SMADL: The Social Machines Architecture Description Language

Leandro Marques do Nascimento[1,2], Vinicius Cardoso Garcia[1],
Silvio R. L. Meira[1]

[1] Informatics Center - Federal University of Pernambuco (UFPE) ,
[2] Department of Informatics - Federal Rural University of Pernambuco (UFRPE)
{lmn2, vcg, srml}@cin.ufpe.br

Abstract. We are experiencing a high growth in the number of web applications being developed. This is happening mainly because the web is going into a new phase, called programmable web, where several web-based systems make their APIs publicly available. In order to deal with the complexity of this emerging web, we define a notion of social machine and envisage a language that can describe networks of such. To start with, social machines are defined as tuples of input, output, processes, constraints, states, requests and responses; apart from defining the machines themselves, the language defines a set of connectors and conditionals that can be used to describe the interactions between any number of machines in a multitude of ways, as a means to represent real machines interacting in the real web. This work presents a preliminary version of the Social Machine Architecture Description Language (SMADL).

1 Introduction

Software systems are built upon programming languages. A programming language is a notation for expressing computations (algorithms) in both machine and human readable form. Appropriate languages and tools may drastically reduce the cost of building new applications as well as maintaining existing ones [1].

In the context of programming languages, a Domain-Specific Language (DSL) is a language that provides constructs and notations tailored toward a particular application domain [2]. Usually, DSLs are small, more declarative than imperative, and more attractive than General-Purpose Languages (GPL) for their particular application domain.

However, in software engineering several different artifacts are developed besides code and one of the most important is the software architecture. Most developers agree that architecture is needed in some way, shape, or form, but, they can't agree on a definition, don't know how to manage it efficiently in nontrivial projects, and usually can't express a system's architectural abstractions precisely and concisely [3]. When asking a developer to describe a system's architecture Voelter [3] says "*I get responses that include specific technologies, buzzwords*

such as AJAX (asynchronous JavaScript and XML) or SOA (service-oriented architecture), or vague notions of "components" (such as publishing, catalog, or payment). Some have wallpaper-sized UML diagrams in which the meanings of the boxes and lines aren't clear."

These answers mention aspects that are actually related to a system's architecture, but none of them represent an unambiguous and/or "formal" description of a system's core abstractions. Indeed, it is not surprising because, although there are languages that directly express software architectures, they are not quite common among software developers.

In order to better define software architectures, it is worthy using DSLs and taking advantage of their expressiveness in a limited domain. Our proposal relies on top of an Architecture Description Language (ADL) for describing web based software systems in terms of Social Machines, a new concept developed by our research group which tries to increase the abstraction level for comprehending the web. Next, we present the context and the details about Social Machines.

2 An Emerging Web of Social Machines

The traditional concept of software has been changing during the last decades. Since the first definition of a computing machine described by Turing in [4], software started to become part of our lives and has been turned pervasive and ubiquitous with the introduction of personal computers, the internet, smartphones and recently the internet of things. In fact, one can say that software and the internet changed the way we communicate, the way business is done and the way software is developed, deployed and used. Nowadays, computing means connecting [5] and sometimes it is said that developing software is the same as connecting services [6], since there are several up and running software services available.

Recently, we all can clearly see that a new phase is emerging, the web "3.0", the web as a programming platform, the network as an infrastructure for innovation, on top of which all and sundry can start developing, deploying and providing information services using the computing, communication and control infrastructures in a way fairly similar to utilities such as electricity.

An overview of this Web 3.0 scenario can be seen in the *ProgrammableWeb* website[1]. It gathers around 6500 publicly available APIs and more than 6700 mashups using them (last visit in July 2012). Although there have been many studies about the future of the internet and concepts such as web 3.0, programmable web [7, 8], linked data [9] and semantic web [10, 11], the segmentation of data and the issues regarding the communication among systems obfuscates the interpretation of this future. Unstructured data, unreliable parts and non-scalable protocols are all native characteristics of the internet that needs a unifying view and explanations in order to be developed, deployed and used in a more efficient and effective way.

[1] www.programmableweb.com

Furthermore, the Web concepts, as we know, are recent enough to represent many serious difficulties while understanding their basic elements and how they can be efficiently combined to develop real, practical systems in either personal, social or enterprise contexts. Therefore, we developed a new concept called Social Machine (SM), in order to provide a common and coherent conceptual basis for understanding this still immature, upcoming and possibly highly innovative phase of software development. SM concept was firstly conceived in [12] and later demonstrated with a case study in [13].

So, we define a SM as a tuple, as following:

```
SM = <Rel, WI, Req, Resp, S, Const, I, P, O>
```

In general, a SM represents a connectable and programmable entity containing an internal *processing unit* **(P)** and a *wrapper interface* **(WI)** that waits for *requests* **(Req)** from and replies [with *responses* **(Resp)**] to other social machines. Its processing unit receives *inputs* **(I)**, produces *outputs* **(O)** and has *states* **(S)**; and its connections define intermittent or permanent *relationships* **(Rel)** with other SMs. These relationships are connections established under specific sets of *constraints* **(Const)**. Our goal with this concept of a Social Machine is not to formally describe software services as can be seen in [14], but instead we want to describe the programmable web in a higher level of abstraction, thus increasing the power of new programming structures or paradigms dedicated to this context. Figure 1 illustrates a basic representation of a Social Machine.

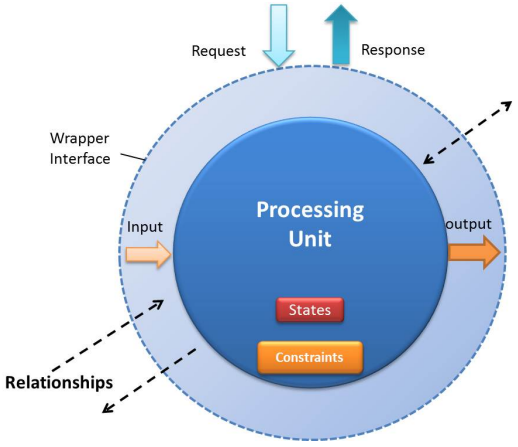

Fig. 1. A graphical representation of a Social Machine.

The idea behind Social Machines is to take advantage of the networked environment they are in to make it easier to combine and reuse exiting services from different SMs and use them to implement new ones. Hence, we can highlight some of its main characteristics, as following: *Sociability, Compositionality,*

Platform and Implementation independency, Self-awareness, Discoverability and last, but not least, *Programmability*.

There may be different types of social machines, but one way to classify them is through the simple taxonomy shown in Figure 2, based on the types of interactions they have with each other, as follows:

- **Isolated** - Social Machines that have no interaction with other Social Machines;
- **Provider** - Social Machines that provide services for other Social Machines to consume;
- **Consumer** - Social Machines that consume services that other Social Machines provide;
- **Prosumer** - Social Machines that both provide and consume services.

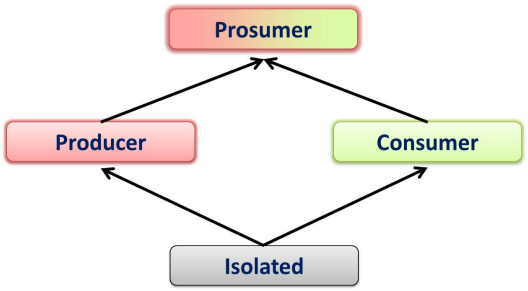

Fig. 2. Social Machines as a partial order diagram.

In this work, we envisage an Architecture Description Language that can describe networks of SMs. Apart from defining the machines themselves, the ADL defines a set of connectors and conditionals that can be used to describe the interactions between any number of machines in a multitude of ways, as a means to represent real machines interacting in the real web. Details are presented next.

3 The Social Machines Architecture Description Language - SMADL

This work is an attempt to answer the following research questions: "*Is it possible to integrate diverse web applications using a standard architecture description language?*". In order to answer it, this work purposes a new ADL for defining social machines: SMADL.

Social Machines can be connected (or establish a relationship) in basically two phases: in the first phase, the SMs must find each other, and a there must be a SM registry service much likely Internet DNS; in the second phase the SMs actually connect to each other and exchange information for a limited period

of time. The SM registry service is out of the scope of this proposal. We are assuming SMs can find each other without much effort.

In order to comprise these two phases, SMADL is composed by two mini-languages:

- **VCL** (*Visitor Card Language*): presents the externally visible properties of a SM, i.e., which requests and responses it accepts, which types of inputs/outputs it handles, if an internal state is maintained and/or how many requests it can handle per amount of time. A vCard is provided by the SM registry to the consumer SM, so it can decide if the relationship is interesting or not. Business issues are also present in a SM vCard, such as, billing information and service level agreements. Nowadays, popular web APIs do not make available such business information in a programmatic way. Usually, there are only few lines in reduced font size contracts mentioning that important information.
- **WIL** (*Wrapper Interface Language*): we are assuming every SM has a vCard with which the wrapper interface is fully complaint. This language is responsible for actually connecting SMs, establishing pre and post conditions, applying different connectors in a SM composition, and implementing business rules associated with a given set of SMs. Our proposal is to use WIL not as substitute for the currently available technologies. Instead it increases the level of abstraction of these technologies, freeing the programmer to concentrate on business issues of the SM relationships.

To understand better how these mini-languages are used, Figure 3 shows the steps for establishing a relationship between two SMs, as following:

1. Initially, the requester SM, in this case represented by Evernote (upper left icon), searches for some SM registered as a micro blog, in our case Twitter (upper right icon). Note that Evernote presents its vCard to DNS while searching for a service, once the responder may or may not accept connections with that specific SM.
2. The SM DNS finds Twitter and requests its vCard.
3. Twitter responds with its vCard, accepting the relationship.
4. The SM DNS replies back to Evernote with the Twitter vCard, which includes its address.
5. Using Twitter vCard and knowing its wrapper interface, Evernote establishes a relationship with Twitter, following all conditions imposed.

There are several popular technologies for integrating web based or service oriented systems, such, REST [15] and OSGi [16]. The current version of SMADL generates code for REST based apps, as it is becoming the most popular on the web, adopted by big players such as Facebook and Google. According to ProgrammableWeb site 4300 out of approximately 6500 APIs uses REST as base technology.

SMADL is being developed on Xtext language workbench [17]. As this is a work in progress, we are preliminarily evaluating alpha versions of the language and planning an experiment using the approach proposed by [18].

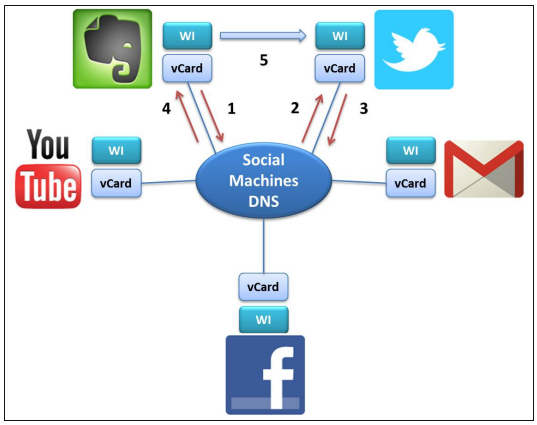

Fig. 3. Steps for establishing a relationship between two example Social Machines.

4 Related Work

We performed a systematic mapping study [19] for better understanding the DSL/ADL research field as shown in [20]. Initially, 4450 studies were identified, and, after filtering, 1440 primary studies were selected and categorized. Among all those primary studies, different methods/techniques for handling DSLs (creating, evolving, maintaining, testing) could be listed and several DSLs applied to several different domains could be identified. The domain where DSLs are most frequently applied is the Web domain. Other domains such as embedded systems, data intensive apps, and control systems where quite common too.

In our study we could enumerate 30 publications directly related to ADL. Amongst them, only two of them mention the Web domain, both from 2010. In the first one [21], the authors propose to formalize the architectural model using domain-specific language, an ADL which supports the description of dynamic, adaptive and evolvable architectures, such as SOA itself. Their ADL allows the definition of executable ver-sions of the architecture. The second one is [**?**] which presents a framework for the implementation of best practices concerning the design of the software architecture. The authors present an implementation of the framework in the Eclipse platform and an ADL dedicated to Web applications.

In addition, practical examples, such as *Yahoo! Pipes*[2] and *IfThisThenThat*[3] can be seen as related work. The former uses a graphical tool for customizing data flows from different sources. The latter allows end users to program the web based on pre-defined events fired by a set of channels, for example, if someone tags you on a given social network (channel 1), then save this photo in the person's virtual drive (channel 2). The user can choose among different events from different channels, which are, in practice, websites that make available their APIs. Our work is an attempt to be a completely different way to program

[2] http://pipes.yahoo.com
[3] http://ifttt.com/

the Web, not based on pre-defined parameters. The idea behind SMADL is to actually define every public API in the Web and the relationships among them. This way, composition possibilities for several SMs can be infinite.

As can be seen, this is a relatively new research field and we believe we can make a considerable contribution by establishing the concept of a Social Machine and developing an ADL for supporting it.

5 Concluding Remarks and Future Work

This work presents SMADL "The Social Machines Architecture Description Language" as a possible solution for modeling web-based software systems.

In general, a Social Machine (SM) represents a connectable and programmable entity containing an internal *processing unit* **(P)** and a *wrapper interface* **(WI)** that waits for *requests* **(Req)** from and replies [with *responses* **(Resp)**] to other social machines. Its processing unit receives *inputs* **(I)**, produces *outputs* **(O)** and has *states* **(S)**; and its connections define intermittent or permanent *relationships* **(Rel)** with other SMs. These relationships are connections established under specific sets of *constraints* **(Const)**.

Our main goal is to use SMADL to describe SM relationships and then we can have one unique way to program the web, independently of what technology/platform is being used. The current version (alpha) of SMADL generates code for REST based apps, as it is becoming the most popular on the web, adopted by big players such as Facebook and Google. Nowadays, we are working on several sample apps which have their architecture written in SMADL. These apps are basically consumer SMs, it means they do not make available public features, but only consumes other public APIs. These apps are going to be distributed as open source.

Our next steps include writing fully prosumer social machines, i.e. applications that connects to others, process their data, and someway make this data available for others to consume. At this phase, we are planning to perform an experiment, following the methodology described in [18].

Acknowledgments. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037-1.03/08 and APQ-1044-1.03/10 and Brazilian Agency (CNPq processes number 475743/2007-5 and 140060/2008-1).

References

1. Pressman, R.S.: Software engineering: a practitioner's approach (2nd ed.). McGraw-Hill, Inc., New York, NY, USA (1986)
2. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4) (December 2005) 316–344

3. Völter, M.: Architecture as language. IEEE Software **27**(2) (2010) 56–64
4. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London Mathematical Society **42** (1936) 230–265
5. Roush, W.: Social Machines. Technology Review (2006) 1–18
6. Turner, M., Budgen, D., Brereton, P.: Turning software into a service. Computer **36**(10) (October 2003) 38–44
7. Yu, S., Woodard, C.J.: Service-oriented computing — icsoc 2008 workshops. Springer-Verlag, Berlin, Heidelberg (2009) 136–147
8. Hwang, J., Altmann, J., Kim, K.: The structural evolution of the web 2.0 service network. Online Information Review **33**(6) (2009) 1040–1057
9. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. **5**(3) (2009) 1–22
10. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5) (2001) 34–43
11. Hitzler, P., Krtzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. 1st edn. Chapman & Hall/CRC (2009)
12. Meira, S.R.L., Burégio, V.A., Nascimento, L.M., de Figueiredo, E.G.M., Neto, M., Encarnação, B.P., Garcia, V.C.: The Emerging Web of Social Machines. CoRR **abs/1010.3** (2010)
13. Meira, S.R.L., Buregio, V.A.A., Nascimento, L.M., Figueiredo, E., Neto, M., Encarnacao, B., Garcia, V.C.: The Emerging Web of Social Machines. In: 2011 IEEE 35th Annual Computer Software and Applications Conference, IEEE (July 2011) 26–27
14. Broy, M., Krüger, I.H., Meisinger, M.: A formal model of services. ACM Trans. Softw. Eng. Methodol. **16**(1) (February 2007)
15. Richardson, L., Ruby, S.: Restful web services. First edn. O'Reilly (2007)
16. Hall, R.S., Pauls, K., McCulloch, S., Savage, D.: OSGi in Action: Creating Modular Applications in Java. Volume 188. Manning (2010)
17. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. SPLASH '10, New York, NY, USA, ACM (2010) 307–309
18. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Springer (2001)
19. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering. EASE'08, Swinton, UK, UK, British Computer Society (2008) 68–77
20. Nascimento, L.M., Viana, D.L., da Mota Silveira Neto, P.A., Souto, S.F., Martins, D.A.O., Garcia, V.C., Meira, S.R.L.M.: Domain-Specific Languages - A Systematic Mapping Study. In: Proceeedings of 7th International Conference on Software Engineering Advances (ICSEA). (2012)
21. López-Sanz, M., Cuesta, C.E., Marcos, E.: Formalizing high-level service-oriented architectural models using a dynamic adl. In: Proceedings of the 2010 international conference on On the move to meaningful internet systems. OTM'10, Berlin, Heidelberg, Springer-Verlag (2010) 57–66