



HAL
open science

SOCIOPATH: In Whom You Trust?

Nagham Alhadad, Philippe Lamarre, Yann Busnel, Patricia Serrano-Alvarado,
Marco Biazzini, Christophe Sibertin-Blanc

► **To cite this version:**

Nagham Alhadad, Philippe Lamarre, Yann Busnel, Patricia Serrano-Alvarado, Marco Biazzini, et al..
SOCIOPATH: In Whom You Trust?. 2011. hal-00608435v1

HAL Id: hal-00608435

<https://hal.science/hal-00608435v1>

Submitted on 13 Jul 2011 (v1), last revised 9 Sep 2011 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SOCIOPATH: In Whom You Trust?

Naghm Alhadad Philippe Lamarre Yann Busnel
Patricia Serrano-Alvarado Marco Biazzini Christophe Sibertin-Blanc

{Name.Lastname}@univ-nantes.fr
{Christophe.Sibertin-Blanc}@univ-tlse1.fr

LINA / Université of Nantes
2, rue de la Houssinière
BP 92208 – 44322 Nantes, France

First Edition – July 13, 2011

Abstract

Distributed systems are getting more and more numerous, complex and used in a wide variety of applications. New solutions and new architectures arise (*e.g.*, clouds) that support new functionalities (*e.g.*, social networks) and pile up several software layers. This evolution implies new non negligible *dependences* increasing in the number of actors involved in the system (*e.g.*, providers and users). Some undesirable dependences could be hidden by this layer stacking, implying a reduced transparency for users and a misunderstanding of her actual autonomy. Given that any software is directly dependent of the underlying layers, if any of these layers misbehaves, the given software may be unable to provide promised services. We argue that users should be aware of the potential risks resulting from their dependences. To be able to deduce those dependences, one should know the way the system works (architecture, involved resources, providers, participants, *etc.*). This would help to deduce the potential trust a user could or should have toward the system. We consider this of utmost importance, as professional efficiency and personal privacy could be compromised if untrusted actors control the access to resources. This work proposes SOCIOPATH, a generic meta-model that allows to expose hidden or implied relationships among participants in the digital world, which also introduce dependences at the social level. The notions presented in this approach are basics of many fields, like security, privacy, trust, sociology, economy and so forth. SOCIOPATH can be used in the evaluation process of a system as well as in its upstream design.

1 Problem definition

A large number of distributed systems arise nowadays that are more and more complex and used for a tremendous variety of applications. Actually, solutions proposed to users evolve toward new functionalities (*e.g.*, social networks), new architectures that support them (*e.g.*, clouds) and pile up several software layers. This evolution implies new non negligible *dependences* among providers and actors in the system .

When users need to choose a system, they are overwhelmed by the plethora of free and lucrative available options. To make a choice, they *evaluate* systems according to its perceived capability to satisfy their needs and the time they have to make their choice. Traditionally, the evaluation covers functional, technical and economical aspects. From the functional point of view, users analyze the quality of provided services and the user-friendliness of a system. Technical aspects orient the evaluation to operational criteria (*e.g.*, response time, reliability, availability, safety, security, *etc.*) but also to deployment and maintenance requirements. Moreover, economical aspects are considered, like the necessary investments to start up the system and to assure a durable using.

From one user to another, those evaluation criteria have different weights and consequences and for the same applicative needs, different systems may be chosen. For instance, consider two users who need a software to edit and share documents. One user, working in a non-profit organization, may choose a freeware and disregard performance characteristics, while another, which considers himself unskilled, may choose an easy-to-use software, without installation and maintenance efforts. Thus, the first one may decide to use OpenOffice¹ and the second one GoogleDocs². Such a decision has several implications. For instance, besides having a legal and running version of an operating system, the unskilled user is obliged to have a stable and reliable Internet connection. Most of the time, functionalities and performance of any software are directly dependent on underlying layers. If one of these layers misbehaves, the given software may be unable to provide the promised services.

In general, as users, we assume software developers and device manufacturers are competent and have the best intentions. Using a system entails the drawing of relationships of *trust* between users and providers, and users are not always aware of those implicit relationships. A kind of “trust among participants” is *de facto* constructed, based on a mix of more or less conscious factors such as the quality of their exchanges [1, 2, 3, 4, 5] or the trust toward resources (data, programs, communications, *etc.*) and providers [6, 7, 8, 9]. In this work we consider the trust from another point of view. We argue that users should be aware of the potential risks resulting from their dependences on the system, either by means of public information or by their own deductions. To this end, the way the system works (underlying architecture, involved resources, providers, participants, *etc.*) must be explicit. This would help the users deduce the potential trust they should have toward the system. We consider this very important, because professional efficiency and personal privacy may be compromised if untrusted actors *control* the access to others’ resources in a system.

SOCIOPATH, the meta-model we propose in this paper, allows to identify, in the digital world, the hardware and software components of a system and the dependence relationships between them, together with, in the social world, the actors that are involved in their availability, proper operating or use. The idea is that deduced relationships underline the potential repercussions of the trust users have toward the system in terms of security, privacy, social relationships, economy, *etc.* Thus, when assessing the suitability of a system, we should take into account functional, technical, economical but also dependence-related aspects.

This paper is organized as follows. Section 2 introduces SOCIOPATH. Deduction rules and basic definitions are respectively presented in Sections 3 and 4. Section 5 illustrates different use cases of SOCIOPATH. Finally, Section 6 presents a brief overview of related works and Section 7 gives some conclusions and points out some ongoing works.

2 SOCIOPATH meta-model

SOCIOPATH, the meta-model proposed in this work, aims at providing a formalism to help the user answer questions related to her relations in the sociological world (dependences on persons) and in the digital world (dependences on resources). In this section, we are interested in what the users might be able to do, rather than on what they are permitted to do. Thus, imposed access and/or control restrictions are not considered here. Some interesting questions we try to answer are:

1. *On whom the user depends to access her data?*

If a user stores her data instances on a server, she depends at least on the provider of the server and on the person who owns the server.

2. *On which applications the user depends to access her data?*

When data instances are stored outside the user’s computer, she may access her data through FTP clients, web browsers, *etc.*

¹<http://www.openoffice.org/>

²<https://docs.google.com/>

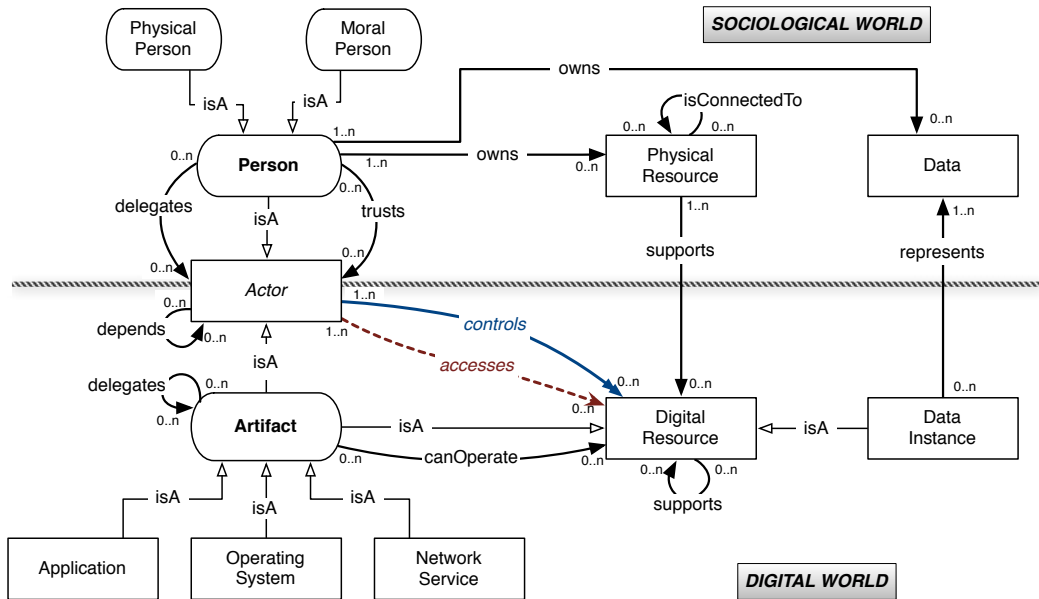


Figure 1: The Meta Model

3. *Who can access user's data?*

When a user stores her data instances on a server, the administrator of the server and the service provider can access her data.

4. *Through which resources somebody else can access user's data?*

Some of the resources a user depends on to access her data may be used by other users, to maliciously get the same data.

5. *What are the necessary coalition between persons to access a particular data instance?*

Those persons on whom a user depends to access her data might be able to access her data, if they colluded.

Figure 1 presents a graphical representation of SOCIOPATH. This meta-model may be seen as a tool that helps representing the reality of two worlds, which come together and interact between each other: the *sociological world* and the *digital world*.

2.1 The sociological world

The sociological world describes physical and moral persons (enterprises, companies, etc.), resources, data and the relations among them.

- *Person*: either a Physical or a Moral Person;
- *Data*: an abstract notion that does not necessarily imply a physical instance (e.g., address, age, software, etc.);
- *Physical Resource*: hardware device (e.g., PC, USB device, etc.).

2.2 The digital world

The digital world has nodes characterizing digital resources, artifacts, data instances, operating systems, networks services and applications.

- *Data Instance*: a digital representation of Data. It may be semantically equivalent to data that exists in the sociological world. For instance, a person has an address (Data) in the sociological world. Whenever she writes it in a file, she creates a semantically equivalent digital instance of her address in the digital world (Data Instance).
- *Artifact*: it may be an Application, an Operating System or a Network Service. We mean all of them to be “running software”, thus considering them only in that they are being executed. By *Application*, we mean a whole running entity. It may be a single process or a group of processes that may even be distributed in different locations, yet defining a single logically coherent entity;
- *Digital Resource*: an Artifact or a Data Instance.

Between the two worlds there is the node Actor;

- *Actor*: a Person or an Artifact.

2.3 The relations in SOCIOPATH

Several relations are drawn in SOCIOPATH. We briefly describe them as follows.

- *owns*: it means ownership. This relation exists only in the sociological world;
- *isConnectedTo*: it means that two nodes are physically connected. This relation exists only in the sociological world and it is intrinsically symmetrical;
- *trusts*: relations of trust exist among persons and can be drawn from persons to artifacts. Assessing if and how a given architecture “deserves” the users trust towards the system is one of the future goal of our work.
- *delegates*: a Person can delegate another Actor to perform some kind of access or control on a resource. The same concept of delegation can be implemented among artifacts (*e.g.*, in large databases distributed transactions are often performed by means of chains of delegations).
- *canOperate*: it means that the artifact is able to process, communicate, interact with the target resource. This ability may be given as a part of the artifact specification (*e.g.*, MSWord canOperate the document toto.doc) or deduced by some contingent property of the system (*e.g.*, an operating system only canOperate those files that are stored in a mounted partition).
- *accesses*: An Actor can access a Digital Resource (*e.g.*, the operating system accesses the applications installed on it, or a person who owns a PC that supports an operating system accesses this operating system). The access relations we consider are: read, write, execute.
- *controls*: An Actor can *control* a digital resource. There may be different kinds of control relations. For instance, a moral person, who provides a resource to other persons, controls the functionality of this resource. The persons who use this resource have some control on it as well. Each of these actors controls the resource in a different way³;
- *depends*: An Actor may *depend* on another Actor to perform an activity (*e.g.*, a person depends on Google when she accesses her data instances by using the GoogleDocs application);

³In this work, we do not focus on all the different types of control; we rather consider the control notion in a general meaning. Part of our future work will be devoted to specify the types of control

- *supports*: it means that the target node could never exist without the source node. We may say that the latter allows the former to exist (e.g., a running operating system exists only if it is hosted on a given hardware, an application is supported by the operating systems that hosts it or the code of an application supports this application);
- *represents*: it is a relation that exists between data in the sociological world and their instances on the digital world (e.g., the source code of the Windows operating system is a representation in the digital world of the data known as “Microsoft Windows ©” in the sociological world);

By applying SOCIOPATH, it is possible to make non-trivial deductions about relations among nodes. For instance, an actor may be able to access digital resources supported by different physical resources connected to each other (e.g., a user may access processes running on different hosts).

Every person owns data in the sociological world. These data have a concrete existence in digital world if they are represented by data instances, supported by physical resources (e.g., CD, DVD, PC, etc.). As an actor in the digital world, a person can access and control data instances representing her (and others’) data. This may possibly be done through chains of delegations, or accessing different resources, which imply some dependence on other persons.

In this work, we are particularly interested in formalizing the relations in the digital world, in order to derive the dependences among persons in the sociological world.

2.4 Introducing example

In order to illustrate our meta-model, we model a simple and basic example, presented in Figure 2:

In the sociological world, there is a user John, who owns some Data and a PC. There are also moral persons as Microsoft (provider of Windows, MSWord and MSEXcel), Apple (provider of MacOS) and Oracle (provider of OOWrite).

In the digital world, two operating systems exist on John’s PC: Windows and MacOS. On Windows, there are two applications available: MSWord and MSEXcel. On MacOS, there are OOWrite and Pages. John’s Data are represented in the digital world by the document toto.doc.

This example will be used in the next section to illustrate some deduction rules.

3 Language and deduction rules

We use a language based on First Order Logic (FOL) to describe the model of a specific architecture. The nodes in the meta-model are described using unary predicates, and the edges between nodes are described by binary predicates. Moreover, we propose some rules, based on this language, that formalize the relations in the architecture.

3.1 Deduction rules

In this section, we enumerate some deduction rules. These rules are not exhaustive and by no mean we pretend them to capture the whole complexity of a system. They capture several aspects of a simplified vision of the systems, that serves the purpose of building an understandable and expressive model. In the predicates, we use the same names as in the meta-model, except for the long names, for which we make simpler synonyms. For instance, the predicate *phyresource*(\bar{R}) is used for physical resource. Table 1 summarizes all the notations used in the following.

- An artifact accesses a digital resource if the artifact can operate the digital resource and the artifact and the digital resource are supported by the same physical resource or supported by different physical resources

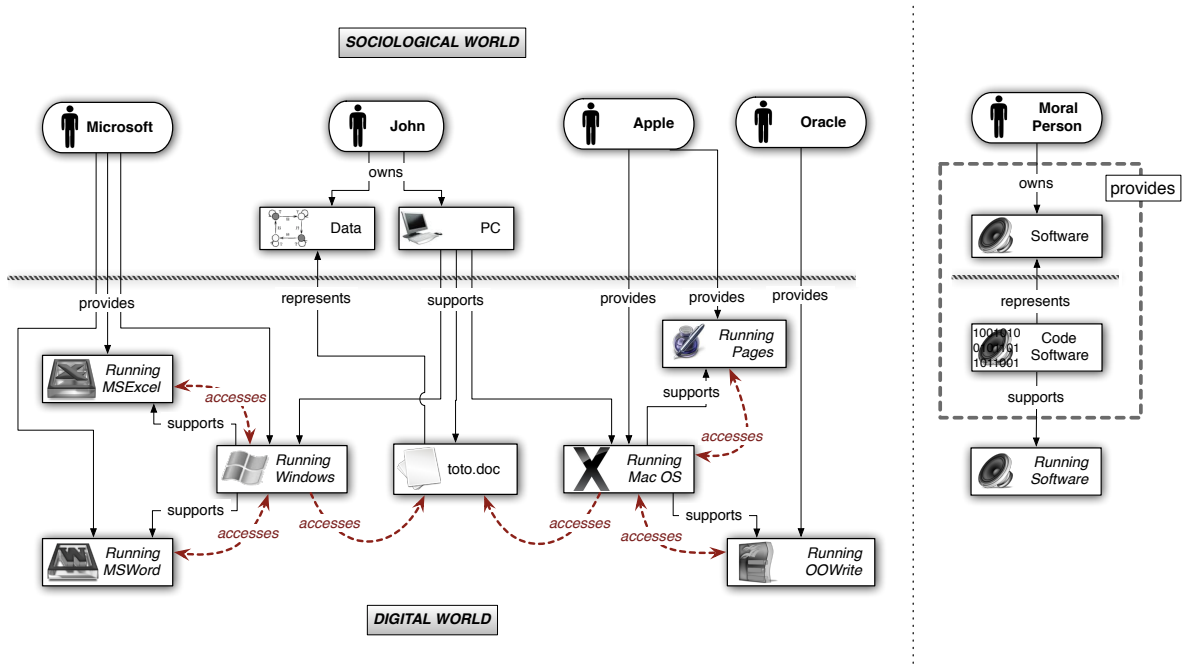


Figure 2: Introducing example: a document accessible by 2 different operating systems

connected to each other.

$$\forall F \in \mathbb{F}, \forall R \in \mathbb{R}, \forall \bar{R}1, \bar{R}2 \in \bar{\mathbb{R}} : \bigwedge \begin{cases} \text{canOperate}(F, R) \\ \text{supports}(\bar{R}1, F) \\ ((\text{supports}(\bar{R}1, R)) \\ \vee (\text{supports}(\bar{R}2, R) \wedge \text{isConnectedTo}(\bar{R}1, \bar{R}2))) \end{cases} \Rightarrow \text{accesses}(F, R) \quad (1)$$

e.g., Windows accesses MSWord:

$$\text{canOperate}(\text{Windows}, \text{MSWord}) \wedge \text{supports}(\text{PC}, \text{Windows}) \wedge \text{supports}(\text{PC}, \text{MSWord}) \Rightarrow \text{accesses}(\text{Windows}, \text{MSWord}).$$

- If an operating system supports an artifact and can operate this artifact, it controls this artifact.

$$\forall F \in \mathbb{F}, \forall OS \in \mathbb{O} : \text{supports}(OS, F) \wedge \text{canOperate}(OS, F) \Rightarrow \text{controls}(OS, F) \quad (2)$$

e.g., Windows controls MSWord, Windows controls MSExcel, MacOS controls OOWrite, MacOS controls Pages:

$$\begin{aligned} \text{supports}(\text{Windows}, \text{MSWord}) \wedge \text{canOperate}(\text{Windows}, \text{MSWord}) &\Rightarrow \text{controls}(\text{Windows}, \text{MSWord}); \\ \text{supports}(\text{Windows}, \text{MSExcel}) \wedge \text{canOperate}(\text{Windows}, \text{MSExcel}) &\Rightarrow \text{controls}(\text{Windows}, \text{MSExcel}); \\ \text{supports}(\text{MacOS}, \text{OOWrite}) \wedge \text{canOperate}(\text{MacOS}, \text{OOWrite}) &\Rightarrow \text{controls}(\text{MacOS}, \text{OOWrite}); \\ \text{supports}(\text{MacOS}, \text{Pages}) \wedge \text{canOperate}(\text{MacOS}, \text{Pages}) &\Rightarrow \text{controls}(\text{MacOS}, \text{Pages}). \end{aligned}$$

- A person, who owns a physical resource that supports an operating system, accesses this operating system.

$$\forall P \in \mathbb{P}, \forall \bar{R} \in \bar{\mathbb{R}}, \forall OS \in \mathbb{O} : \text{owns}(P, \bar{R}) \wedge \text{supports}(\bar{R}, OS) \Rightarrow \text{accesses}(P, OS) \quad (3)$$

e.g., John accesses Windows:

$$\text{owns}(\text{John}, \text{PC}) \wedge \text{supports}(\text{PC}, \text{Windows}) \Rightarrow \text{accesses}(\text{John}, \text{Windows}).$$

Basic type of instance	The set of all instances		A subset of instances		One instance	
	<i>Notation</i>	<i>Remark</i>	<i>Notation</i>	<i>Remark</i>	<i>Notation</i>	<i>Remark</i>
Person	\mathbb{P}	$\{P : person(P)\}$	\mathcal{P}	$\mathcal{P} \subset \mathbb{P}$	P	$P \in \mathbb{P}$
Actors	\mathbb{A}	$\{A : actor(A)\}$	\mathcal{A}	$\mathcal{A} \subset \mathbb{A}$	A	$A \in \mathbb{A}$
Artifact	\mathbb{F}	$\{F : artifact(F)\}$	\mathcal{F}	$\mathcal{F} \subset \mathbb{F}$	F	$F \in \mathbb{F}$
Digital resource	\mathbb{R}	$\{R : resource(R)\}$	\mathcal{R}	$\mathcal{R} \subset \mathbb{R}$	R	$R \in \mathbb{R}$
Physical resource	$\overline{\mathbb{R}}$	$\{\overline{R} : phyresource(\overline{R})\}$	$\overline{\mathcal{R}}$	$\overline{\mathcal{R}} \subset \overline{\mathbb{R}}$	\overline{R}	$\overline{R} \in \overline{\mathbb{R}}$
Data	\mathbb{D}	$\{D : data(D)\}$	\mathcal{D}	$\mathcal{D} \subset \mathbb{D}$	D	$D \in \mathbb{D}$
Data instance	$\overline{\mathbb{D}}$	$\{\overline{D} : dataInstance(\overline{D})\}$	$\overline{\mathcal{D}}$	$\overline{\mathcal{D}} \subset \overline{\mathbb{D}}$	\overline{D}	$\overline{D} \in \overline{\mathbb{D}}$
Operating System	\mathbb{O}	$\{OS : operatingSystem(OS)\}$	\mathcal{O}	$\mathcal{O} \subset \mathbb{O}$	OS	$OS \in \mathbb{O}$
Path	Γ	$\{\sigma : path(\sigma)\}$	Υ	$\Upsilon \subset \Gamma$	σ	$\sigma \in \Gamma$
Architecture	Λ	—	—	—	α	$\alpha \in \Lambda$
Criteria	\mathbb{C}	—	—	—	\mathcal{C}	$\mathcal{C} \in \mathbb{C}$
Activity	\mathbb{W}	—	—	—	ω	$\omega \in \mathbb{W}$

Table 1: Glossary of notations

- A person, who owns a physical resource that supports an operating system, controls this operating system.

$$\forall P \in \mathbb{P}, \forall \overline{R} \in \overline{\mathbb{R}}, \forall OS \in \mathbb{O} : owns(P, \overline{R}) \wedge supports(\overline{R}, OS) \Rightarrow controls(P, OS) \quad (4)$$

e.g., John controls Windows:

$$owns(\text{John}, \text{PC}) \wedge supports(\text{PC}, \text{Windows}) \Rightarrow controls(\text{John}, \text{Windows}).$$

- A person, who owns data represented in the digital world by a data instance which supports an artifact, controls this artifact.

$$\exists P \in \mathbb{P}, \exists D \in \mathbb{D}, \exists \overline{D} \in \overline{\mathbb{D}}, \exists F \in \mathbb{F} : \bigwedge \begin{cases} owns(P, D) \\ represents(\overline{D}, D) \\ supports(\overline{D}, F) \end{cases} \Rightarrow controls(P, F) \quad (5)$$

e.g., Microsoft controls Windows:

$$owns(\text{Microsoft}, \text{Windows}) \wedge represents(\text{CodeWindows}, \text{Windows}) \wedge supports(\text{CodeWindows}, \text{Windows}) \Rightarrow controls(\text{Microsoft}, \text{Windows}).$$

- The relation ‘accesses’ is transitive.

$$\forall A \in \mathbb{A}, \forall F \in \mathbb{F}, \forall R \in \mathbb{R} : accesses(A, F) \wedge accesses(F, R) \Rightarrow accesses(A, R) \quad (6)$$

e.g., MSWord accesses Windows, and Windows accesses toto.doc, so accesses toto.doc:

$$accesses(\text{MSWord}, \text{Windows}) \wedge accesses(\text{Windows}, \text{toto.doc}) \Rightarrow accesses(\text{MSWord}, \text{toto.doc}).$$

- The relation ‘controls’ is transitive.

$$\forall A \in \mathbb{A}, \forall F_1, F_2 \in \mathbb{F} : controls(A, F_1) \wedge controls(F_1, F_2) \Rightarrow controls(A, F_2) \quad (7)$$

e.g., John controls windows and windows controls toto.doc so John controls toto.doc:

$$controls(\text{John}, \text{Windows}) \wedge controls(\text{Windows}, \text{toto.doc}) \Rightarrow controls(\text{John}, \text{toto.doc})$$

4 Basic Definitions

In this section, we detail the main concepts used in an architecture, to ease the analysis of a system. The examples in this section corresponds to the architecture presented in Section 2.4.

4.1 Path

Definition 1 (Access path).

A path σ is a list of actors and digital resources such that:

- $actor(\sigma[1]);$
- $\forall i \in [2 : |\sigma|], artifact(\sigma[i]) \wedge access(\sigma[i - 1], \sigma[i]).$
- $resource(\sigma[|\sigma|]);$

where $\sigma[i]$, respectively $|\sigma|$, denotes the i^{th} element of σ , respectively the length of σ .

The access paths in the architecture α is noted Υ_α or, simply Υ where there is no ambiguity for α .

Some examples of the access paths in the architecture presented in the Section 2.4:

```
[John, Windows, MSWord, Windows, toto.doc],
[Windows, toto.doc],
[Windows, MSeExcel, Windows, toto.doc],
[MacOS, OOWrite, MacOS, toto.doc],
[John, MacOS, OOWrite, MacOS, toto.doc],
[John, Windows, MSeExcel, Windows, MSWord, Windows, toto.doc],...
```

When a path σ_2 uses additional resources with respect to σ_1 , it is quite natural to say that σ_1 is longer than σ_2 . This leads to a notion of order.

Definition 2 (Order over paths).

Let α be an architecture. Let σ_1 and σ_2 be two paths within α : $\sigma_1, \sigma_2 \in \Upsilon_\alpha$
 $\sigma_1 \leq \sigma_2$ iff⁴ there exists a function f such that:

- $\begin{cases} f : \sigma_1 & \mapsto & \sigma_2 \\ \sigma_1[i] & \mapsto & \sigma_2[j] \end{cases}$
- $\forall i \in [1 : |\sigma_1|] : f(\sigma_1[i]) = \sigma_2[i]$
i.e., The function f maps elements of σ_1 to identical elements in σ_2 .
- $\forall i \in [1 : |\sigma_1| - 1] : \xi(f(\sigma_1[i]), \sigma_2) \leq \xi(f(\sigma_1[i + 1]), \sigma_2)$, where $\xi(x, \sigma)$ gives the rank of x in the path σ .
i.e., The elements are in the same order within the two paths

Notation: Consider the following notations:

$$\begin{aligned} \sigma_1 = \sigma_2 &\iff \sigma_1 \leq \sigma_2 \text{ and } \sigma_2 \leq \sigma_1 \\ \sigma_1 < \sigma_2 &\iff \sigma_1 \leq \sigma_2 \text{ and } \sigma_2 \not\leq \sigma_1 \end{aligned}$$

According to the aforementioned example (see Section 2.4), we can notice that $\sigma_1 < \sigma_2$:

$$\begin{cases} \sigma_1 = [\text{John}, \text{Windows}, \text{MSWord}, \text{Windows}, \text{toto.doc}], \\ \sigma_2 = [\text{John}, \text{Windows}, \text{MSWord}, \text{Windows}, \text{MSWord}, \text{Windows}, \text{toto.doc}]. \end{cases}$$

Definition 3 (minimal path).

Let α be an architecture. Let Υ be a set of paths within an architecture α .

A path σ of Υ is said to be minimal in Υ iff

$$\nexists \sigma' \in \Upsilon : \sigma' <_f \sigma.$$

Notation: The set of minimal paths is noted $\widehat{\Upsilon}$.

⁴iff denotes “if and only if”.

In our example, [John, Windows, toto.doc] is a minimal path from a user John to toto.doc.

Different paths may be used to reach the same goal, or for the same activity. We focus on a subset of paths so we propose to use some criteria.

Definition 4 (Criteria over paths).

Let Υ be a set of paths. A criterion is a function that:

$$\mathcal{C} : \Upsilon \mapsto \{true, false\}$$

Notation: $\Upsilon^{|\mathcal{C}} = \{\sigma : \sigma \in \Upsilon \wedge \mathcal{C}(\sigma) = true\}$ is the subset of Υ restricted to the correct paths with respect to some criteria \mathcal{C} .

Criteria may be used to select some paths, for instance:

- to select which paths enable a person to access a data instance.
- to select which paths enable a person to understand a data instance (e.g., to understand a .doc document, the person need to use an application that can understand this type of document, like MSWord. Thus the access path should contain this application).
- to select which paths make natural use of resources (e.g., when a user use a SVN service, the path should contain the SVN client and SVN server to be able to access the document, and the SVN client should precede the SVN server).

We can define *functionally equivalent* access paths, which are the set of paths that comply with the criteria: consider two paths σ_1 and σ_2 such that

$$\sigma_1[1] = \sigma_2[1] \quad \wedge \quad \sigma_1[|\sigma_1|] = \sigma_2[|\sigma_2|].$$

The set of equivalent paths for the actor A to access resource R is noted $\Upsilon^{A,R}$.

It is also worth considering the minimal functionally equivalent paths, noted $\widehat{\Upsilon^{A,R}}$. The formal definition of the criteria is out of scope of this paper. We can use for example regular expressions.

4.2 Activities

Intuitively, an activity is related to an actor that wants to do something concerning a resource (e.g., to access a directory, to copy a file, to edit a document, etc.). We assume that the criteria allow to obtain the complete set of paths corresponding to an activity.

Consider the following example of activity: “John accesses toto.doc”. The corresponding criterion \mathcal{C} should be:

$$\sigma_1[1] = \text{John} \quad \wedge \quad \sigma_1[|\sigma_1|] = \text{toto.doc.}$$

4.3 Dependence

Definition 5 (Artifact classification for an activity).

Let \mathcal{C} be the criteria characterizing the activity ω .

Usable artifact

The usable artifacts are the artifacts that appear in one of the access path.

$$F \text{ is a usable artifact for } \omega \text{ iff } \exists \sigma \in \Upsilon^{|\mathcal{C}}, F \in \sigma.$$

e.g., MSExcel is a usable artifact for the activity: “John accesses toto.doc”.

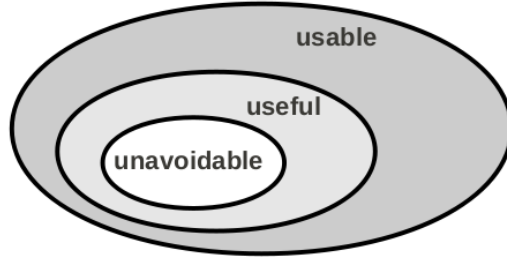


Figure 3: Artifact classification with respect to an activity.

Useful artifacts

The useful artifacts are artifacts that appear in one of the minimal path.

F is a useful artifact for ω iff $\exists \sigma \in \widehat{\Upsilon}^{\mathcal{C}}, F \in \sigma$.

e.g., *Windows is a useful artifact for the activity: “John accesses toto.doc”.*

Unavoidable artifact

the unavoidable artifacts are the artifacts that appear in every minimal path.

F is an unavoidable artifact for ω iff $\forall \sigma \in \widehat{\Upsilon}^{\mathcal{C}}, F \in \sigma$.

e.g., *If the user John has only the Windows operating system, then Windows is an unavoidable artifact for the ω : “John accesses toto.doc”.*

Figure 3, illustrates the classification of the artifacts with respect to an activity.

Definition 6 (Actor’s dependence on a group of artifacts for an activity).

Let ω be an activity for an actor A to a resource R , and \mathcal{C} be its criteria. Let \mathcal{F} be a set of artifacts.

A depends on \mathcal{F} iff $\forall \sigma \in \Upsilon^{\mathcal{C}}, \exists F \in \mathcal{F} : F \in \sigma$

For instance, John depends on the group [Windows, MacOS, MSWord, OOwrite, MSeExcel] for the activity: “John accesses toto.doc”.

Definition 7 (Actor’s strict-dependence on a group of artifacts for an activity).

Let ω be an activity for an actor A to a resource R , and \mathcal{C} be its criteria. Let \mathcal{F} be a set of artifacts.

A strictly-dependes on \mathcal{F} iff $\forall \mathcal{F}' \subsetneq \mathcal{F}, ((A \text{ depends on } \mathcal{F}) \wedge \neg(A \text{ depends on } \mathcal{F}'))$.

For instance, John strictly-dependes on the group [Windows, MacOS] for the activity: “John accesses toto.doc”.

Definition 8 (Degree of actor dependence on a group of artifacts for an activity).

Let ω be an activity for an actor A to a resource R , and \mathcal{C} be its criteria. Let \mathcal{F} be a set of artifacts.

The degree of dependence of A on \mathcal{F} , noted $d_{\mathcal{F}}^A$, is based on frequency of presence of \mathcal{F} elements in the paths related to the activity ω :

$$d_{\mathcal{F}}^A = \frac{|\{\sigma : \sigma \in \widehat{\Upsilon}^{\mathcal{C}} \wedge \exists F \in \mathcal{F}, F \in \sigma\}|}{|\widehat{\Upsilon}^{\mathcal{C}}|}$$

For instance, the degree of dependence of John on MSWord for the activity “John edits the document toto.doc” is equal to 1/3.

The degree of dependence of John on MacOS for the activity “John edits the document toto.doc” is equal to 2/3.

The degree of dependence of John on the group [MacOS, MSWord] for the activity “John edits the document toto.doc” is equal to 1.

The degree of dependence of John on the group [Pages, MSWord] for the activity “John edits the document toto.doc” is equal to 2/3.

The degree of dependence of John on the group [Windows, MSWord] for the activity “John edits the document toto.doc” is equal to 1/3.

The degree of dependence of John on the group [Pages, OOWrite] for the activity “John edits the document toto.doc” is equal to 2/3.

Definition 9 (A set of persons controls a set of resources).

Let \mathcal{R} be a set of resources, and \mathcal{P} be a set of persons,

$$\mathcal{P} \text{ controls } \mathcal{R} \text{ iff } \wedge \left\{ \begin{array}{l} \forall R \in \mathcal{R}, \exists P \in \mathcal{P} : \text{control}(P, R) \Rightarrow P \in \mathcal{P} \\ \forall P \in \mathcal{P}, \exists R \in \mathcal{R} : \text{control}(P, R). \end{array} \right.$$

For instance, the group [Microsoft, Apple, John] controls [Windows, MacOS].

Definition 10 (Actor’s dependence on a group of persons for an activity).

Let ω be an activity for an actor A to a resource R , and \mathcal{C} be its criteria. Let \mathcal{P} be a set of persons.

$$A \text{ depends on } \mathcal{P} \text{ for } \omega \text{ iff } \wedge \left\{ \begin{array}{l} \exists \mathcal{F} \subset \mathbb{F} : A \text{ depends on } \mathcal{F} \text{ for } \omega \\ \mathcal{P} \text{ control } \mathcal{F}. \end{array} \right.$$

For instance, John depends on [John, Microsoft, Apple] for the activity: “John accesses toto.doc”.

Definition 11 (Degree of actor’s dependence on a group of persons for an activity).

Let ω be an activity for an actor A to a resource R , and \mathcal{C} be its criteria. Let \mathcal{P} be a set of persons.

The degree of dependence of A on \mathcal{P} , noted $d_{\mathcal{P}}^A$ is based on frequency of presence of \mathcal{P} persons who control F in the paths Υ related to the activity ω :

$$d_{\mathcal{P}}^A = \frac{|\{\sigma : \sigma \in \widehat{\Upsilon}^{\mathcal{C}} \wedge \exists P \in \mathcal{P}, \exists F \in \sigma, \text{control}(P, F)\}|}{|\widehat{\Upsilon}^{\mathcal{C}}|}$$

For instance, the degree of dependence of John on Apple for the activity “John edits the document toto.doc” is equal to 2/3.

The degree of dependence of John on Oracle for the activity “John edits the document toto.doc” is equal to 1/3.

The degree of dependence of John on Microsoft for the activity “John edits the document toto.doc” is equal to 1/3.

The degree of dependence of John on the group [Microsoft, Apple] for the activity “John edits the document toto.doc” is equal to 1.

The degree of dependence of John on the group [Microsoft, Oracle] for the activity “John edits the document toto.doc” is equal to 2/3.

The degree of dependence of John on the group [Oracle, Apple] for the activity “John edits the document toto.doc” is equal to 2/3.

5 Examples

5.1 GoogleDocs

To illustrate the meta-model, Figure 4 presents a model drawn by applying SOCIOPATH. This model represents a system where a user uses GoogleDocs.

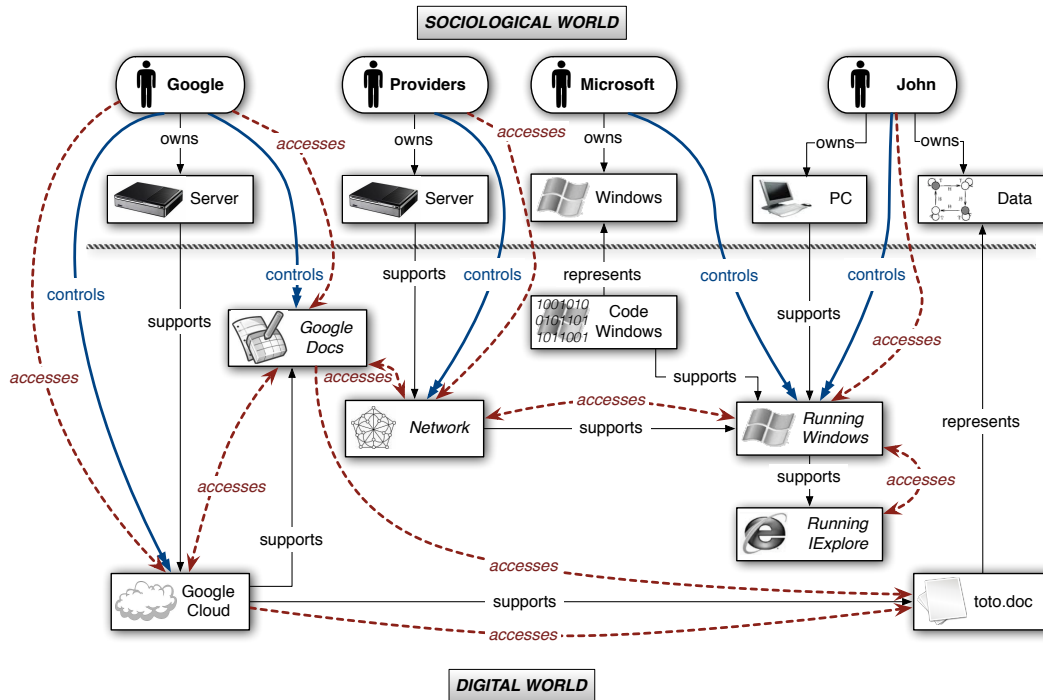


Figure 4: GoogleDocs snapshot

In the sociological world, the person ‘John’ owns some data and a PC. There are also Microsoft (provider of Windows and Internet Explorer) and Google (provider of GoogleDocs service and Google Cloud).

In the digital world, the Windows operating system is running on John’s PC and it supports Internet Explorer. John’s data are represented in the digital world by the document `toto.doc` which is supported by the physical resources owned by Google. We consider Google Cloud as the storage system used by the application GoogleDocs.

According to SOCIOPATH and its rules, John accesses Windows, which is supported by his PC. As Windows supports Internet Explorer, they can access each other and Windows controls Internet Explorer.

The physical resource that supports the network is connected to John’s PC and to the physical resources owned by Google, so the network services, GoogleDocs and Windows may access each other. The Google Cloud supports GoogleDocs that can access the file `toto.doc`.

Thus, if John wants to access his document, he passes through Windows, then Internet Explorer, then Windows again. Then by accessing the network, he gets to GoogleDocs, then to Google Cloud, and finally to his document. According to Section 2, 3 and 4, we can answer the questions introduced in Section2.

- Considering the activity “John accesses `toto.doc`”:

1. On whom John depends to access his data?

On Microsoft, Network providers and Google.

2. On which applications John depends to access his data?

On Windows, Network, GoogleDocs and Google Cloud.

• **Considering the activity “Other persons access `toto.doc`”:**

3. Who can access John’s data?

Microsoft, Network providers and Google.

4. Through which resources they can access the John’s data?

For Microsoft, through Windows, the network, Google Docs and Google Cloud.

For the network provider, through the network, Google Docs and Google Cloud.

For Google only through Google Cloud.

5. What are the necessary coalition between persons to access `toto.doc`?

A coalition is needed by all the persons listed in the answer to question 4, who do not own all the resources they must use. This makes evident that Google doesn’t need to collude with anyone.

• **Considering the activity “John accesses and understands `toto.doc`”:**

6. On whom John depends to understand his data?

On Microsoft, Network providers and Google.

7. On which applications John depends to understand his data?

On Windows, Internet Explorer, Network, GoogleDocs and Google Cloud.

5.2 SVN

Figure 5, shows the architecture of a system where the user ‘Philippe’ uses the application ‘SVN’ to reach his own document `toto.doc`. The latter is supported by a PC owned by an administrator, and shared with another user ‘Patricia’.

Formalizing such a system allows to illustrate the dependences of a user who performs an activity, *e.g.*, to check out the document `toto.doc`. As we said, in this work we do not deal with the formal definition of the criteria that characterize an activity. We just notice here that the activity “Philippe checks out `toto.doc`” implies several non-trivial aspects. For instance, Philippe must use an `svn` client that connects to an `svn` server process to access the given document; other possible paths that do not include this constraint are not acceptable, in this specific case.

In this example, Philippe owns a PC that supports MacOS, which supports the application SVN Client (SVNC). In order to keep the figure as readable as possible, not all the actors involved in this example are represented (*e.g.*, the persons who provide the aforementioned software). Philippe’s PC is connected to different networks: Orange (OrangeNW) and SFR (SFRNW), which are connected to a specific artifact representing the global Internet network (InternetNW). Then, the set of local networks also includes the national french university’s network (RenaterNW) and the one of the University of Nantes specifically (UnivNantesNW). The administrator’s PC is connected to UnivNantesNW. All the providers of these networks appear in our architecture in the sociological world (except the ones of University of Nantes, for sake of simplicity). Some nodes do not required a fine-grained detail description (*e.g.* the networks) and are considered as “black boxes”, controlled by their provider.

According to Section 2, 3 and 4, we can answer the questions introduced in Section2.

Considering the activity “Philippe checks out `toto.doc`”:

Philippe depends on a the following group of artifacts:

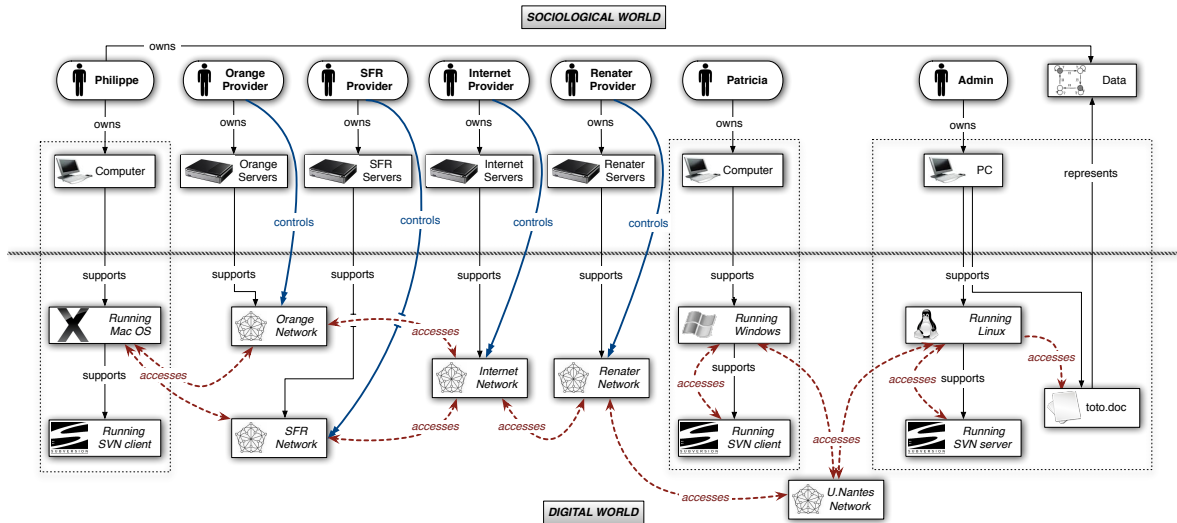


Figure 5: SVN

- [SFRNW, OrangeNW], [PhilippeMacOS], [AdminSVNS], [AdminLinux], [InternetNW], [RenaterNW], [UnivNantesNW], [PhilippeSVNC].

Philippe is forced to pass either through SFRNW or through OrangeNW and not through both of them, so he depends on this set, but not on each of the two. For he is forced to pass through all the other artifacts, he thus depends on each of them for the activity “Philippe checks out `toto.doc`”.

Philippe depends on a the following group of persons:

- [SFRNW Provider, OrangeNW Provider], [Philippe], [Apple], [SVN Provider], [InternetNW Provider], [RenaterNW Provider], [UnivNantesNW Provider], [Linux Provider], [The Administrator].

Philippe depends on the group [SFRNW Provider, OrangeNW Provider] because he is forced to pass through an artifact controlled by either one of them, and his degree of dependence on each of them is equal to 1/2. Beside this case, he depends on each of the other persons 100 %, because he is forced to pass through the artifacts they control for the activity “Philippe checks out `toto.doc`”.

Considering the activity “Philippe shares `toto.doc` with Patricia”:

We have to analyze also the paths that Patricia has to access `toto.doc`. So Philippe depends on a the following group of artifacts and persons respectively for the activity “Philippe shares `toto.doc` with Patricia”:

- [SFRNW, OrangeNW], [PhilippeMacOS],[PatriciaWindows], [AdminSVNS], [AdminLinux], [InternetNW], [RenaterNW], [UnivNantesNW], [PatriciaSVNC], [PhilippeSVNC].
- [SFRNW Provider, OrangeNW Provider], [Philippe], [Patricia], [Apple], [SVN Provider], [InternetNW Provider], [RenaterNW Provider], [UnivNantesNW Provider], [Linux Provider], [The Administrator].

In this case, if Patricia does not want to show Philippe her architecture, All her local part will look like a black box controlled by Patricia.

By analyzing the paths that other persons could have to access `toto.doc`, we conclude that each of the following persons: [SFRNW Provider], [OrangeNW Provider], [Philippe], [Patricia], [SVN Provider], [InternetNW Provider], [RenaterNW Provider], [UnivNantesNW Provider], [Linux Provider], [The Administrator] have a possible path to access `toto.doc`.

This makes evident that the administrator does not need to collude with anyone to access `toto.doc`, where all the other persons need a coalition with at least the administrator who controls the access to `toto.doc`.

6 Related work

There exist frameworks and tools to create models of IT (Information Technology) systems in the context of EAM (Enterprise Architecture Management). EAM aims at giving a structured description of large IT systems in terms of their business, application, information and technical layers, with the additional goal of understanding how existing architectures and/or applications should be changed to improve business or strategic goals.

EAM frameworks are mainly developed by governmental institutions (FEAF⁵, FDIC⁶), defence industries (DODAF⁷, NAF⁸, AGATE⁹) or large IT consortia (TOGAF¹⁰, GERAM¹¹, RM-ODP¹²). Their main concern is to assess the interoperability of different systems by clearly defining compatibility standards. There are few examples of proprietary (IAF¹³, OBASHI¹⁴, PROMIS¹⁵) and open source frameworks (TRAK¹⁶, MEGAF¹⁷) as well, though, which are more business-centric. A key benefit they provide is the ability to support decision making in changing businesses, by bringing together business models (*e.g.*, process models, organizational charts, *etc.*) and technical models (*e.g.*, systems architectures, data models, state diagrams, *etc.*).

The motivation of SOCIOPATH is completely different but it converges with those systems in some aspects. In the following we give a brief overview of some of those frameworks.

RM-ODP (Reference Model of Open Distributed Processing), also named ITU-T Rec.X.901-X.904 and ISO/IEC 10746, is a reference model in computer science, which provides a coordinating framework for the standardization of open distributed processing. It supports distribution, interworking, platform and technology independence, and portability, together with an enterprise architecture framework for the specification of ODP systems. Besides being mostly concerned with enterprise integration and business-related aspects, RM-ODP is a generic set of standards and tools to create and manage aspect-oriented models of systems. While RM-ODP analyze and decompose the systems in great details, according to five different viewpoints, each of them being very specific and mainly focused on standard compliance, SOCIOPATH gives a simpler overview of a system that is meant to inform the users about the relations that are implied by the system architectures, without exposing technical details. Moreover, RM-ODP does not consider trust relations among the users, like SOCIOPATH is meant to do.

TRAK is a general systems-oriented architecture framework that can be used to describe both hard and soft systems. It conforms to the standard for architecture description ISO/IEC 42010. TRAK has five architecture perspectives, each of which contains a number of related views (22 in total). The TRAK metamodel specifies the allowed object types that can be used when modelling and the relationships between them. Each TRAK

⁵http://www.whitehouse.gov/omb/assets/fea_docs/FEA_CRM_v23_Final_Oct_2007_Revised.pdf

⁶<http://www.fdic.gov/index.html>

⁷<http://cio-nii.defense.gov/sites/dodaf20/DM2.html>

⁸http://www.nhqcs.nato.int/ARCHITECTURE/_docs/NAF_v3/ANNEX1.pdf

⁹<http://www.achats.defense.gouv.fr/article33349>

¹⁰<http://pubs.opengroup.org/architecture/togaf9-doc/arch/>

¹¹<http://www.mel.nist.gov/workshop/iceimt97/ice-gera.htm>

¹²<http://www.rm-odp.net/>

¹³http://www.cappgemini.com/services-and-solutions/technology/soa/soa-solutions/ent_architecture/iaf/

¹⁴<http://www.obashi.co.uk/>

¹⁵<http://pro-mis.com/framework.html>

¹⁶<http://trak.sourceforge.net/>

¹⁷<http://megaf.di.univaq.it/>

view is specified by an architectural viewpoint which states what stakeholder concerns it addresses, it determines what is shown and how it is presented and how the view must be consistent with other views. The strong goal-oriented approach of TRAK is somehow similar to our focus on users' activity, which are an important concept in SOCIOPATH. Anyway, all TRAK's perspectives and views are meant to evaluate the enterprise architecture with respect to the ability to fulfil a given purpose, while SOCIOPATH aims at revealing hidden relations and dependences among persons and well identified digital actors of the system.

TOGAF is a framework for enterprise architecture which provides an approach for designing, planning, implementation, and governance of an enterprise information architecture. It is modelled at four levels: Business, Application, Data, and Technology. SOCIOPATH converges with TOGAF in some concepts of the technology level where the technical architecture of the enterprise is modelled. Unlike SOCIOPATH, the technical architecture of TOGAF focus on several aspects of the software engineering process (*e.g.*, system requirement, objectives, maintenance, evolution, reuse, integration) while describing the hardware, software and network infrastructure needed to support the deployment of applications.

The OBASHI methodology provides a framework and method for capturing, illustrating and modelling the relationships, dependencies and data flows between business and IT assets and resources in a business context. The six layers that modelize OBASHI are : Ownership, Business Process, Application, System, Hardware and Infrastructure. OBASHI gives a big picture that helps to design, optimise and monitor a business by analyzing the relations of Connection, Dependence, Spatial, Set, Layer and Sequential). SOCIOPATH converges with OBASHI in analyzing the relations in the level of application and system to deduce the relations between persons using these applications. OBASHI aims at providing a tool for the directors of the enterprise to monitor their business.

None of these works considers trust relations among users. SOCIOPATH aims at improving the users satisfaction and awareness, by making them aware of the relations of trust among social and digital actors of the used architecture.

7 Ongoing work and Conclusion

This paper introduced SOCIOPATH, a meta-model that formalizes systems in order to reveal the relations of dependence among participants. A formalism of SOCIOPATH is given by several definitions, upon which we have defined some rules, based on first order logic. These rules captures those aspects of the model that are needed to build the relations among the system's components.

These rules and definitions have been implemented in ProLog, in order to develop a tool, based on SOCIOPATH that infers dependences automatically. Such a tool may be very valuable in all the situations that require a person to evaluate the degree of inter-dependence of the various components of a given architecture. For instance, it may help a manager in understanding all the implications entailed by decisions such as: switching from a corporate licensed software to an open source alternative, choosing a network service provider over a competitor one, validating a risk assessment plan for a given logistic architecture, *etc.* Moreover, by applying SOCIOPATH to build a model of her own system, a user may easily evaluate the "cost" of replacing something, in terms of side-effects and dependence shifts. One may also be able to evaluate the system's exposition to risks of misbehavior or failures of components the system itself depends on.

SOCIOPATH can be used to point out accesses, controls and relations within an architecture. This is particularly useful to check whether the system respects the trust and the privacy expected by its users. Being able to test an architecture compliance with respect to users' privacy policies and trust models is one of our future goals.

We believe that the use of our meta-model is not limited to the few possible ways presented above. SOCIOPATH may be used by people of the real world, to better analyze and develop useful insights about the digital world, upon which everyone relies more and more.

References

- [1] Cornelli, F., Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: Choosing Reputable Servents in a P2P Network. In: Int. Conference on World Wide Web (WWW). (2002) 376–386
- [2] Damiani, E., di Vimercati, D.C., Paraboschi, S., Samarati, P., Violante, F.: A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In: Int. Conference on Computer and Communications Security (CCS). (2002) 207–216
- [3] Fahrenholtz, D., Lamersdorf, W.: Transactional Security for a Distributed Reputation Management System. In: Int. Conference on E-Commerce and Web Technologies (EC-WEB). (2002) 214–223
- [4] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The Eigentrust Algorithm for Reputation Management in P2P Networks. In: Int. Conference World Wide Web Conference (WWW). (2003) 640–651
- [5] Gupta, M., Judge, P., Ammar, M.: A Reputation System for Peer-to-Peer Networks. In: Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). (2003) 144–152
- [6] Carchiolo, V., Longheu, A., Malgeri, M.: Reliable Peers and Useful Resources: Searching for the Best Personalised Learning Path in a Trust and Recommendation-Aware Environment. *Information Sciences* **180** (2010) 1893–1907
- [7] Wang, L., Hill, R.: Trust Model for Open Resource Control Architecture. In: Int. Conference on Computer and Information Technology (CIT). (2010) 817–823
- [8] Yoon, J.P., Chen, Z.: Service Trustiness and Resource Legitimacy in Cloud Computing. In: Int. Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC). (2010) 250–257
- [9] Varalakshmi, P., Nandini, M., Krithika, K., Aarthi, R.: An Optimal Trust Based Resource Allocation Mechanism for Cross Domain Grid. In: Int. Conference on Recent Trends in Business Administration and Information Processing (BAIP). (2010) 342–348