathcal{B}$:$\mathcal{S}$	allows the business side to subscribe to an event of interest (e.g, changing in a $\mathcal{SA}$ status) and be notified later.
unsubscribe	$\mathcal{B}$:$\mathcal{S}$	coupled with *subscribe*; allows the business side to unsubscribe from an event of interest.
notify	$\mathcal{S}$:$\mathcal{B}$	coupled with *subscribe*; notifies the business side when an event of interest occurs.

Abstracting a set of messages into a single SM's functionality

Let us use Jones-Onslow to illustrate how the SMs' functionalities, e.g., *link*, abstract messages between the business and social sides. First of all, we assume that Jones-Onslow has accounts (*usernames/passwords*) registered in different Web 2.0 platforms. These accounts enable the creation of specific *SAs* hosted by this platforms. For example, the members of the marketing department signs in Facebook to create a marketing campaign post on the Jones-Onslow's Facebook page. This is an example of a "manual" creation of posts.

However, if we want to allow a business process to create posts on Jones-Onslow's Facebook page, we need to understand how Facebook allows third party applications to manipulate information on behalf of the Jones-Onslow's account. In practice, this involves an authentication process of acquiring access tokens, requesting approvals, exchanging authorization codes, and so on. In order to facilitate this process, SMs provide a single functionality named *link* (Table 6.6), which establishes a "*pre-authorized*" communication channel between the business and social sides. The functionality *link* is implemented by every SM and- is responsible for abstracting the whole authentication process required to interact with specific platforms. Figure 6.14 shows a UML sequence diagram of the authentication process implemented by *link* of a generic meet-in-the-middle SM.

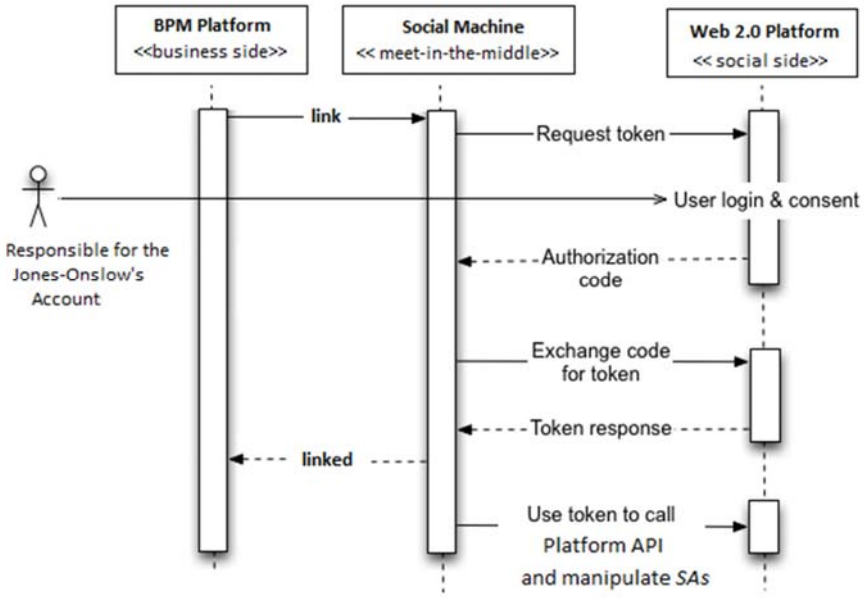

Figure 6.14 - Simplified authentication process using a single link functionality

In Figure 6.14 we have a sequence diagram similar to the one presented in Section 6.4, which reinforce that this is indeed a common functionality to be considered by Social Machines in different contexts.

6.5.4. Realizing the Meet-in-the-middle SMs

A proof-of-concept of how to bridge the gap between the business and social worlds has been developed using JavaTM JDK 1.7 platform and the Web application framework Play Framework 2.2.15[42]. This framework is based on "convention over configuration" concept that facilitates the implementation of the meet-in-the-middle Social Machines through a development model for building easy-to-maintain scalable services. The proof-of-concept provides a friendly Web interface to access functionalities that simulate messages between elements from the business and social sides (as per Table 1). Initial implementation of Jones-Onslow considers *Campaign-BA* as the main *BA* in the business side. In the social side, we adopt *Facebook* as the Web 2.0 platform to host some *SAs* for instance, *Invite-SA*. Invite-SA abstracts a *social event* (e.g., a social event on the user's Facebook Calendar) that aims at making some people sign up in an ongoing social initiative. The *SA's* properties and lifecycle are shown in Figure 6.15.

[42] http://www.playframework.com.

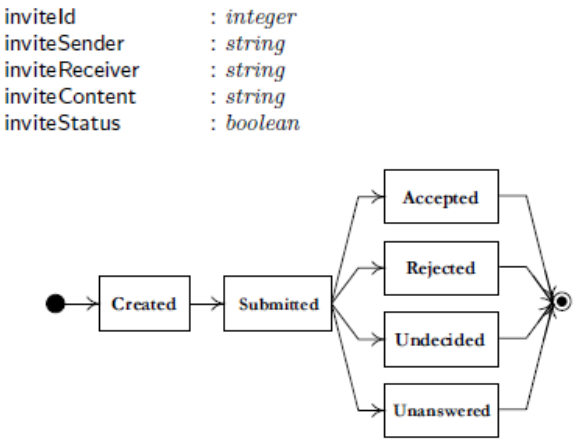

Figure 6.15- Invite-SA's lifecycle

A meet-in-the-middle SM, namely *Invite-SM*, wraps Facebook and provides specialized APIs for the business side. These APIs simplify the manipulation of *Invite-SA* by abstracting its properties, lifecycle, and activities executed on it through Facebook's operations.

REST principles are used to guide the development of the different APIs (FIELDING, 2000). Furthermore, the widely known technologies such as JavaScript Object Notation (JSON)[43] (data-interchange format) and Open Authorization Protocol (OAuth)[44] (authentication) are also used to implement message and data exchanges between the business and social sides.

Realizing the Common Functionalities

Table 6.7 groups *Invite-SM*'s specialized APIs into the common functionalities presented in Table 6.6 and summarizes how they are designed by showing which HTTP method is used to support each functionality. As per Table 6.7, *Invite-SM* connects the business side (hosting *Campaign-BA*) to the social side (hosting *Invite-SA*) through APIs. It considers the *properties* (e.g., *campaign-id*, *inviteSender*, and *inviteReceiver*) and *states* (e.g., *submitted*, *accepted*, and *unanswered*) of the involved artifacts in order to implement the specified functionalities. For example, based on *Invite-SA's* properties and lifecycle presented in Figure 6.15, the implementation of the API that lists users attending a specific campaign (i.e., GET https://*{host}*/invite-sm/*{campaign –id}*/invited)

[43] http://json.org.
[44] http://oauth.net.

considers all *inviteReceivers* from Invite-SAs in the *Accepted* state and whose *inviteSender* is campaign-id.

Table 6.7 - *Invite-SM*'s specialized APIs grouped into common functionalities

Specialized APIs		
Name	**HTTP request**	**Description**
URIs relative to https://{*host*}/invite-sm		
link	POST /link	Connects to the Jones-Onslow's Facebook account.
unlink	POST /unlink	Disconnects to the Jones-Onslow's Facebook account.
search	GET /search?*q=query*	Returns Invite-$\mathcal{SA}$s related to the search query.
list	GET /{*campaign-id*}	Gets Invite-$\mathcal{SA}$s that belong to the Campaign-$\mathcal{BA}$ identified by {*campaign-id*}.
	GET /{*campaign-id*}/invited	Lists users invited to the campaign identified by {*campaign-id*}.
	GET /{*campaign-id*}/attending	Lists users attending the campaign identified by {*campaign-id*}.
	GET /{*campaign-id*}/declined	Lists users who declined the invitation to the campaign identified by {*campaign-id*}.
	GET /{*campaign-id*}/noreply	Lists users who have been not yet responded to their invitation to the campaign identified by {*campaign-id*}.
	GET /{*campaign-id*}/maybe	Lists users who have been responded "Maybe" to their invitation to the campaign identified by {*campaign-id*}.
create	POST /{*campaign-id*}	Requests the creation of a new Invite-$\mathcal{SA}$ to the campaign identified by {*campaign-id*}.
delete	DELETE /{*campaign-id*}/{*invite-id*}	Requests the removal of the Invite-$\mathcal{SA}$ identified by {*invite-id*}.
	DELETE /{*campaign-id*}	Requests the removal of all Invite-$\mathcal{SA}$s associated to the campaign identified by {*campaign-id*}.
update	PUT /{*campaign-id*}/{*invite-id*}	Requests changes in the data and/or state of the Invite-$\mathcal{SA}$ identified by {*invite-id*}.
ping	GET /{*campaign-id*}/{*invite-id*}	Gets details (e.g., current data values and state) of the Invite-$\mathcal{SA}$ identified by {*invite-id*}.
ack	HTTP RESPONSE CODE:200 {*data*}	Returns the current data values and state of a pinged Invite-$\mathcal{SA}$.
subscribe	POST /subscribe/{*topic*}	Subscribes to the topic (i.e., {*topic*}) of interest.
unsubscribe	POST /unsubscribe/{*topic*}	Unsubscribes to the topic (i.e., {*topic*}) of interest..
notify	POST {*callback URL*}	Notifies the business side, via an previously informed callback URL, when an event of interest occurs.

Asynchronous communication

Another key aspect in bridging the gap between the business and social worlds is to support asynchronous communication. In the following, we present a specific scenario from the social marketing campaign of *Jones-Onslow* that

shows how this kind of communication takes place in a logical sequence of message exchange.

First of all, it is worth observing that sometimes a social marketing campaign requires a long-term system of events. In the case of Jones-Onslow's campaign, we assume that several social events occur to raise awareness of and funds about renewable energy projects. Furthermore, social media like Facebook is used in different ways to directly engage customers and other stakeholders in the campaign.

Initially, Jones-Onslow plans an opening reception event that aims at disseminating to its stakeholders the specific short- and long-term goals of its new initiatives on renewable energy. To generate interest, Jones-Onslow launches this event into the public consciousness and invites members of the Jones Onslow's Facebook account. A screenshot of the demo tool launching this social campaign event is shown in Figure 6.16.

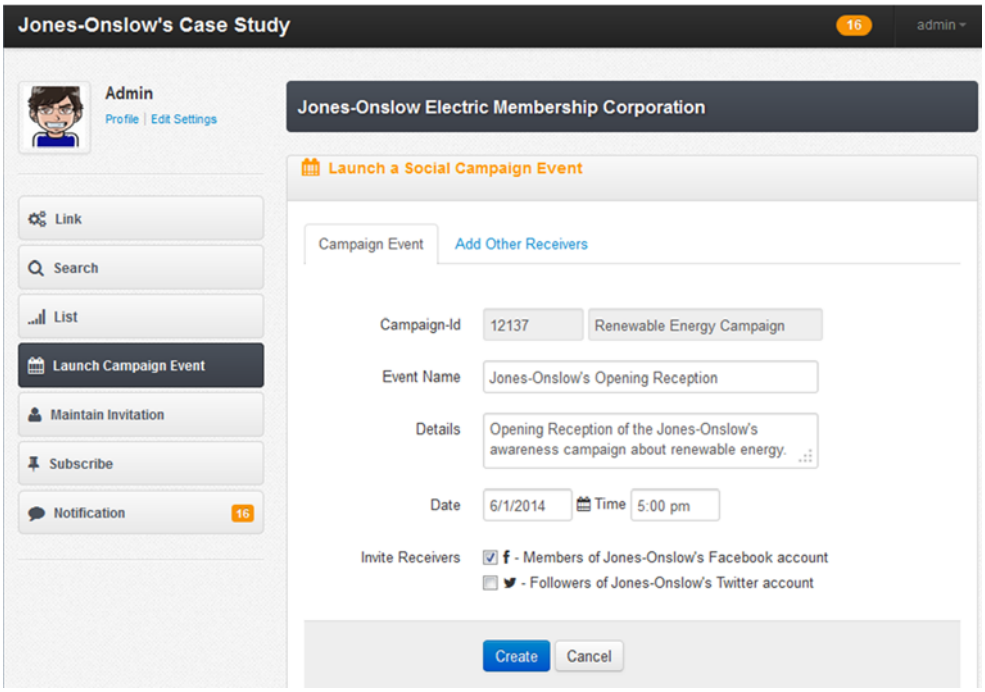

Figure 6.16 - Screenshot of the demo tool

Given the social nature of this event, the reception organizers should monitor who really accept the invitations in Facebook (following the *InviteSA*'s lifecycle presented in Figure 6.15). From a practical point of view, they need to

know how many people will attend the event to proper arrange the catering. This process involves an *asynchronous communication* between the business and social worlds. In this sense, our meet-in-the-middle Social Machine (*Invite-SM*) provides *"subscribe"* and *"notify"* functionalities that enable the implementation of these two-way communications and, in this scenario, help Jones-Onslow keeps a control over the number of invitees to expect. Figure 6.17 presents a UML sequence diagram containing some patterns of interaction among the elements that compose our approach.

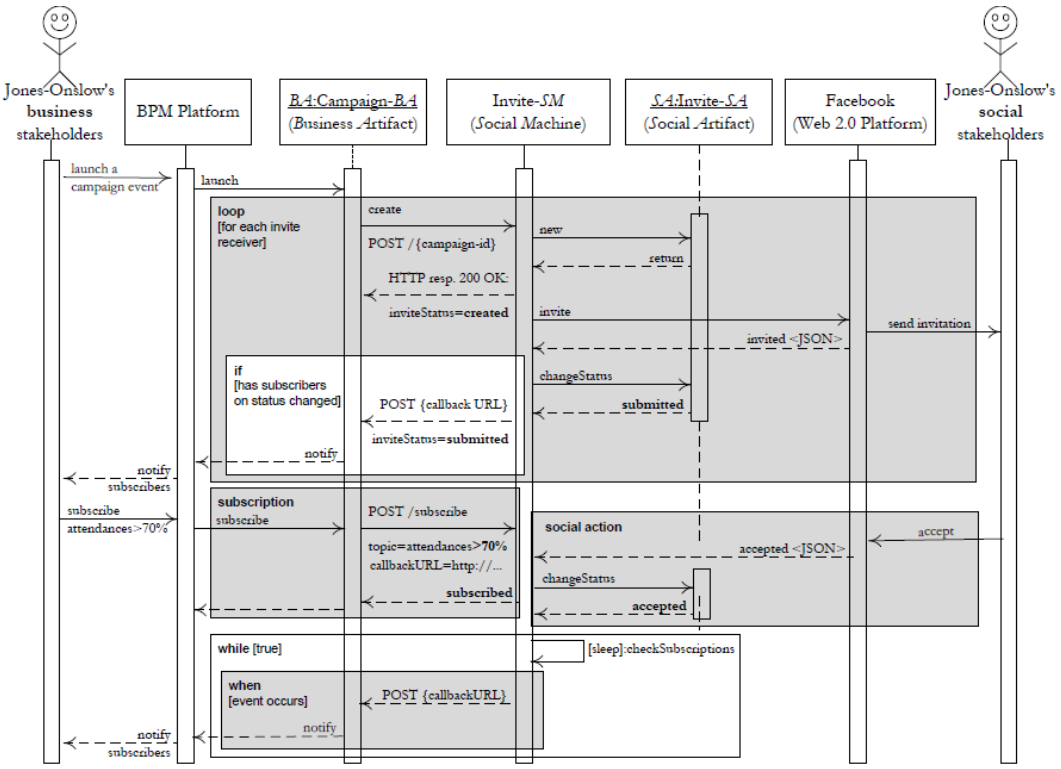

Figure 6.17 - Sequence diagram of some message exchanges

As can be seen in one part of Figure 6.17, someone is interested in being notified when the percentage of accepted invitations exceeds 70%, and provides a *callbackURL* which is called when this topic of interest occurs. This is one of various possibilities of asynchronous communication that can be enabled by this pair of *"subscribe"/"notify"* functionalities.

6.5.5. Discussion

This experience on applying Social Machines to the context of social enterprise points not only to the degree of complexity (to interact with existing Web 2.0 tool) that can be abstracted out by implementing a high-level dynamic set of APIs, but it also points to the number of new opportunities that emerge when supporting the business world to connect to Web 2.0 platforms. One of these opportunities refers to the option of dynamically adapting business processes based on *asynchronous notifications* from the social side. In the aforementioned Jones-Onslow's scenario, for example, a notification about the total number of invitees could trigger a task in the business side to change the booking of the reception's place.

One can say indeed that the meet-in-the-middle SMs bridge the gap between the business and social sides, but what about the new possibilities of *relationships* to be enabled by such SMs? Regarding this question, it worth noting that beyond mediating the two sides, the meet-in-the-middle SMs also provide ways to publish the enterprise's internal capabilities as *managed open APIs* on the Web (see Figure 6.12). It means that it should be possible to manage and consequently be aware of the different kinds of *relationships* with those (i.e., third-party apps, developers, physical objects) that use the SMs' provided services.

In fact, when a Social Machine publishes enterprise's internal capabilities to the outside world, it allows developers to add other innovative capabilities in their enhanced applications. As a consequence, this increases the chances of boosting the usage of such enhanced applications as well as creating an ecosystem of other *socially connected entities* around the enterprise. When this ecosystem expands, the whole market expands. Twitter and Facebook are concrete examples of this fact.

To sum up, the proposed meet-in-the-middle Social Machines are integral parts of a framework that helps to establish what we refer to as the *Connection*, *Open*, *Reachable*, and *Engagement* (CORE) characteristics of the social enterprise.

Connection means converting ad-hoc relations into long-lasting ones and promoting different forms and levels of interaction among the enterprise's

stakeholders and services. The social enterprise should be a truly connected business, relating their employees, customers, partners, and services with each other and with the market as well.

Open means creating new conversation channels with the business world. As aforementioned, providing SM's services as *Open online APIs* constitutes an example of things that help achieve this goal by exposing the enterprise's internal capabilities to the external world and being aware of their possibly huge number of *relationships*.

Reachable means facilitating the ubiquitous accessibility to the social enterprise. It involves, among other things, the necessity of being more responsive to the different forms of social interactions like posting notes, chatting, and updating content. As the number of stakeholders' devices (e.g., mobile, desktop, smartphones, tablets, and consoles) is increasing significantly, there is a need to think about manners to efficiently create adaptable and user-friendly online social-applications.

Last but not least, engagement means creating a culture of community that relies on collaboration, sharing and participation. Social Machines can indeed sustain this engagement by enabling the creation of online communities, crowdsourcing, and so on.

6.6. Government as a Social Machine

This case study corresponds to the third context into which we intend to apply the Social Machine paradigm. Then, it is a practical experience on designing Government as a Social Machine, as per initially planned.

Government initiatives to open data to the public are becoming increasingly popular every day. The vast amount of data made available by government organizations yields interesting opportunities and challenges - both socially and technically. In this case study we propose a social machine-oriented architecture as a way to extend the power of open data and create the basis to design government as a social machine (*Gov-SM*). The proposed *Gov-SM* provides a platform of specialized APIs to enable the creation of several other social-technical systems on top of it. Based on this experience, we can realize that deriving government as a Social Machine, in more than one sense,

collaborate to fully integrate users, developers and crowd in order to participate in and solve a multitude of governmental issues and policies. The design of this case study allows us to have clearer and more comprehensive idea about how the Social Machine paradigm can impact the lives of individuals (Section 6.4), enterprises (Section 6.5) and finally government.

6.6.1. Motivation

The notion of "open government" has been around for a long time. Since the 50s, governments have been concerned about transparency and the idea that citizens must have the "*right to know*" (PARKS, 1957)(MITCHELL, 1977) the government's workings, policies and administration (LITTLE; TOMPKINS, 1974). Since those years, governments agree that freeing government information has the potential to increase accountability, citizen participation and collaboration, while offering better public services to increase efficiency and effectiveness (WONG; WELCH, 2004)(SAYOGO; HARRISON, 2012)(HARRISON; SAYOGO, 2013).

Nowadays, the Web has played a fundamental role in the interaction between government agencies and their citizens. This is because it offers powerful means for enhancing government transparency by providing access to information and services online. In fact, the open approach of the Web has played a key role for fostering the idea that government should also be open to public, and then contribute to the widespread engagement of citizens.

As a practical result, many governments around the world have been making different efforts to benefit from Web technologies as a manner to provide *Open Data* and encourage citizens to get more directly involved in governmental issues and policy. The *Open Government Partnership* (OGP)[45] is the concrete proof of this fact. However, despite the existing efforts on open government, several technical issues continue to be a major impediment toward the widespread adoption of open data. These issues include, for example, the existence of a multitude of unstructured and outdated datasets, and the lack of standardized services to facilitate not only the consumption, but also the generation and updating of governmental datasets by citizens.

[45] Open Government Partnership, available at http://www.opengovpartnership.org

Motivated by these issues and based on some implementation experiences (BRITO et al., 2014)(BURÉGIO et al., 2014b), this case study proposes a social machine-oriented architecture as a way to extend the power of open data and create the basis to design "government as a Social Machine" (Gov-SM). The solution proposed herein use the *reference architecture* for Social Machine (Section 5.5) as a template in order to provide a platform of specialized APIs to enable the creation of several other social-technical systems (*aka* Social Machines) on top of it.

6.6.2. Scenario

Different initiatives on open government can be seen in practice as a way to take advantage of current Web technologies to launch portals of publicly available datasets. For example, we can highlight the open data portal[46] launched by the U.S. government which makes available about 85,000 datasets. Similarly, United Kingdom government also opened up its own portal[47] with more than 13,500 datasets and other additional features as, for instance, a map based search tool. This tool provides a way of searching for records of data sets and services referenced by geographical coordinates.

In comparison to these efforts, other initiatives are only at the beginning like the Brazilian government's open data, whose portal[48] contains just about 240 datasets. Regardless of some initiatives have shown that it is possible to take advantage of e-government and open data (ANDERSEN, 2009; BERTOT; JAEGER; GRIMES, 2010; KIM; KIM; LEE, 2009; PICAZO-VELA; FERNANDEZ-HADDAD; LUNA-REYES, 2013; SAYOGO; HARRISON, 2012; WONG; WELCH, 2004), existing approaches have presented problems that range from cultural to technological aspects (DADA, 2006; HUNG; CHANG; YU, 2006; JHO, 2005; PIOTROWSKI; VAN RYZIN, 2007).

Hence, based on these reports and some implementation experiences over Brazilian open data, the following issues should be considered:

1. **Overlapped and decentralized data sources:** although governments try to create central repositories, we could observe that some initiatives at

[46] U.S. Government's open data, available at http://www.data.gov
[47] U.K. Government's open data, available at http://data.gov.uk
[48] Brazilian Government's open data, available at http://dados.gov.br

local/regional level have been overlapping the ones at national level and vice-versa. New York City and Rio de Janeiro are examples of cities that conduct their own open data initiatives and portals at local level, while other disassociated efforts (dealing with similar datasets) are launched at national level. As a consequence, developers have difficulties in creating new consistent applications, because they need an extra effort to analyze, understand and deal with overlapped data extracted from a multitude of distributed sources;

2. **Lack of standards:** in addition to the overlapped and decentralized data sources, there is a lack of standards for data publishing. Each publisher chooses what and how to publish their dataset. Often, there is no common agreement between countries, states, cities or even within one city or a single government agency. As a consequence the services provided to consume open data as well as the data formats and types vary significantly; and
3. **One-way communication channel:** in general, governments tend to publish data in a one-way communication channel, i.e., from government to citizens. Due to that, the majority of existing applications are limited to help citizens only to visualize such data, not being possible to get feedback from them.

6.6.3. Realizing Government as a Social Machine

This section provides a general overview of our proposed reference architecture, by walking through the process of deriving government as a social machine (Gov-SM). This process is based on the design guidelines (Section 5.4), taking into account our proposed SM building block (Section 4.5). Next, we present more details about the specific steps we performed in order to achieve the preliminary reference architecture for the Gov-SM.

Step 1. Wrap datasets as individual Social Machines

The first step to design government as a Social Machine is to define which representative sources of data should be wrapped as individual SMs, and also how these SMs should be designed. In practice, by using our SM abstraction model (Section 4.5), any provider of open data can be considered a kind of *IPS* (i.e., *Information Processing System*) to be involved by a *Wrapper Interface* (*WI*).

Hence, in this case, we designed the whole Gov-SM as a composite social machine internally formed by the combination of multiple sources of data wrapped as independent and autonomous social machines as well. Each SM has its own identification URL, used to access its provided services. In this way, it is possible to independently deploy each SM and offer its services on different providers.

Table 6.8 shows some internal SMs considered by Gov-SM, including their base identifiers (URL), the wrapped sources of data and their provided data formats, and a brief description of what each SM actually wraps. In the base URL, *{host}* represents the service provider on which the SM is deployed.

Table 6.8 - List of some internal SMs considered to compose our governmental social machine

Social Machine	Base URL	Wraped Source	Data Format	Description
Deputy-SM	*http://{host}/deputies*	*Chamber of Deputies open data portal*[1]	*json, xml*	*Services and files with data related to federal deputies.*
Senator-SM	*http://{host}/senatores*	Federal Senate open data portal[2]	*json, csv*	*Services and files with data related to senators.*
Company-SM	*http://{host}/companies*	*Federal Revenue Service webpage*[3]	*html**	*Web page, with CAPTCHA, to verify companies data*
Health-SM	*http://{host}/health*	*National register of health facilities*[4]	*html**	*Web page to verify health units data*
Crime-SM	*http://{host}/crime*	*Brazil Open Data portal*[5] *and Municipal open data portals*	*csv, xml*	*Files with data related to crime reports*
Education-SM	*http://{host}/education*	*Brazil Open Data portal*[5] *and Municipal open data portals*	*html*, json, csv*	*Pages, services and files with data related to schools and*
Tourism-SM	*http://{host}/tourism*	*Tourism Ministry website*[6] *and Municipal open data portals*	*html*, json, csv*	*Pages, services and files with data related to touristic places*

*Not open data: Human readable web pages

[1] *http://www2.camara.leg.br/transparencia/dados-abertos*
[2] *http://dadosabertos.senado.gov.br/*
[3] *http://www.receita.fazenda.gov.br/*
[4] *http://cnes.datasus.gov.br*
[5] *http://dados.gov.br*
[6] *http://turismo.gov.br*

Step 2. Design Data Extraction Mechanisms

As the proposed *Gov-SM* deals with data from multiple sources, there is an evident need to provide ways of integrating such heterogeneous data. In the *Gov-SM* context, we categorize the wrapped datasets into different abstract data types to be handled by the designed SMs. These abstract data types include, for example, Deputy, Senator, Company, HealthUnit, School, TouristicPlace and others.

As shown in Table 6.8, each SM manipulates one or more of these abstract data types. However, the wrapped government portals and other websites do not directly provide such abstract data types. Instead of that, a variety of different data formats, e.g., csv, xml, httml, pdf, json and xls, are available. Thus, it is necessary to have mechanisms to retrieve data out of such publicly available datasets for further data processing and use.

In this case, the element *Wrapper Interface* of our SM model was used to extract and convert data from each specific *IPS* (i.e., datasets from *Brazil Open Data*, *Federal Revenue Services*, *National register of health facilities*, and other websites). Figure 6.18 shows an example of an individual SM wrapping a dataset from one of the public sources of data listed in Table 6.8.

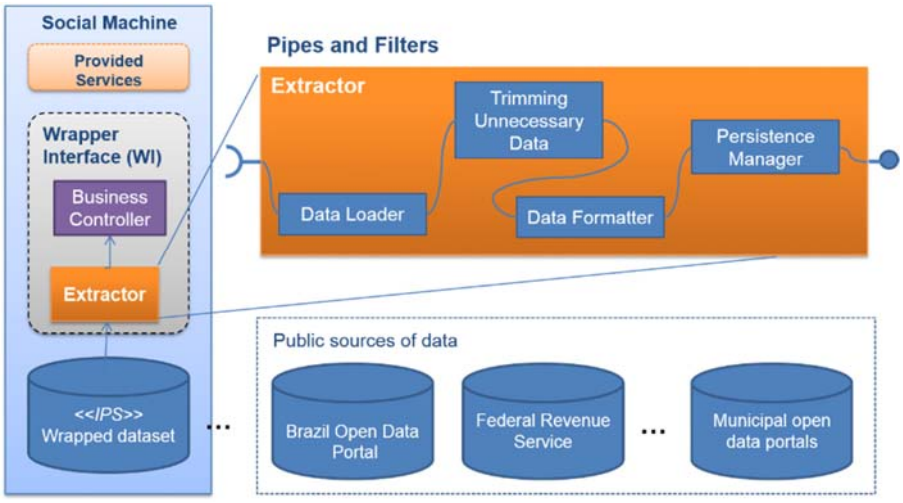

Figure 6.18 - Representation of an individual Gov-SM wrapping a data source

WI follows the pattern described in our reference architecture (see Section 5.5). It has an *extractor* component that uses *pipes and filters* as an integration pattern to create the logic for collecting and filtering the flows of data from wrapped government datasets and converting data into common and consistent abstract data types for SM manipulation.

Step 3. Specify a common set of specialized APIs

After wrapping representative datasets (Step 1) and designing the data extraction mechanisms (Step 2), this step concerns the design of the services provided by each SM. These services are often designed as endpoints of a REST API. Hence, a set of common specialized APIs is specified for each social machine, which includes services like *search*, *list*, *get details*, *report abuse* and *subscribe*. Table 6.9 shows how some services of the Deputy-SM are specified. The majority of these services are published by the composite social machine (i.e., Gov-SM) as a way to minimize the complexity of third-party applications to consume and handle the existing public datasets.

In brief, the steps presented so far help us to overcome the *lack of standards* and *one-way communication channel*. The former is reduced through the definition of abstract data types and the common set of specialized APIs as well; and the latter is minimized by some services listed in Table 6.9, like "*report abuse*".

Table 6.9 - Some Deputy-SM's specialized APIs grouped into common functionalities

Deputy-SM		
Name	**HTTP Request**	**Description**
list	GET /	Returns a list containing all deputies and their basic info
	GET /expenses	Returns a list containing current aggregated expenses of all deputies
	GET /expenses/commisions	Returns a list containing current aggregated data on commissions
	GET /expenses/laws	Returns a list containing current aggregated data about project laws
search	GET /search*?q=query*	Return deputies, expenses or laws related to the search query
get details	GET /{id}	Return basic data of deputy specified by {id}
	GET /{id}/expenses	Return expenses data of deputy specified by {id}
	GET /{id}/commisions	Return commisions data of deputy specified by {id}
	GET /{id}/laws	Return proposed laws of deputy specified by {id}
report abuse	POST /abuse/*{id}*	Report inappropriate or abusive things related to the deputy specified by *{id}*
subscribe	POST /subscribe/*{topic}*	Subscribes to a topic of interest specified by *{topic}*

Two-way communication channel

As aforementioned, governments often publish data in a static communication channel, i.e., from government to citizens. As a consequence, most of the time, it is not possible for citizens to give a feedback on something according to their concerns. In order to face this problem, some proposed SMs enable the implementation of two-way communication between government and citizens. The Deputy-SM's service called "*report abuse*" allows citizens to give a feedback on inappropriate or abusive things related to a specified deputy. Abuse complaints should be stored on the Gov-SM and possibly be redirected to social media as, for example, be posted on the Facebook deputy's message wall.

Asynchronous communication

In addition to the two-way communication, the proposed SMs services also allow the establishment of asynchronous communication. The service "*subscribe*" shows this fact. It allows requesting a subscription on a specific topic of interest, and then the SM notifies the subscriber when the event of interest occurs. Figure 6.19 shows an example of a HTTP request to the Deputy-SM's "*subscribe*" service (see Table 6.9).

This example is a request for subscribing on a specific topic of interest, i.e., a deputy's monthly expenditure on fuel. The set of parameters (Lines 6-10 of Figure 6.19) is passed via HTTP post and specifies a notification *constraint* on the Deputy-SM. Such *constraint* indicates that when the specified deputy's monthly expenditure on fuel exceeds 6,000 BRL the callback URL (Line 10) should be called by the Deputy-SM, as part of an asynchronous notification process. Other kinds of notification can also be considered such as SMS and email.

```
1  POST  /subscribe/expenditure HTTP/1.1
2  Content-type: application/json
3  Authorization: OAuth
4  ...
5
6  deputyId:71636
7  expenditureOn:FUEL
8  period:MONTHLY
9  whenExceed:6000BRL
10 callbackURL: http://<url_provided_by_the_subscriber>
```

Figure 6.19 - Example of a HTTP request for subscribing on a specific topic of interest

Compose the "relationship-aware" Gov-SM

By adopting the proposed reference architecture (Section 5.5), we also designed the Gov-SM as a combination of different architectural styles (i.e., *pipes and filters*, *data federation* and *MVC*) to aggregate and relate data and services from various publicly available sources. The overall obtained architecture is depicted in Figure 6.20.

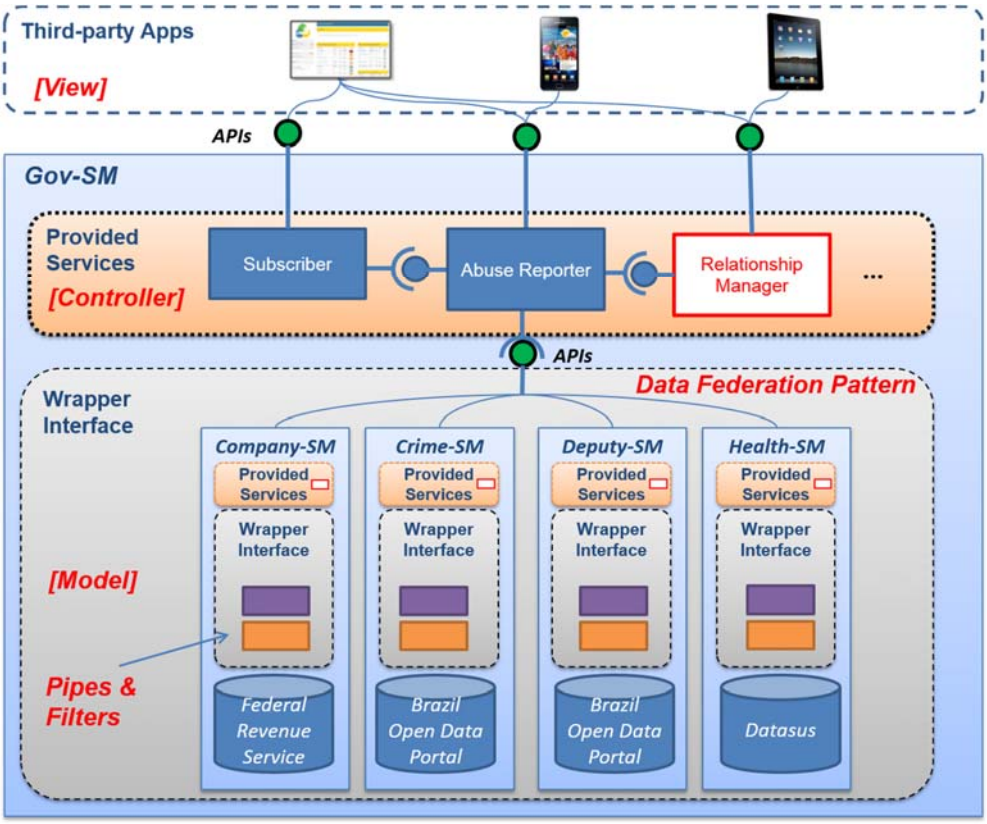

Figure 6.20 - GovSM: architecture overview

It is worth noting that for a better understanding of our approach, some details of our SM model were hidden away in Figure 6.20, and only the provided services and wrapper interface elements were explored in the high level architecture diagram. Essentially, the Gov-SM defines a unified model to wrap and deal with both structured and unstructured data from multiple disparate sources of government open datasets.

Additionally, the Gov-SM platform comprises a set of internal SMs (Table 6.8) that together provide dynamic sets of specialized APIs in order to support the development of *third-party* applications build atop of Gov-SM's services. The whole system is therefore a "relationship-aware" social machine (BURÉGIO et al., 2013b). That is why it represents an enabler for creating an ecosystem of possibly related and interacting applications and services.

In such ecosystem the relationships between third-party apps and Gov-SM should be established according to the model described in Chapter 4. Component *Relationship Manager* is responsible for mediating the establishments of such relationships. Hence, prior to access Gov-SM's services,

developers need to perform a registration process to create the desired relationship between Gov-SM and his/her application. The steps of this registration process follows the same default sequence of actions adopted by the UML sequence diagram of the component *Relationship Manager introduced in* Section 5.5.

During the registration process, developers should fill out a form provided by the Gov-SM's *Relationship Manager* which asks for basic information about the application, such as its name, domain, category and so on. The next step is to inform the desired relationship properties. In this step, the developer should choose, among other things, the permission his/her application will need. Finally, the confirmation is sent and the established *constraints* (e.g., rate limiting) of the relationship between the registered app and Gov-SM is approved.

In this environment, the possibilities of interactions among related parties (i.e., end-users, developers and applications) might potentiate the creation of large-scale social initiatives by combining the existing loosely-coupled SMs in a crowd-powered effort on the Web.

.

6.6.4. Discussion

There are no doubts that open governments practices need to be revisited in preparation for building a unified platform that indeed promote transparency, citizen participation, and collaboration. In this case study, we discussed some issues on existing open government initiatives and then used the Social Machine paradigm to support the process of deriving Government as a Social Machine. By combining computational and social processes into a composite, crowd-powered platform, the SM paradigm can significantly extends the power of open government initiatives, while requiring only a proper combination of patterns to manipulate available open datasets. The Social Machine proposed in this case study supports the fully integration of *users*, *developers* and *crowd* in order to participate in and solve current and future governmental problems.

6.7. Opinion Survey Based on Practical Experiences

As aforementioned, our evaluation process consists of 3 stages (see Section 6.1). So far, we discussed Stage 1 (Section 6.2) and Stage 2 (three case studies). In this Section, we present Stage 3 which consists of an *Opinion Survey*. While previous stages were carried out to understand and discuss *how* to derive Social Machines in different contexts, this section aims to collect the opinions of other people that have also been used our proposed paradigm in other practical Social Machine projects. The idea is to attest some findings of the literature and make clearer what people consider as the main benefits and limitations of our proposal.

6.7.1. Research Methodology

In the context of software engineering, *interviews* and *questionnaires* are commonly used techniques to collect data and evaluate a variety of aspects of software development (LETHBRIDGE; SIM; SINGER, 2005; SINGER; SIM; LETHBRIDGE, 2008). Hence, we use these techniques to collect data concerning individual's opinions, also referred to as *Opinion Survey* (KITCHENHAM; CHARTERS, 2007).

According to them, surveys are likely the most well-known and used method for researchers to gather relevant information about products, processes, services and so on. In fact, even in our daily life we are often asked to participate in a number of different surveys

In practice, there are different types of surveys: interview, surveys based on observing participant behavior and polling. It is worth noting that a survey is not only the questionnaire itself (i.e., the instrument), it includes the whole research process for planning questions, collecting, grouping, comparing and explaining information.

In our case, we performed an opinion survey to elicit the *benefits* and *limitations* that identified by other people when they used our proposed approach to develop practical Social Machines.

6.7.2. The survey

We focused on the type of survey in which data is collected by means of an online questionnaire filled in by the participants. The survey was performed after some preliminary experiences as well as the establishment of basic concepts (Chapter 3) and guidelines for the development of Social Machines (Chapters 4 and 5).

6.7.2.1. Precondition

As aforementioned, the main goal of this survey is to evaluate how the proposed Social Machine paradigm is viewed by other people. Hence, the precondition for answering this opinion survey was to have already used our approach in the development of some practical Social Machine which fits into our classification scheme (Section 3.3).

6.7.2.2. Target audience

The target audience was formed by people from both academy and software industry. On one hand, to support the academic context, two practical graduate courses on advanced topics in Software Engineering in 2012 and 2013. These courses were focused on the theory and practice of Social Machines which included the development of real practical systems using the proposed concepts. On the other hand, as a way to gather information from people in the industrial context, two enterprises based on the *Porto Digital ICT cluster*[49] were considered, namely SODET[50] and USTO.RE[51].

6.7.2.3. Types of questions

Given the open nature of this survey, most of its questions were designed as *open-ended* questions. This is because open-ended question increases the chances to gather not only the information foreseen, but also unexpected types of information. Figure 6.21 shows two examples of questions in which the open nature was totally (Figure 6.21 (a)) and partially (Figure 6.21 (b)) taken into account. In Figure 6.21 (b), although we have foreseen a predefined set of desired properties of the SoMAr style (see Subsection 5.3.3), it is possible to add other obtained properties, as the subject sees fit.

[49] http://www.portodigital.org/
[50] *SODET: Shifting Business into Social Machines* - http://sodet.biz
[51] *USTO.RE: Private Cloud Storage* – http://usto.re

Figure 6.21 - Examples of Questions with an open nature

It is worth noting that beyond collecting mostly subjective data, i.e., concerning personal opinions, we also designed some specific objective questions, concerning for example the number of people involved in a project.

6.8. The Survey Results

We had a total of 19 subjects who answered the survey and satisfied our requirement of having participated in a practical development project using our Social Machine approach. This section presents the analysis of the data collected in the survey, discusses each issue and highlights some correlation points that must be considered.

6.8.1. Audience Experience and Expertise

Initially, we asked the subjects about how many years of experience they have with software development. Figure 6.22 shows the results and gives us an idea about the maturity of the audience.

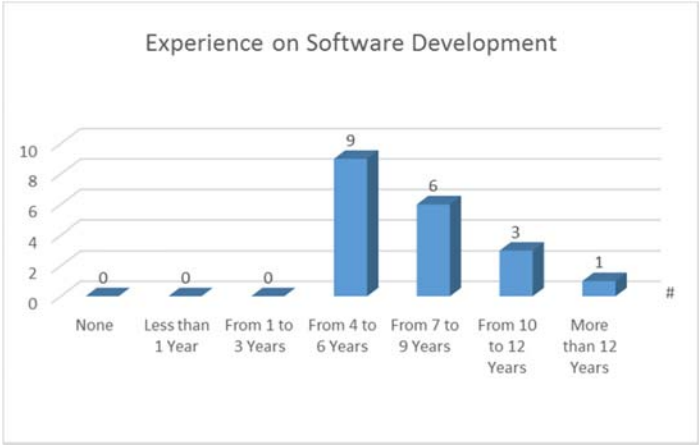

Figure 6.22 – Subjects Experience on Software Development

Also regarding the audience, we also asked them about the roles that they played in the Social Machine development project they participated in. The idea was try to understand the other responses under the perspective of the audience expertise. Figure 6.23 shows this distribution. In general we have more technical roles, but S*oftware engineer*, *System Analyst* and *Software Architect* were the most frequent roles. It worth noting that some people performed multiple roles in the same project.

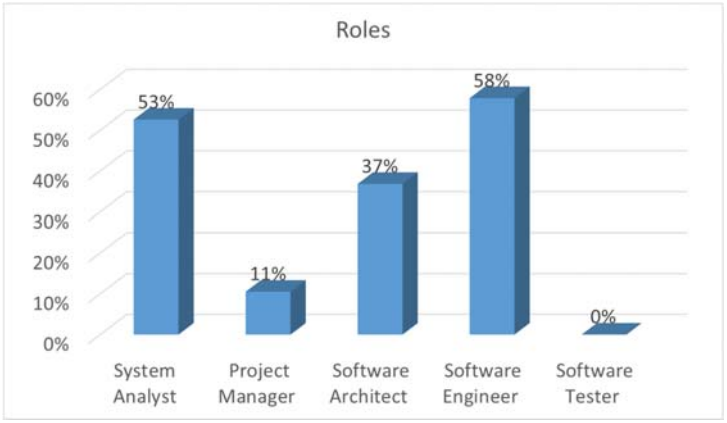

Figure 6.23 - Roles of the subjects

6.8.2. Projects

We asked the subjects to provide information about the projects they participated in. After grouping the responses we realized a total of 6 different projects as shown in Table 6.10. As can be seen, 4 (four) projects were developed in the academic contexts (i.e., *Lookatme, ReviewIt, SMADL* and *WhatHere*) and 2 (two) in the industrial context, namely *DWARF* and *uCloud*.

6.8.2.1. Project Classification

Considering the converging diagram of the different research visions of social machines presented in Chapter 3, we also asked the in which part of the diagram their projects better take place. Figure 6.24 shows the result of this mapping process. It is worth noting that different participants of the same project mapped their project in the same way. This can suggest a good coherence of our classification scheme. Another important thing is that, although we do not have any system in the intersection between "*People as Computational Units*" and "*Software as Sociable Entities*", the analyzed projects form a good set of representative Social Machines in the different visions.

Table 6.10 - Projects Overview

Project	Description	Context
DWARF[1]	DWARF defines a framework for developing Social Machines by applying Event-Driven Architecture patterns. DWARF is a platform for abstracting real-time communication between external software, apps that use its platform's API and each node in the cluster. The benefits of this platform include allowing applications to govern the business rules of communication, data transfer and behaviour invocation while abstracting concurrency handling and network / infrastructure minutia.	*Industry*
Lookatme[2]	Lookatme is a Open Mapping Platform that bridges the gap between buyers and sellers. It matchs buyers' needs to products and services offered by sellers. Lookatme is a Social Machine by design and uses other Social Machines to provide different kinds of social search engines and matchers that use context information such as localization, reputation, recommendation, user's social graph and so on.	*Academia*
ReviewIt[3]	A proposta do ReviewIt era a criação de um portal de revisão colaborativa de documentos. Nele os usuários poderiam postar seus documentos e ter ajuda da comunidade para revisá-lo. ReviewIt is a collaborative document review tool in which users can submit their documents and benefit from the crowd that revise the documents' content in a collaborative way.	*Academia*
SMADL	SMADL is an Architecture Description Language (ADL) based on the Social Machine model. Ideally, the language can be used to describe Web-based systems, mainly those where their APIs are made publicly available to produce and/or consume RESTful services.	*Academia*
Ucloud[4]	Ucloud provides a console for creating and managing data centers as a way of offering Data Center as a Service, in which all data center's elements are virtualized. A Domain-Specific Language based on the Social Machine abstraction model is used to specify data centers as relationship-aware Social Machines. uCloud also uses an orchestrator Social Machine which is responsible for the management of data centers and other connected entitites.	*Industry*
WhatHere	WhatHere is an application to help people not familiarized with a city or city area. Tourists, business people or citizens benefit from WhatHere through gatherig information about places nearby them. By using a smartphone or a traditional web browser, the application allows user to see nearby places (registered on google places or foursquare services) and browse on a combination of useful information related to the place, such as: wikipedia' content, information from google search , photos from Flickr, online comments on twitter about (or post by people at) the interested place.	*Academia*

1 - http://dwarf.hyperreactive.com
2 - http://lookatmeproject.blogspot.com.br/
3 - http://reviewit.elasticbeanstalk.com/
4 - Powered by UsTore (http://usto.re/)

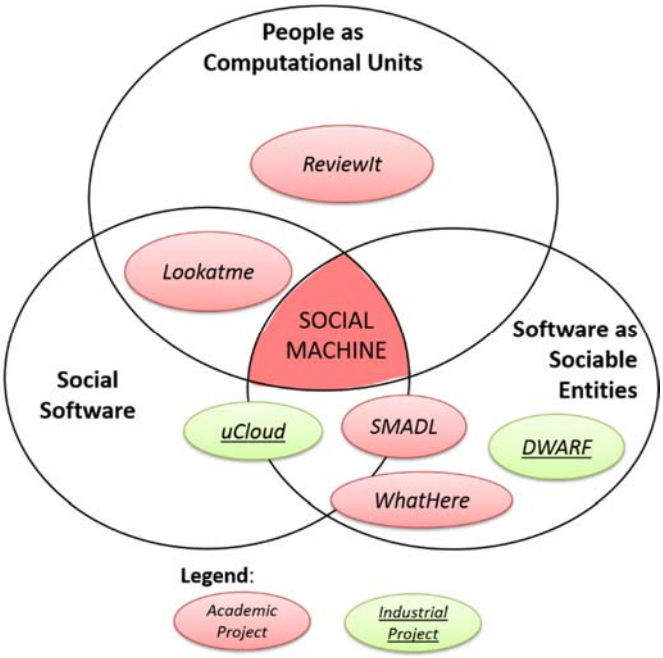

Figure 6.24 – Informed Classification of projects

6.8.2.2. Social Machine's building blocks

Considering our guideline about defining building blocks to be wrapped as a Social Machine (Section 5.4), we asked about the subjects wrapped as a Social Machine in their systems. Table 6.11 shows the result of this question.

Table 6.11 - Parts of the Systems were wrapped as Social Machines

	Wrapped as a SM		
Project	*The system as a whole*	*Internal Parts of the System*	*People*
DWARF	X		
Lookatme	X		
ReviewIt	X		X
SMADL	X		
Ucloud		X	
WhatHere	X	X	

It is clear that most projects are solutions in which the system as a whole was designed as a Social Machine. Meanwhile, it might indicate some open opportunities, given few projects deal with inner Social Machines or even do no consider people as a computing units.

6.8.3.Limitations

We can group the drawbacks and limitations provided by the participants as follows:

- **Lack of implementations**. Social Machine paradigm lacks real world implementations and the use cases that can be proven benefits are not yet widely spread.
- **Cost of Composite Social Machines**. It is still difficult and hardware expensive to deploy each inner Social Machine as an independent computing unit. In practice, the solution often deploys several SM's on the same environment, which sometimes can represent performance issues.
- **Technologies**. The adoption of existing technologies (languages, architectures, design patterns, infrastructure) to be SM compatible was also pointed as one limitation of the paradigm. It is necessary to provide tools that could facilitate the addition of the "sociability" layer around different types of computing units.
- **Learning curve Vs Productivity**. With the current literature the initial learning curve was pointed as one thing that impacts the productivity during the development of Social Machines.
- **Availability**. There is a lack of guidelines in the literature about the challenges related to availability issues of SM's external dependencies. One mentioned that his Social Machine would be totally inoperative if Google Maps is unavailable.
- **Dependability**. Also related to the previous issue, the high level of dependability is a big challenge to be investigated in the context of Social Machines. They mentioned some malfunction of the Twitter's API they used, as an example of dependability issue they faced during the development and operation of their Social Machines. In this context, application neutrality and loose coupled were cited as part of the strategies to overcome the dependability issues.
- **Reliability**. One of the main risks of SMs that combine different sources of data and services is related to ensuring reliability. Commercial restrictions change over time and consequently some

Web services consumed by SMs can change as well, possibly impacting the system's reliability at all. Certain families of composite SMs can increase the existing reliability issues of service-oriented solutions. As an example of that, we can highlight a possible change in the rate limits of APIs provided by Facebook, Twitter and other open platforms. The reliability of lots of SMs can be impacted by these kinds of changes.

- **Security and privacy.** Since security and privacy aspects both include business aspects such as roles, resources, processes and services as well as technology aspects such as applications, data, platform and infrastructure, ensuring security around the entire concept of Social Machines is very work intensive as well as costly.

6.8.4.Benefits

Prior to summarize the quantitative analysis, we can group and highlight some answers gave for the following open question: ***"Have you realized any benefit associated to the use of the SM paradigm? If so, which ones you could cite?"***

- **Modularization and Reuse**. Modularization and reuse were mentioned as a relevant benefit from applying our model. In this sense, we can highlight the comment of one of the participants who affirms the following:

> *"I think that the major benefit is the system modularization, allowing me reuse several parts in another system. In addition, it is easy to focus on specific problems and to share work. For instance, I can hire some specialist in external API's and he/she doesn't need to understand the entire systems, only the wrapper."*

- **Relationships as facilitators of constraints specification**. The facility to specify constraints between machines was also identified as an important benefit. In this sense, one participant commented the following:

> *"As we describe the interactions between SM entities in terms of relationships, it is easier to setup constraints upon them. That means the*

relationships themselves may have their own properties, avoiding to change the particular code of each involved social machine."

- **Application neutrality and loose coupling.** A participant said that

"The upfront gain of using the SM concept is 'decoupling', which is a generally pursued software engineering principle. This is achieved by using techniques of, for example, REST."

Interoperability, *encapsulation* and *maintainability* were also confirmed by some participants as beneficial attributes obtained from applying the proposed reference architecture into their solutions.

- **Relationships and interactions.** Relationships and interactions were indeed cited as important aspects to be considered in the engineering of Social Machines. In this context, it is worth highlighting what one of the senior software engineer said:

"By considering the communication between services and the interaction of people with these services, made it possible to realize the benefits of using social machines. Third-part services are becoming the new web infrastructure, similar to software components that are reused as well. This way of building new systems by combining several services of third-parties are not presented in the literature with a mature use of techniques and processes that address the new challenges intrinsic of this emerging area. All these factors were seen as a beneficial learning, in relation to the market itself and possibilities for it."

- **Being "prosumers" as a beneficial property.** The fact that Social Machines can represent entities that act at the same time as "providers" and "consumers", led some participants to state that:

"Creating entities that provide and at the same time consume existing services allows phenomena such as higher productivity and expansion of the base of users. In the lookatme project, for example, the [re]use of Google Maps' services were fundamental to delivery ours products in time. If we were supposed to implement such service it would not be possible to release any version of our product within the specified period of time. Regarding the expansion of the base of users, consuming services from well-known social

networks such as 'Facebook' and proving enhanced services on top of them, allows us to attract a huge number of users who already use such social network."

- **Blending of patterns to deal with integration issues.** Some benefits related to the proposed reference architecture were also cited by some participants, like the following:

"The reference architecture for social machines helped us to mitigate some risks, mainly the ones related to the huge variety of data types, formats and structures provided by existing systems to be wrapped. The combination of different design patterns has indeed proven to be a good option for dealing with this and other integration issues."

6.8.4.1. Obtained Properties

During the specification of SoMAr (Section 5.3), we defined a set of possible desired properties that could be obtained by using the *principles* and *constraints* specified by the SoMAr style. In order to evaluate such set of properties, we asked the following question to the participants: "***When using the SoMAr style, have you get any of its desired property? If do, which ones have you realized?***"

As aforementioned in Section 0, this question was designed in a way that beyond the default the set of properties, the participant could inform any other property, as they see fit. Figure 6.25 shows this distribution.

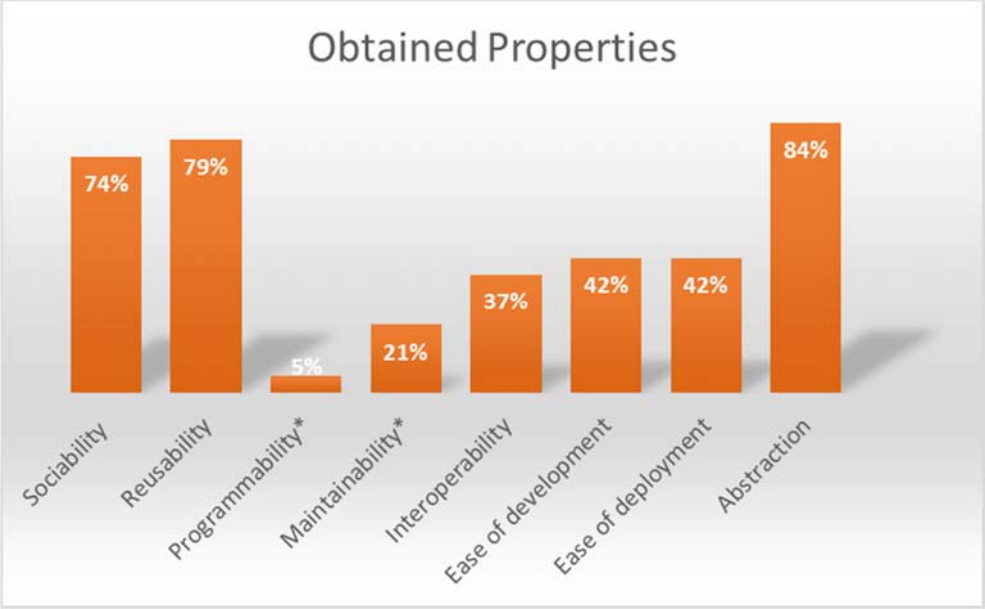

Figure 6.25- Obtained Properties according to the Opinion Survey

As can be seen, the top three properties indicated by the participants were *Abstraction*, *Reusability* and *Sociability*. New properties like *Programmability*[52] and also *Maintainability* were also added to the set of properties. In this case, we can make an analysis of the specific features considered in each project. For example, *Programmability* was a property added by people involved with the development of the SMADL project, which is an Architectural Description Language and in this case *Programmability* is more evident than in the other projects. Another interesting point is related to the participants' expertise. As we did not have any "*Software Tester*" among the participants (Section 6.8.1), we could realize that properties directly related to such discipline were not mentioned, such as, e.g., "*Testability*".

6.9. Concluding Remarks

In this chapter we outlined our experience on using the Social Machine paradigm to design and implement practical emerging social systems. We outlined our evaluation process, presented *Futweet* as our preliminary

[52] According to the The Free Dictionary (http://www.thefreedictionary.com/), "Programmability" refers to the "*capability within hardware and software to change; to accept a new set of instructions that alter its behavior*."

experience, discussed three case studies of applying Social Machines into different contexts (i.e., *individuals*, *businesses* and *governments*) and finally presented the outcomes obtained from an *Opinion Survey* on seven practical experiences on adopting the proposed concepts.

Even based on a small subset of practical implementations and considering the number of mentioned limitations, we can conclude that our proposal can produce significant benefits as well as enable the creation of Web-enabled systems that have, or may yet soon have, a profound impact on the lives of *individuals*, *businesses*, *governments*, and the society as a whole in substantial ways.

7 Conclusions and Future Developments

"If I have seen a little farther than others, it is because I have stood on the shoulders of giants"

Isaac Newton (1642 – 1727)
English mathematician & physicist

This chapter presents the final remarks about this thesis, by highlighting the main contributions and presenting possible future work.

7.1. Concluding Considerations

This thesis presented an approach to incorporate social aspects into software, leading to the notion of Social Machines. In this context, Social Machines were proposed as a unified paradigm to describe, design and implement emerging social systems.

To enable the development of Social Machines, a common base of understanding was established and a unified abstraction model was defined as well. On top of such model, some engineering guidelines were specified in conjunction with a combination of *principles*, *constraints* and *properties* that drive the design and implementation of what we refer to as *SoMAr* (Social Machine-oriented Architecture). *SoMAr* is an architectural style which, among other things, establishes a reference architecture formed by the blending of different integration patterns to support the development of composite Social Machines.

We applied the Social Machine paradigm into different contexts and discussed how these systems can influence *individuals*, *business*, *governments*, and the society as a whole. Furthermore, an *opinion survey* was performed with the aim of collecting opinions of people that have been using our proposed paradigm in practical Social Machine projects, as a way to confirm some

findings from the literature and make clearer what people consider as the main benefits and limitations of our proposal.

In conclusion, we can say that, although some existing limitations and challenges, our unified paradigm indeed helps converging the different visions of Social Machines (BURÉGIO et al., 2013a), referred to as *"Social Software"*, *"People as computational units"* and *"Software as Sociable Entities"*. Exemplifying, we can assume our case study on *Government as a Social Machine* (BURÉGIO et al., 2014b) in order to state that:

1. The "social software" vision is achieved by implementing applications on top of the governmental Social Machine with the aim of providing two-way communication channels between governments and their citizens (as *users*), leading to different levels of social interactions between them;
2. The "people as computational units" vision is achieved by using the resultant crowd-powered platform as the basis to launch different kinds of initiatives that encourage the *crowd* to solve numerous governmental issues and policy;
3. Last but not least, the "software as sociable entities" vision is achieved by providing dynamic sets of specialized APIs that naturally conduct to the establishment of an ecosystem of possibly related and interacting applications and services, built by *developers* with a passionate interest in a more effective public oversight.

Government as a Social Machine is just one of the examples we discussed in this work. Thus, in more than one sense, the approach proposed herein offers different avenues of possibilities that converge to the full integration of *software*, *things*, *developers* and *crowd* in order to participate in and solve a multitude of current (and future) issues in diverse areas of a society ever more formed by different *socially connected computing units*, also known as Social Machines.

7.2. Future Work

Future work includes:

- Implement more sophisticated mechanisms to fully support *inferred relationships* based on real relations between people;
- Define a framework to provide dynamic adaptations for dealing with availability issues;
- Extend the *[YOU]-SM* to allow the *focus* service combines other source of data;
- Design strategies to support a *social search engine* in the reference architecture of social machines;
- Define a security framework to deal with privacy and ownership in the context of Social Machines;
- Characterize the different social actions and social artifacts that can take place in the context of the Social Enterprise (it is part of an on-going joint-initiative);
- Extend the Gov-SM to create an even more comprehensive framework - the *GOvernment Open Data* (GOOD) framework - to tackle other aspects of research inquiries;
- Create an environment for the development of Social Machines with a new language supported by visual tools;
- Apply the unified Social Machine model into machine-to-machine communications as a way to enable the "*Internet of things";*
- Apply the concept of Social Machine to the context of smart cities;
- Enable the creation of "learning social machines", in which learning processes could be integrated with the social and computing ones.

References

AGRAWAL, A. et al. **Web services human task (WS-HumanTask), version 1.0**, 2007a.

AGRAWAL, A. et al. **WS-BPEL Extension for People (BPEL4People), Version 1.0**, 2007b.

AGRAWAL, S.; DAS, M. L. **Internet of Things — A paradigm shift of future Internet applications**2011 Nirma University International Conference on Engineering. **Anais**...IEEE, dez. 2011Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6153246>. Acesso em: 14 nov. 2013

ANDERSEN, T. B. E-Government as an anti-corruption strategy. **Information Economics and Policy**, v. 21, n. 3, p. 201–210, 2009.

ANDERSON, P. et al. On managing social data for enabling socially-aware applications and services. **Proceedings of the 3rd Workshop on Social Network Systems - SNS '10**, p. 1–6, 2010.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, out. 2010.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice (3rd Edition) (SEI Series in Software Engineering)**. [S.l.]: Addison-Wesley Professional, 2012. p. 640

BENIOFF, M. **Welcome to Web 3.0: Now Your Other Computer is a Data Center | TechCrunch**. Disponível em: <http://techcrunch.com/2008/08/01/welcome-to-web-30-now-your-other-computer-is-a-data-center-2/>. Acesso em: 4 out. 2013.

BENTAHAR, J. et al. An Argumentation Framework for Communities of Web Services. **IEEE Intelligent Systems**, v. 22, n. 6, p. 75–83, nov. 2007.

BERNERS-LEE, T. WWW: past, present, and future. **Computer**, v. 29, n. 10, p. 69–77, 1996.

BERNERS-LEE, T. **Weaving the Web**. New York, NY, USA: Harper Collins, 1999.

BERTOT, J. C.; JAEGER, P. T.; GRIMES, J. M. Using ICTs to create a culture of transparency: E-government and social media as openness and anti-corruption tools for societies. **Government Information Quarterly**, v. 27, n. 3, p. 264–271, 2010.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked Data - The Story So Far. **International Journal on Semantic Web and Information Systems**, v. 5, n. 3, p. 1–22, 2009.

BOHAN BRODERICK, P. On Communication and Computation. **Minds and Machines**, v. 14, n. 1, p. 1–19, fev. 2004.

BRITO, K. et al. **Brazilian Government Open Data: Implementation, Challenges, and Potential Opportunities**International Digital Government Research Conference. **Anais**...2014

BRITO, K. S. et al. Implementing Web Applications as Social Machines Composition : a Case Study. **The 24th International Conference on Software Engineering and Knowledge Engineering**, v. (SEKE'2012, p. 311–314, 2012.

BUDGEN, D. et al. Using mapping studies in software engineering. **Proceedings of PPIG**, v. 2, p. 195–204, 2007.

BURÉGIO, V. et al. Social Machines: a Unified Paradigm to Describe Social Web-oriented Systems. **22nd International World Wide Web Conference (WWW 2013 Companion)**, p. 885–890, 13 maio 2013a.

BURÉGIO, V. et al. **Personal APIs as an Enabler for Designing and Implementing People as Social Machines**23rd International World Wide Web Conference (WWW 2014 Companion). **Anais**...Seoul, Korea: 2014a

BURÉGIO, V. et al. Towards Government as a Social Machine. **to be published**, 2014b.

BURÉGIO, V. A. et al. **Moving Towards " Relationship-aware " Applications and Services: A Social Machine-oriented Approach**17th IEEE International EDOC Conference (EDOCW 2013). **Anais**...Vancouver, Canada: 2013b

BUREGIO, V. A.; MEIRA, S. R. L.; ALMEIDA, E. S. **Characterizing Dynamic Software Product Lines: A Preliminary Mapping Study**Software Product Line Conf. (SPLC 10). **Anais**...Jeju Island, South Corea: Lancaster University, 2010

BURÉGIO, V.; MAAMAR, Z.; MEIRA, S. An Architecture and Guiding Framework for the Social Enterprise. **IEEE Internet Computing**, v. 19, n. 1, p. 64–68, 2015.

BURGIN, M. **Super-Recursive Algorithms**. New York, NY: Springer Monographs in Computer Science, 2005.

CHEN, H. et al. Using Open Web APIs in Teaching Web Mining. **IEEE Transactions on Education**, v. 52, n. 4, p. 482–490, nov. 2009.

CHEN, P. P.-S. The entity-relationship model---toward a unified view of data. **ACM Transactions on Database Systems**, v. 1, n. 1, p. 9–36, 1 mar. 1976.

CHUNG, J.-Y. **An Industry View on Service-Oriented Architecture and Web Services**IEEE International Workshop on Service-Oriented System Engineering (SOSE'05). **Anais**...IEEE, 2005Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1551130>. Acesso em: 15 nov. 2013

DA SILVA, B. N.; CRISTINA, A.; GARCIA, B. KA-CAPTCHA: an opportunity for knowledge acquisition on the web. p. 1322–1327, 22 jul. 2007.

DADA, D. The Failure of E-Government in Developing Countries: A literature review. **The Electronic Journal of Information Systems in Developing Countries**, v. 26, n. 7, p. 1–10, 2006.

DANIEL, S.; PRZEMYSLAW, T.; LARS, H. Integrating Information Systems Using Web Oriented Integration Architecture and RESTful Web Services. **2010 6th World Congress on Services**, p. 598–605, jul. 2010.

DEMARCO, T. **Structured Analysis and System Specification**. [S.l.]: Prentice Hall, 1979. p. 348

DODIG-CRNKOVIC, G. Significance of Models of Computation, from Turing Model to Natural Computation. **Minds and Machines**, v. 21, n. 2, p. 301–322, 2 fev. 2011.

DONATH, J. **The Social Machine: Designs for Living Online**. [S.l.]: The MIT Press, 2014. p. 432

DUFFY, B. R. Fundamental Issues in Affective Intelligent Social Machines. **The Open Artificial Intelligence Journal**, v. 2, n. 1, p. 21–34, 3 jun. 2008.

DUSTDAR, S.; BHATTACHARYA, K. The Social Compute Unit. **IEEE Internet Computing**, v. 15, n. 3, p. 64–69, maio 2011.

DUVANDER, A. **Twitter Reveals: 75% of Our Traffic is via API (3 billion calls per day)**. Disponível em: <http://blog.programmableweb.com/2010/04/15/twitter-reveals-75-of-our-traffic-is-via-api-3-billion-calls-per-day/>.

DUVANDER, A. **6,000 APIs: It's Business, It's Social and It's Happening Quickly**. Disponível em: <http://blog.programmableweb.com/2012/05/22/6000-apis-its-business-its-social-and-its-happening-quickly/>.

ERL, T. **Service-oriented architecture: concepts, technology, and design**. [S.l.]: Prentice Hall PTR, 2005. p. 792

ERL, T. **SOA: Principles of Service Design**. [S.l.]: Prentice Hall, 2007. p. 608

FIELDING, R. **Architectural Styles and the Design of Network-based Software Architectures**. [S.l.]: UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

FIELDING, R. T.; TAYLOR, R. N. **Architectural Styles and the Design of Network-based Software Architectures**. [S.l.]: Citeseer, 2000.

GARLAN, D.; SHAW, M. An Introduction to Software Architecture. 1 jan. 1994.

GHADERI, M. A.; YAZDANI, N.; MOSHIRI, B. **A social network-based meta search engine**2010 5th International Symposium on Telecommunications. **Anais**...IEEE, dez. 2010Disponível em:

<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5734121>. Acesso em: 12 maio. 2014

GIBBINS, N.; HARRIS, S.; SHADBOLT, N. Agent-based Semantic Web Services. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 1, n. 2, p. 141–154, fev. 2004.

GOYAL, P. **Issues in the adoption of object-oriented paradigm**COMPCON Spring '91 Digest of Papers. **Anais**...IEEE Comput. Soc. Press, 1991Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=128862>. Acesso em: 11 nov. 2013

GRAY, D. Everything is a service. **The Social Business Journal**, n. 01, p. 14–21, 2012.

HALB, W.; RAIMOND, Y.; HAUSENBLAS, M. Building Linked Data For Both Humans and Machines. **In proceedings of the Linked Data on the Web Workshop**, 2008.

HARRISON, T.; SAYOGO, D. **Open budgets and open government: beyond disclosure in pursuit of transparency, participation and accountability**International Conference on Digital Government Research. **Anais**...2013Disponível em: <http://dl.acm.org/citation.cfm?id=2479757>. Acesso em: 24 fev. 2014

HAYES, B. Cloud computing. **Communications of the ACM**, v. 51, n. 7, p. 9, jul. 2008.

HAZRA, K. **Cloud computing - the next chasm**, 2009.

HEINEMAN, G. T.; COUNCILL, W. T. **Component-Based Software Engineering: Putting the Pieces Together**. [S.l.]: Addison-Wesley Professional, 2001. p. 880

HENDLER, J.; BERNERS-LEE, T. From the Semantic Web to social machines: A research challenge for AI on the World Wide Web. **Artificial Intelligence**, v. 174, n. 2, p. 156–161, fev. 2010.

HITZLER, P.; KRÖTZSCH, M.; RUDOLPH, S. **Foundations of Semantic Web Technologies (Chapman & Hall/CRC Textbooks in Computing)**. [S.l.]: Chapman and Hall/CRC, 2009. p. 456

HOARE, C. A. R.; JIFENG, H. **Unifying Theories of Programming**. [S.l.]: Prentice Hall College Div, 1998. p. 320

HOST, M.; RUNESON, P. Checklists for Software Engineering Case Study Research. **First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)**, p. 479–481, set. 2007.

HUNG, S.-Y.; CHANG, C.-M.; YU, T.-J. Determinants of user acceptance of the e-Government services: The case of online tax filing and payment system. **Government Information Quarterly**, v. 23, n. 1, p. 97–122, 2006.

HWANG, J.; ALTMANN, J.; KIM, K. The structural evolution of the Web 2.0 service network. **Online Information Review**, v. 33, n. 6, p. 1040 – 1057, 2009.

IAMNITCHI, A.; BLACKBURN, J.; KOURTELLIS, N. The Social Hourglass: An Infrastructure for Socially Aware Applications and Services. **IEEE Internet Computing**, v. 16, n. 3, p. 13–23, 29 maio 2012.

ISSARNY, V. et al. Service-oriented middleware for the Future Internet: state of the art and research directions. **Journal of Internet Services and Applications**, v. 2, n. 1, p. 23–45, 25 maio 2011.

JACOBS, I.; JAFFE, J.; HEGARET, P. LE. How the Open Web Platform Is Transforming Industry. **IEEE Internet Computing**, v. 16, n. 6, p. 82–86, nov. 2012.

JHO, W. Challenges for e-governance: protests from civil society on the protection of privacy in e-government in Korea. **International Review of Administrative Sciences**, v. 71, n. 1, p. 151–166, 2005.

JIANDONG CAO; YANG TANG; BINBIN LOU. **Social search engine research**2010 3rd International Conference on Computer Science and Information Technology. **Anais**...IEEE, jul. 2010Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5563672>. Acesso em: 12 maio. 2014

KAJAN, E. et al. The Network-Based Business Process. **IEEE Internet Computing**, v. 18, n. 2, p. 63–69, 2014.

KIM, S.; KIM, H. J.; LEE, H. An institutional analysis of an e-government system for anti-corruption: The case of OPEN. **Government Information Quarterly**, v. 26, n. 1, p. 42–50, 2009.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. [S.l: s.n.].

KIWON LEE. **Technical architecture for land monitoring portal using google maps API and open source GIS**2009 17th International Conference on Geoinformatics. **Anais**...IEEE, ago. 2009Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5293060>. Acesso em: 4 out. 2013

KO, M. N. et al. Social-Networks Connect Services. **Computer**, v. 43, n. 8, p. 37–43, ago. 2010.

KORIS, N. J.; HODDINOTT, A. P. Using Web 2.0 applications to deliver innovative services on the internet. **2008 International Conference on Service Systems and Service Management**, p. 1–4, jun. 2008.

KUHN, T. S. **The Structure of Scientific Revolutions**. [S.l.]: University of Chicago Press, 1970. p. 210

LANTHALER, M.; GRANITZER, M.; GUETL, C. Semantic web services: state of the art. 2010.

LANTHALER, M.; GÜTL, C. Towards a RESTful Service Ecosystem Perspectives and Challenges. **Ecosystems**, p. 209–214, 2010.

LAU, K.-K.; WANG, Z. Software Component Models. **IEEE Transactions on Software Engineering**, v. 33, n. 10, p. 709–724, out. 2007.

LEE, D. **Facebook surpasses one billion users as it tempts new markets**. Disponível em: <http://www.bbc.co.uk/news/technology-19816709>.

LEE, J.; KOTONYA, G.; ROBINSON, D. Engineering Service-Based Dynamic Software Product Lines. **Computer**, v. 45, n. 10, p. 49–55, out. 2012.

LEIBA, B. OAuth Web Authorization Protocol. **IEEE Internet Computing**, v. 16, n. 1, p. 74–77, jan. 2012.

LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying Software Engineers: Data Collection Techniques for Software Field Studies. **Empirical Software Engineering**, v. 10, n. 3, p. 311–341, 1 jul. 2005.

LITTLE, J.; TOMPKINS, T. Open Government Laws: An Insider's View. **North Carolina Law Review**, v. 53, p. 451, 1974.

LIU, Y. et al. **Using architecture integration patterns to compose enterprise mashups**2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. **Anais**...IEEE, set. 2009Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5290797>. Acesso em: 27 dez. 2013

LUIS VON AHN, M. B. et al. CAPTCHA: Using Hard AI Problems for Security. **Advances in Cryptology—EUROCRYPT 2003**, Lecture Notes in Computer Science. v. 2656, p. 646–646, 13 maio 2003.

MAAMAR, Z. Commerce, e-commerce, and m-commerce. **Communications of the ACM**, v. 46, n. 12, p. 251, 1 dez. 2003.

MAAMAR, Z. et al. **WEB SERVICES COMMUNITIES - Concepts & Operations**WEBIST'2007. **Anais**...Barcelona, Spain: 2007

MAAMAR, Z. et al. **Even Web Services Can Socialize: A New Service-Oriented Social Networking Model**2009 International Conference on Intelligent Networking and Collaborative Systems. **Anais**...IEEE, nov. 2009Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5369118>. Acesso em: 24 nov. 2011

MAAMAR, Z. et al. LinkedWS: A novel Web services discovery model based on the Metaphor of "social networks. **Simulation Modelling Practice and Theory**, v. 19, n. 1, p. 121–132, 2010.

MAAMAR, Z. et al. Using Social Networks for Web Services Discovery. **IEEE Internet Computing**, v. 15, n. 4, p. 48–54, jul. 2011.

MAAMAR, Z.; BUREGIO, V.; MEIRA, S. From Business-Data Artifacts to Social-Data Artifacts. **to be published**, 2014.

MAAMAR, Z.; HACID, H.; HUHNS, M. N. Why Web Services Need Social Networks. **IEEE Internet Computing**, v. 15, n. 2, p. 90–94, mar. 2011.

MAARADJI, A.; HACID, H.; DAIGREMONT, J. Towards a Social Network Based Approach for Services Composition. **IEEE International Conference on Communications**, p. 1–5, 2010.

MAXIMILIEN, E. M.; RANABAHU, A.; GOMADAM, K. An Online Platform for Web APIs and Service Mashups. **IEEE Internet Computing**, v. 12, n. 5, p. 32–43, set. 2008.

MCAFEE, A. P. Enterprise 2.0: The Dawn of Emergent Collaboration. **MITSloan Management Review**, v. 47, n. 3, p. 21–28, 2006.

MCGOWAN, C. Structured Programming: a Review of Some Practical Concepts. **Computer**, v. 8, n. 6, p. 25–30, jun. 1975.

MCNAUGHT, C.; LAM, P. Using Wordle as a Supplementary Research Tool. **Qualitative Report**, v. 15, n. 3, p. 630–643, 30 abr. 2010.

MEIRA, R. S. D. L. et al. On the Internet , Privacy and the Need for a New Architecture of Networked Information Services. **Academia.edu**, 2013.

MEIRA, S. R. L. et al. The Emerging Web of Social Machines. **CoRR**, v. abs/1010.3, out. 2010.

MEIRA, S. R. L. et al. **The Emerging Web of Social Machines**2011 IEEE 35th Annual Computer Software and Applications Conference. **Anais**...IEEE, jul. 2011Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6032321&contentType=Conference+Publications&searchField=Search_All&queryText=.QT.The+emerging+web+of+social+machines.QT.>. Acesso em: 11 nov. 2011

MEYER, J. D. et al. **Microsoft® Application Architecture Guide (Patterns & Practices)**. [S.l.]: Microsoft Press, 2009. p. 560

MILES, R.; HAMILTON, K. **Learning UML 2.0**. [S.l.]: O'Reilly Media, 2006. p. 290

MITCHELL, D. The constitutional right to know. **Hastings Constitutional Law Quarterly**, v. 4, p. 109, 1977.

MURUGESAN, S. Understanding Web 2.0. **IT Professional**, v. 9, n. 4, p. 34–41, jul. 2007.

NASCIMENTO, L.; GARCIA, V.; MEIRA, S. **SMADL: The Social Machines Architecture Description Language**Paper presented at the meeting of the SLE (Doctoral Symposium). **Anais**...2012

NATH, K.; DHAR, S.; BASISHTHA, S. **Web 1.0 to Web 3.0 - Evolution of the Web and its various challenges**, 2014.

NICOL, J. R.; WILKES, C. T.; MANOLA, F. A. Object orientation in heterogeneous distributed computing systems. **Computer**, v. 26, n. 6, p. 57–67, jun. 1993.

NIGAM, A.; CASWELL, N. S. Business artifacts: An approach to operational specification. **IBM Systems Journal**, v. 42, n. 3, p. 428–445, 2003.

O'REILLY, T. **What Is Web 2.0 - O'Reilly Media**. Disponível em: <http://oreilly.com/web2/archive/what-is-web-20.html>.

PARKS, W. Open Government Principle: Applying the Right to Know Under the Constitution. **The George Wawhington Law Review**, v. 26, n. 1, 1957.

PATTAL, M. M. I.; LI, Y.; ZENG, J. **Web 3.0: A Real Personal Web! More Opportunities and More Threats**2009 Third International Conference on Next Generation Mobile Applications, Services and Technologies. **Anais**...IEEE, set. 2009Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5337255>. Acesso em: 18 abr. 2014

PATTERSON, D.; FOX, A. **Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing**. [S.l.]: Strawberry Canyon LLC, 2012.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. **ACM SIGSOFT Software Engineering Notes**, v. 17, n. 4, p. 40–52, 1 out. 1992.

PETERSEN, K. et al. Systematic Mapping Studies in Software Engineering. p. 1–10, 2007.

PICAZO-VELA, S.; FERNANDEZ-HADDAD, M.; LUNA-REYES, L. F. **IT's alive!!: social media to promote public healthProceedings of the 14th Annual International Conference on Digital Government Research**Quebec, CanadaACM, , 2013.

PIOTROWSKI, S. J.; VAN RYZIN, G. G. Citizen Attitudes Toward Transparency in Local Government. **The American Review of Public Administration**, v. 37, n. 3, p. 306–323, 1 set. 2007.

PREECHAVEERAKUL, L.; KAEWNOPPARAT, W. **A Novel Approach: Secure Information Notifying System Using RSS Technology**2009 International Conference on Future Networks. **Anais**...IEEE, mar. 2009Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5189906>. Acesso em: 15 nov. 2013

RECORDON, D.; HARDT, D.; HAMMER-LAHAV, E. The OAuth 2.0 Authorization Protocol. **IEEE Internet Computing**, v. 8, n. 1, p. 1–47, 2011.

ROMERO, S. et al. On the Internet , Privacy and the Need for a New Architecture of Networked Information Services. **on submission**, 2013.

ROUSH, W. Social Machines - computing means connecting. **MIT Technology Review, August**, p. 1–18, 2005.

ROUSH, W. (MIT). Social Machines. **Technology**, p. 1–18, 2006.

SAVOLAINEN, J.; MYLLARNIEMI, V. **Layered architecture revisited — Comparison of research and practice**2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. **Anais**...IEEE, set. 2009Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290685>. Acesso em: 25 maio. 2014

SAYOGO, D. S.; HARRISON, T. **Effects of the internet and sociocultural factors on budget transparency and accountabilityProceedings of the 13th Annual International Conference on Digital Government Research**College Park, MarylandACM, , 2012.

SAYRE, R. Atom: The Standard in Syndication. **IEEE Internet Computing**, v. 9, n. 4, p. 71–78, jul. 2005.

SCHALL, D.; DUSTDAR, S.; BLAKE, M. B. Programming Human and Software-Based Web Services. **Computer**, v. 43, n. 7, p. 82–85, jul. 2010.

SCHALL, D.; TRUONG, H.-L.; DUSTDAR, S. Unifying Human and Software Services in Web-Scale Collaborations. **IEEE Internet Computing**, v. 12, n. 3, p. 62–68, maio 2008.

SCHIEL, U.; MISTRIK, I. **Using object-oriented analysis and design for integrated systems**Systems Integration '90. Proceedings of the First International Conference on Systems Integration. **Anais**...IEEE Comput. Soc. Press, 1990Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=138672>. Acesso em: 14 nov. 2013

SEMMELHACK, P. **Social Machines: How to Develop Connected Products That Change Customers' Lives**. [S.l.]: Wiley, 2013. p. 245

SHADBOLT, N. Knowledge acquisition and the rise of social machines. **International Journal of Human-Computer Studies**, v. 71, n. 2, p. 200–205, fev. 2013.

SHADBOLT, N. R. et al. Towards a classification framework for social machines. p. 905–912, 13 maio 2013.

SHAW, M.; CLEMENTS, P. **A field guide to boxology: preliminary classification of architectural styles for software systems**Proceedings Twenty-First Annual

International Computer Software and Applications Conference (COMPSAC'97). **Anais**...IEEE Comput. Soc, 1997Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=624691>. Acesso em: 25 maio. 2014

SIMOES, J.; WAHLE, S. The future of services in next generation networks. **IEEE Potentials**, v. 30, n. 1, p. 24–29, jan. 2011.

SINGER, J.; SIM, S. E.; LETHBRIDGE, T. **Guide to Advanced Empirical Software Engineering**. London: Springer London, 2008.

SKOPIK, F. et al. Towards Social Crowd Environments Using Service-Oriented Architectures. **it - Information Technology**, v. 53, n. 3, p. 108–116, maio 2011.

SURYANARAYANA, G. et al. PACE: an architectural style for trust management in decentralized applications. **Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)**, p. 221–230, 2004.

SZYPERSKI, C. **Component Software: Beyond Object-Oriented Programming (2nd Edition)**. [S.l.]: Addison-Wesley Pub (Sd), 2002. p. 411

TAN, W. et al. Social-Network-Sourced Big Data Analytics. **IEEE Internet Computing**, v. 17, n. 5, p. 62–69, set. 2013.

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. **Software Architecture: Foundations, Theory, and Practice**. [S.l.]: Wiley, 2009. p. 750

THALER, S.; SIMPERL, E.; WÖLGER, S. An Experiment in Comparing Human-Computation Techniques. **Ieee Internet Computing**, 2012.

TOFFLER, A. **The Third Wave**. [S.l.]: William Morrow & Company, 1980. p. 544

TSAI, W. T.; ZUALKERNAN, I. **Object-oriented paradigm and software engineering**Proceedings COMPSAC 88: The Twelfth Annual International Computer Software & Applications Conference. **Anais**...IEEE Comput. Soc. Press, 1988Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=17209>. Acesso em: 15 nov. 2013

TURING, A. On computable numbers, with an application to the Entscheidungs problem. **Proceedings of the London Mathematical**, v. 38, p. 173–198, 1936.

TURNER, M.; BUDGEN, D.; BRERETON, P. Turning Software into a Service. **Computer**, v. 36, n. 10, p. 38–44, out. 2003.

UPADHYAYA, B. et al. **Migration of SOAP-based services to RESTful services**2011 13th IEEE International Symposium on Web Systems Evolution (WSE). **Anais**...IEEE, set. 2011Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6081828>. Acesso em: 15 nov. 2013

VON AHN, L. et al. reCAPTCHA: human-based character recognition via Web security measures. **Science (New York, N.Y.)**, v. 321, n. 5895, p. 1465–8, 12 set. 2008.

VON AHN, L.; DABBISH, L. Designing games with a purpose. **Communications of the ACM**, v. 51, n. 8, p. 57, 1 ago. 2008.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. **REST in Practice: Hypermedia and Systems Architecture**. [S.l.]: O'Reilly Media, 2010. p. 448

WONG, W.; WELCH, E. Does E-Government Promote Accountability? A Comparative Analysis of Website Openness and Government Accountability. **Governance**, v. 17, n. 2, p. 275–297, abr. 2004.

WS-REST. **WS-REST 2010 | First International Workshop on RESTful Design**. Disponível em: <http://ws-rest.org/2010/>.

YAHYAOUI, H. Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. **Knowledge Creation Diffusion Utilization**, v. 17, n. 5, p. 686–697, 2005.

YE ZHOU; YANG JI. **Design of rest APIS for the exposure of IMS capabilities towards Web services**IET International Conference on Communication Technology and Application (ICCTA 2011). **Anais**...IET, 2011Disponível em: <http://digital-library.theiet.org/content/conferences/10.1049/cp.2011.0724>. Acesso em: 4 out. 2013

YOURDON, E. **Modern Structured Analysis**. [S.l.]: Prentice Hall, 1988. p. 688

YU, J. et al. Understanding Mashup Development. **IEEE Internet Computing**, v. 12, n. 5, p. 44–52, set. 2008.

YU, S.; WOODARD, C. J. **Innovation in the programmable web: Characterizing the mashup ecosystem** (G. Feuerlicht, W. Lamersdorf, Eds.)Service-Oriented Computing ICSOC. **Anais**...: Lecture Notes in Computer Science.Berlin, Heidelberg: Springer Berlin Heidelberg, 21 abr. 2009Disponível em: <http://dl.acm.org/citation.cfm?id=1534121.1534136>. Acesso em: 20 jan. 2013

YUEN, M.-C.; CHEN, L.-J.; KING, I. A Survey of Human Computation Systems. **2009 International Conference on Computational Science and Engineering**, p. 723–728, 2009.

ZHANG, G.; LI, C.; XING, C. **A Semantic++ Social Search Engine Framework in the Cloud**2012 Eighth International Conference on Semantics, Knowledge and Grids. **Anais**...IEEE, out. 2012Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6391851>. Acesso em: 12 maio. 2014

Appendix A : K-RADAR – The Knowledge Radar to guide research efforts

This Appendix gives an overview of the Knowledge Radar (K-RADAR) - a monitoring approach we created for measuring the research progress of this thesis and mapping its involved knowledge.

A.1 Introduction

Doing research, especially conducting a master or Ph.D. thesis, can be a really big challenge for most students. Often, during the research development, students face some difficulties in the process of knowledge acquiring and keeping focus. Generally, it is because research studies involve several kinds of knowledge that must be acquired and consolidated in different levels of understanding. Thus, some focusing issues are raised, especially because i) not all identified knowledge is required for the research; ii) not all required knowledge should be investigated immediately and, at the same time, iii) a researcher should not lose sight of what has already been detected, because it can be useful in the future.

In order to guide our research, it is therefore essential to monitor and map field studies, i.e., to study real practitioners as they solve real problems. To aid this goal, we describe a series of data collection techniques for such studies, organized around a taxonomy based on the degree to which interaction with software engineers is necessary. Motivated by such issues, during this thesis we developed the K-RADAR – a Knowledge Radar - as an approach for monitoring the research progress and mapping its involved knowledge with the aim of guiding the research efforts.

A.2 Origins

K-RADAR came out after a request from Professor Dr. Silvio Meira (my advisor) to report the progress of my Ph.D. research.

Silvio's request: elaborate a report to make explicit...

1. *What you have learnt so far;*
2. *What you know that you do not know, but it is needed to learn;*
3. *What are the distant frontiers, now, the things you do not even know if you will need to learn, but are on the radar just in case.*

Thus, motivated by such questions, K-RADAR emerged as a monitoring method for measuring my research progress and mapping its involved knowledge.

A.3 Levels of Knowledge

We consider three main levels in the knowledge acquiring process: (1) *Detected Knowledge*; (2) *Needed Knowledge*; and (3) *Consolidated Knowledge*. Figure A.1 illustrates the dynamics of the K-RADAR approach. It gives us a photograph of somebody's knowledge about his/her research field based on the three levels aforementioned. Each point represents a specific topic. The closer to the center is the point, the more consolidated is the knowledge of its topic.

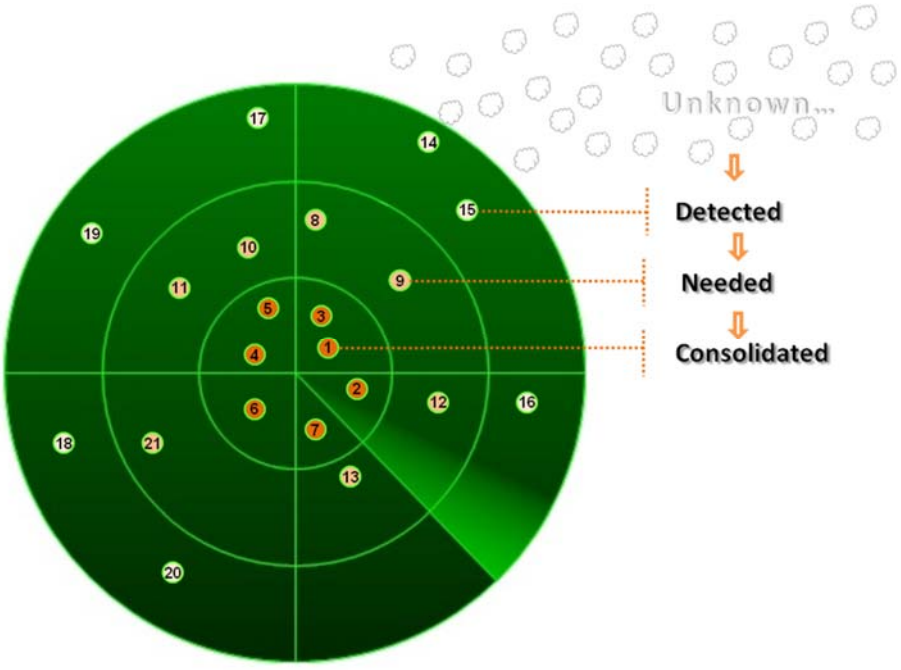

Figure A.1 – The Knowledge Radar

1. *Consolidated*: knowledge accumulated in a consistent way during the analyzed period of study, i.e., topics considered as fundamental and certainly learned during the period of study;

2. *Needed:* sub-set of knowledge identified as important for the progress of research, but has not been fully learned. Generally, it includes topics that need to be studied more deeply in order to support the researcher's decision making process in the short term ("just in time");

3. *Detected:* knowledge identified, but whose use in research is still controversial. Then, it is knowledge that we are not yet sure whether it will be applied or not; and it is waiting for a possible demand (level ii) and ["just in case"] turn out to be studied deeply.

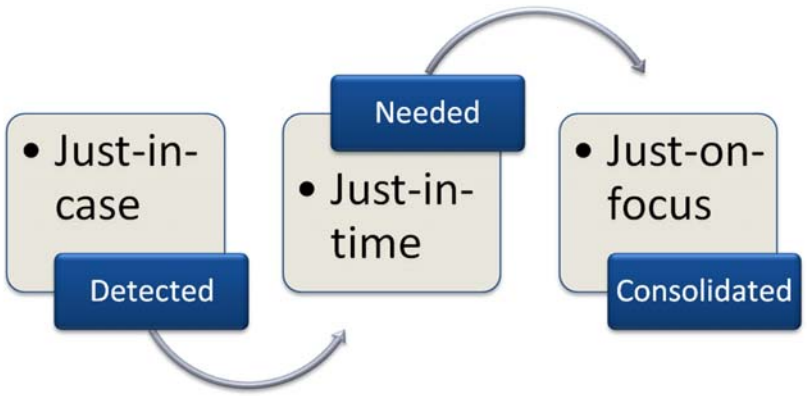

Figure A. 2 - KRADAR's levels of knowledge

Appendix B : The [YOU] Application

B.1 Sign In

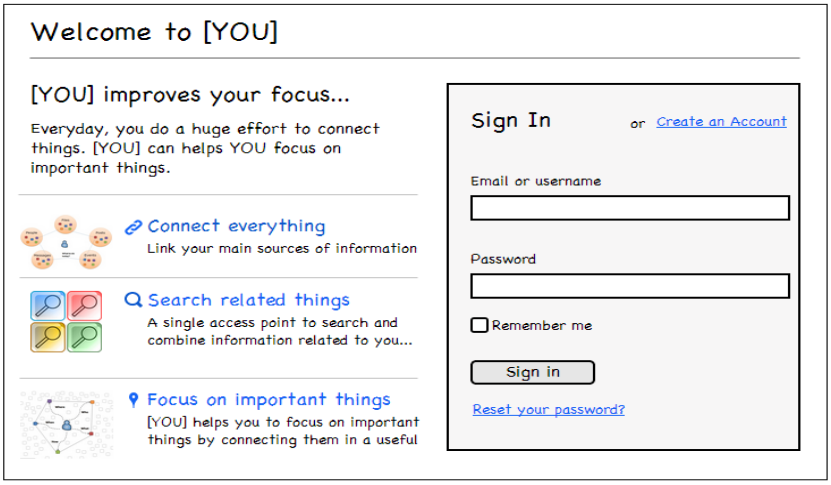

Figure B.1 - Welcome page

B.2 Connect

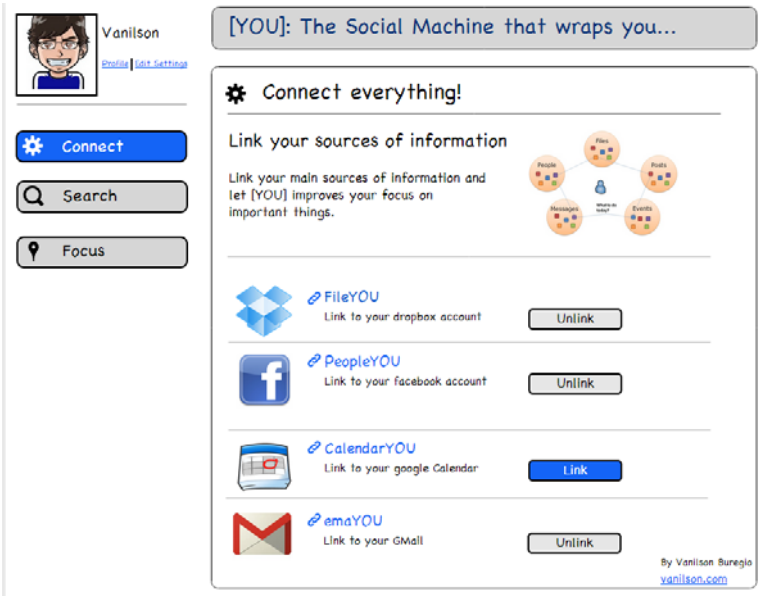

Figure B. 2 - Connect page

B.3 Search

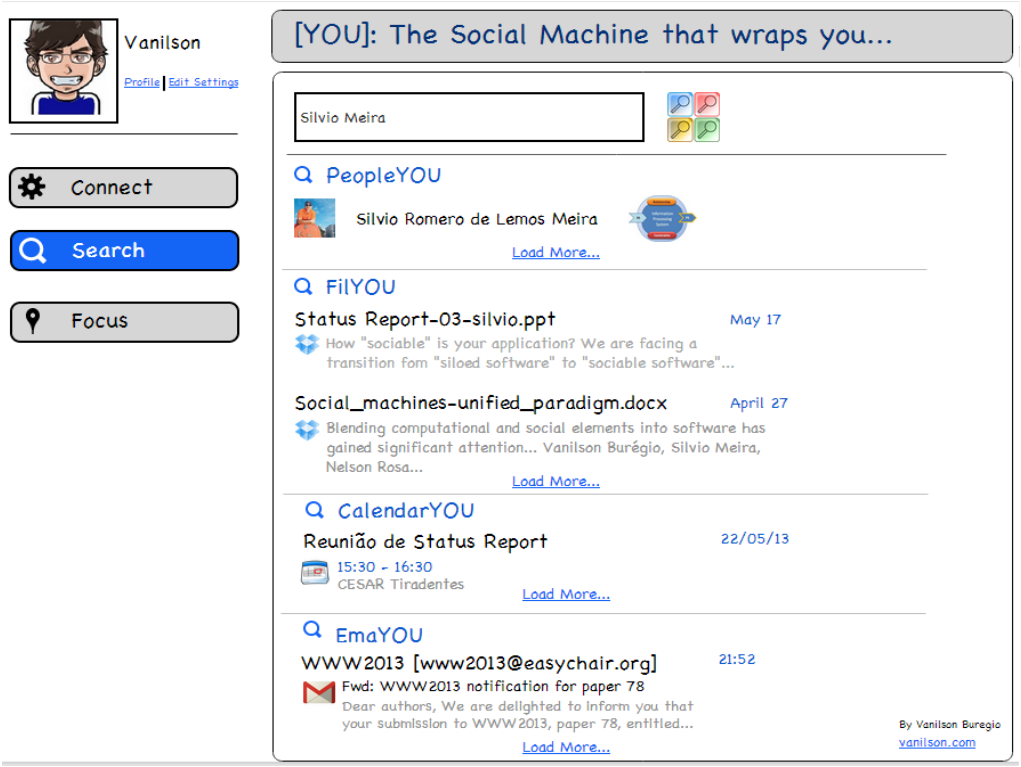

Figure B. 3 - Search page

B.4 Focus

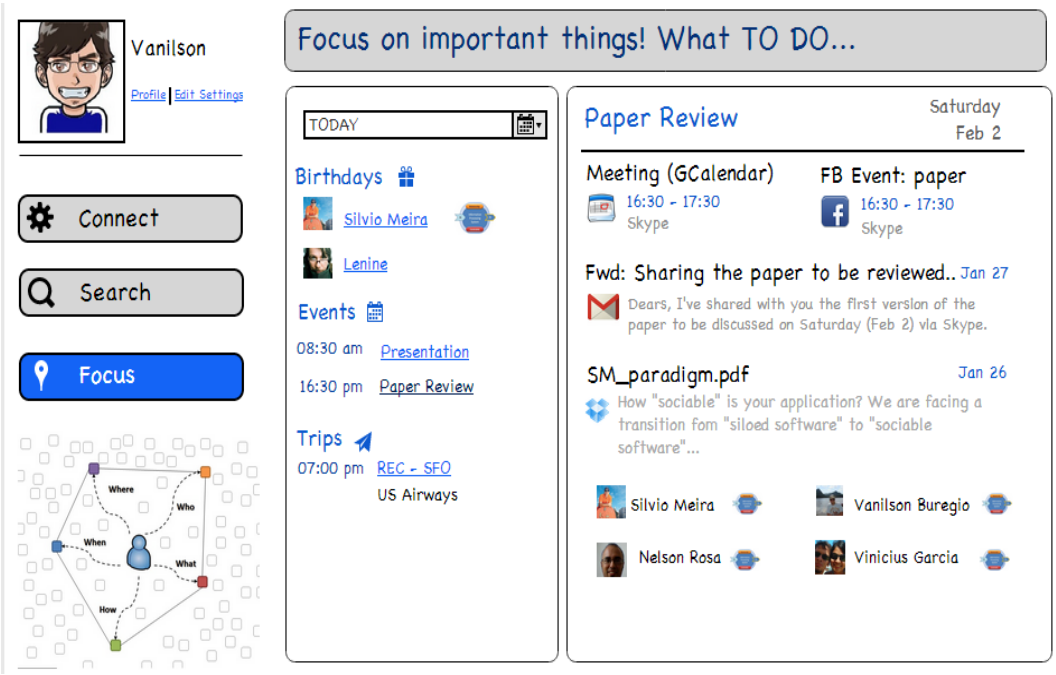

Figure B. 4 - Focus Page