A CONCEPTUAL MODEL FOR PRIVACY POLICIES

Alberto Coen-Porisini Pietro Colombo Sabrina Sicari Alberto Trombetta
Dipartimento di Informatica e Comunicazione – Università degli Studi dell'Insubria
via Mazzini 5, 21100 Varese, Italy
{alberto.coenporisini, pietro.colombo, sabrina.sicari, alberto.trombetta}@uninsubria.it

ABSTRACT
Nowadays privacy is a key issue and enterprises have adopted various strategies to protect customers privacy and to make public their privacy policies.
This paper presents a conceptual model for the definition and enforcement of privacy policies. The model is described by means of UML and it provides a clear and simple way for representing privacy policies. Furthermore, the model supports different ways for enforcing privacy policies by defining when the controls are to be done and what actions need to be performed. Finally, the paper introduces a small example in order to better explain the proposed approach.

KEY WORDS:
Privacy policies, conceptual models, UML, software engineering applications

1. Introduction

Nowadays privacy is a key issue and has received increasing attention from consumers, companies, researchers and legislators. Legislative acts, such as the European Union Directive for personal data [1], the Health Insurance Portability and Accountability Act [2] for healthcare and the Gramm Leach Bliley Act [3] for financial institutions, require governments and enterprises to protect the privacy of their citizens and customers, respectively. Although enterprises have adopted various strategies to protect customers privacy and to make public their privacy policies (e.g., publishing a privacy policy on websites possibly based on P3P [4]), none of these approaches include systematic mechanisms to describe how personal data are actually handled after they are collected. Moreover, privacy protection can only be achieved by enforcing privacy policies within an enterprise's online and offline data processing systems.
Aim of this paper is to define a conceptual model that provides a sound foundation for the definition and enforcement of privacy policies. The proposed model is defined using UML [5], [6] and represents a general schema that can be easily adopted in different contexts.
The paper is organized in the following way: Section 2 introduces the privacy model and discusses its main features; Section 3 presents an example; Section 4 discusses the related works, while Section 5 draws some conclusions and provides hints on the future work.

2. Modeling Privacy

A privacy policy defines the way in which data referring to individuals can be collected, processed and diffused according to the rights that individuals are entitled to.
The rest of the paper adopts the terminology introduced by the EU directive [1] which is summarized in what follows:

- *personal data* means any information relating to an identified or identifiable natural person (referred to as *data subject* or *subject*).
- *processing of personal data* (*processing*) means any operation or set of operations which is performed upon personal data, whether or not by automatic means, such as collection, recording, organization, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, blocking, erasure or destruction;
- *controller* means the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data;
- *processor* means a natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller;
- *the data subject's consent* (*consent*) means any freely given specific and informed indication of his/her wishes by which the data subject signifies his/her agreement to personal data relating to him/her being processed.

As a distinctive feature of a privacy policy, the processor is required to state for what *purpose* data are processed. A purpose can be defined either as a high-level activity (e.g., "marketing", "customer satisfaction") or as a set of actions (e.g., "compute the average price", "evaluate the customer needs"). Moreover, an *obligation* is a set of actions that the processor guarantees to perform, after the data have been processed. Also obligations have to be stated by the processor. Subjects, whenever their data are collected, must be informed of the purposes and of the obligations related to any processing. Moreover, subjects must grant their consent before any processing can be done. Finally, the consent can be given selectively that is, a subject can grant the consent for one purpose while denying it for another one.
The conceptual model proposed in this paper is described by means of UML and it provides a clear and simple way for representing every concept occurring in a privacy

policy (e.g. processors, obligation, purpose, etc.) along with their relationships. Furthermore, this approach provides a straightforward way towards the enforcement of privacy policies.

2.1 The UML Model

In order to provide a complete model in UML it is necessary to define both the structural (static) and behavioral (dynamic) aspects of any privacy policy.

2.1.1 Structural aspects

The structural aspects are defined using UML classes and their relationships such as associations, dependencies and generalizations. Figure 1 depicts a UML class diagram that provides a high level view of the basic structural elements of the model. A *PrivacyPolicy* is characterized by three types of classes: *User*, *Data* and *Action*. Users interact among them in order to perform some kind of action on data that refer to other users. Thus, an instance of *PrivacyPolicy* is characterized by specific instances of *User*, *Data* and *Action*, and by the relationships among such entities.

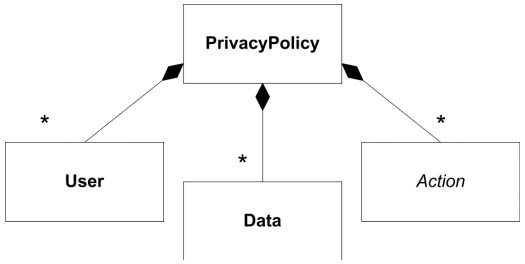

Figure 1: The Privacy Policy Class Diagram

Role [7] is a key concept of this approach that describes how users can behave in a specific context. Thus, users are characterized depending on the role they play. A first characterization introduces three different roles that a user can play: subject, processor or controller.

The class diagram reported in Figure 2 introduces the role concept by extending class *User* into three subclasses named *Subject*, *Processor* and *Controller*, respectively. Moreover, it also introduces extensions to classes *Data* and *Action* and it describes the relationships among the extended classes along with the needed services. Let us focus on the classes introduced by the diagram:

- *User* represents an actor either interested in processing data or involved by such processing. *User* is extended by means of three distinct classes to represent the different roles: *Subject*, which is anyone whose data are referred to, *Processor*, which asks for processing data by performing some kind of action on them and *Controller*, which defines the allowed actions that can be performed by the processors.
- *Data* represents the information referring to subjects that can be processed by processors. *Data* is extended by means of *Identifiable* data (e.g., name, address, phone n.) and *Sensible* data (e.g., health, religion). The former represents the information about subjects that can be used to uniquely identify them, while the latter represents information that deserves particular care and that should not be freely accessible.
- *Action* represents any operation performed by *User* (usually *Processor*). *Action* has been defined using an abstract class and it is extended by *Obligation*, *Processing* and *Purpose*. Moreover, each action can be recursively composed of purposes and obligations and therefore it is defined by means of an aggregation relationship between *Action* and both *Purpose* and *Obligation*.

Figure 2 depicts the aforementioned entities along with their relationships. For instance, the dependency relationship between *Action* and *Data* means that data are processed by actions, while the association between *Subject* and *Data* expresses data ownership.

Notice that this model can be extended in order to support the definition of policies related to different application domains. For example, to specify privacy policies compliant with the Italian privacy legislation [8], it is necessary to extend the model introducing the concept of "judicial data". Such extension can be easily realized by introducing a class *Judicial* that extends the class *Data.*

All the above entities interact among them exchanging information through interaction points represented by means of interfaces. Thus, an interface defines the services that a class can either implement or use (invoke). The model introduces the following five interfaces:

- *ConsentRequest*, which specifies the method *notify()*, that taken an instance of *Obligation*, an instance of *Purpose* and the *id* of *Controller*, notifies to the subject both the purposes and the obligations of the data processing. *ConsentRequest* is implemented by class *Subject* and is used by class *Controller*.
- *ConsentAcquisition*, which provides the method *grant()*, that taken an instance of *Purpose*, an instance of *Obligation*, the *id* of *Subject* and a boolean value, specifies whether the subject has granted the consent for processing his/her data. *ConsentAcquisition* is implemented by class *Controller* and is used by class *Subject*.
- *Control*, which provides the method *verify()* that, taken an instance of *Action*, returns whether the performed action was authorized that is, the consent has been granted. *Control* is implemented by class *Controller* and is used by class *Action*.
- *FactoryAction*, which provides the services to instantiate an *Action*. It is implemented by class *Controller* and is used by class *Processor*.
- *ActionBehavior*, which provides the method *run()* that represents the execution of an action.

Notice that, since the interface *ActionBehavior*, which defines the method *run()*, is provided by *Action*, each class extending *Action* can provide a specific implementation of such method. In fact, the specification of the behavior (i.e., the implementation of the method *run()*) can be provided by *Action* and inherited by its extensions, or can be provided by the extending classes.

Therefore, instances of *Purpose*, *Obligation* and *Processing* are a group of objects characterized by a common interface (*ActionBehavior*).

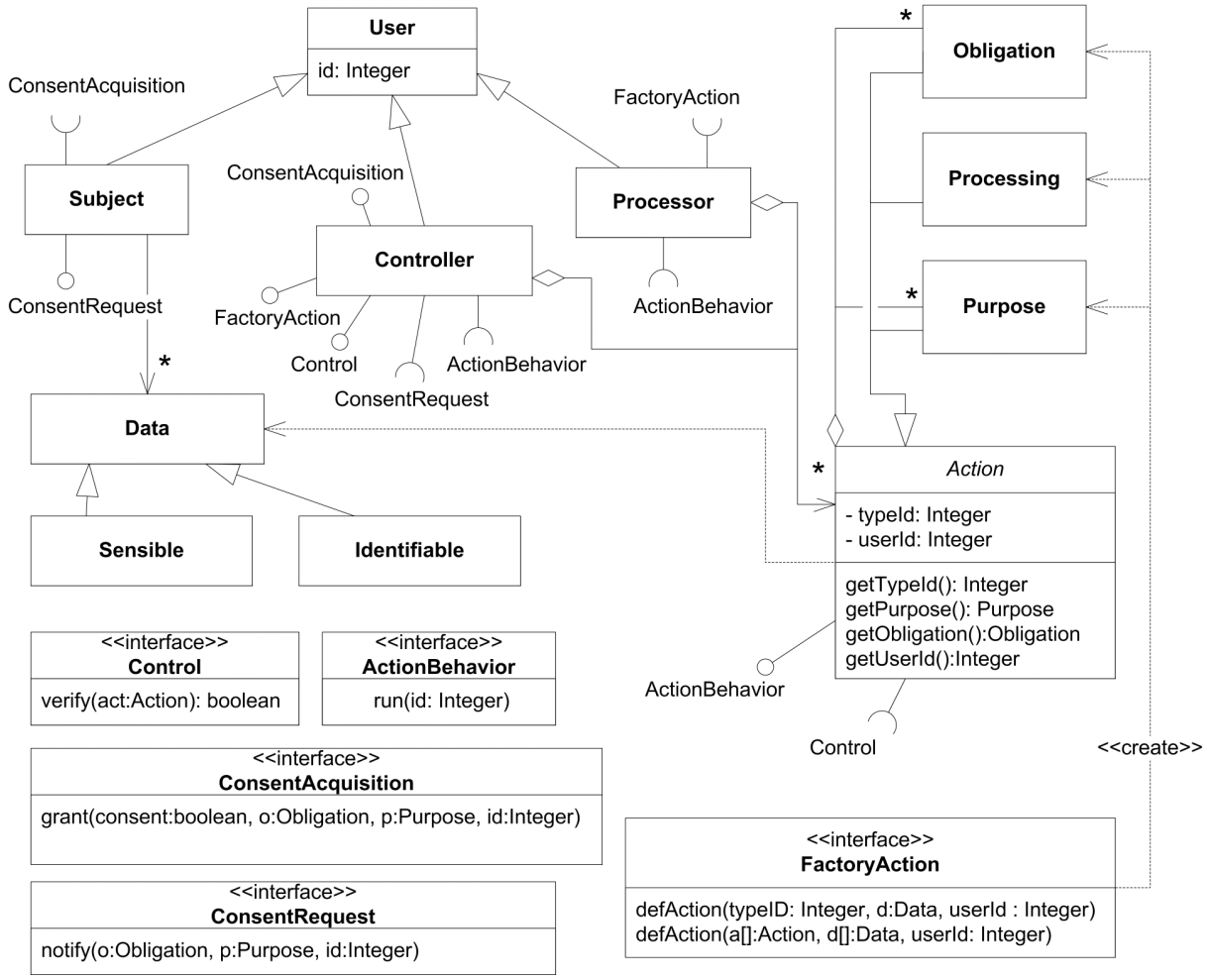

Figure 2: Privacy Class Diagram

2.1.2 Behavioral Aspects

Although the class diagram of Figure 2 faithfully represents the components of privacy policies, it does not explicitly express their dynamic aspects. This can be done by means of UML sequence diagrams.

Figure 3 reports two sequence diagrams showing general scenarios that describe interactions among the main actors and entities involved in the policy. These diagrams are intended to provide general schemas that can be specialized and extended for specific needs.

Both scenarios starts defining several actions and then notifying to *subject* the request to process his/her data. The subject can verify whether the provided purposes and obligations are compliant with his/her wishes and he/she grants or denies the consent.

More specifically, in the first scenario of Figure 3, named *PrivacyProtocol-Scenario1*, *controller* starts defining purposes, obligations and specific processing actions (e.g.., accessing some data, domestic bank transfer).

Notice that for each application domain (e.g., banks, hospitals, insurance companies) it is possible to identify a set of actions that almost every processor will try to execute in order to carry out his/her duty. Such actions are derive from the services that the company provides to its customers (subjects). In this context the subject consent is acquired *a priori*. As an example, let us consider the case of a potential bank customer that wants to open a checking account. The customer is informed that if he/she will request to make a domestic bank transfer, his/her data will be processed for the purpose of complying with his/her request and with the obligation of notifying national authorities whenever the transferred amount exceeds a given threshold. Notice that in this scenario the customer is informed and required to grant consent even if he/she will never request any bank transfer to be made.

Instead in the second scenario, named *PrivacyProtocol-Scenario2*, the *processor* interacts with the *controller* in order to define the purpose, the obligation and the processing that he/she wants to perform on the target data.

In particular, *processor* sends to the *controller* his/her *Id*, and the controller instantiates the three actions (a purpose, an obligation and a processing) requested by a processor for processing the data.. Notice that *Processing* represents the intention to perform a specific data processing, which can be carried out after the involved subject has granted the consent.

This scenario describes a situation in which specific actions are built in order to fulfill a specific request coming from a processor; in this case the subject is required to grant consent for actions for which the consent

was not granted *a priori*. As an example, let us consider the case of an actual bank customer requesting to make an international bank transfer. Since international bank transfer are less common than domestic ones, the customer was not informed nor he/she granted the consent when he/she opened the checking account. Therefore, when the customer requests the international bank transfer, he/she is informed that his/her data will be processed for the purpose of complying with his/her request with the obligation of notifying the National Security Agency. Notice that in this scenario the customer is informed and required to grant consent only when he/she requests for the first time to make an international bank transfer.

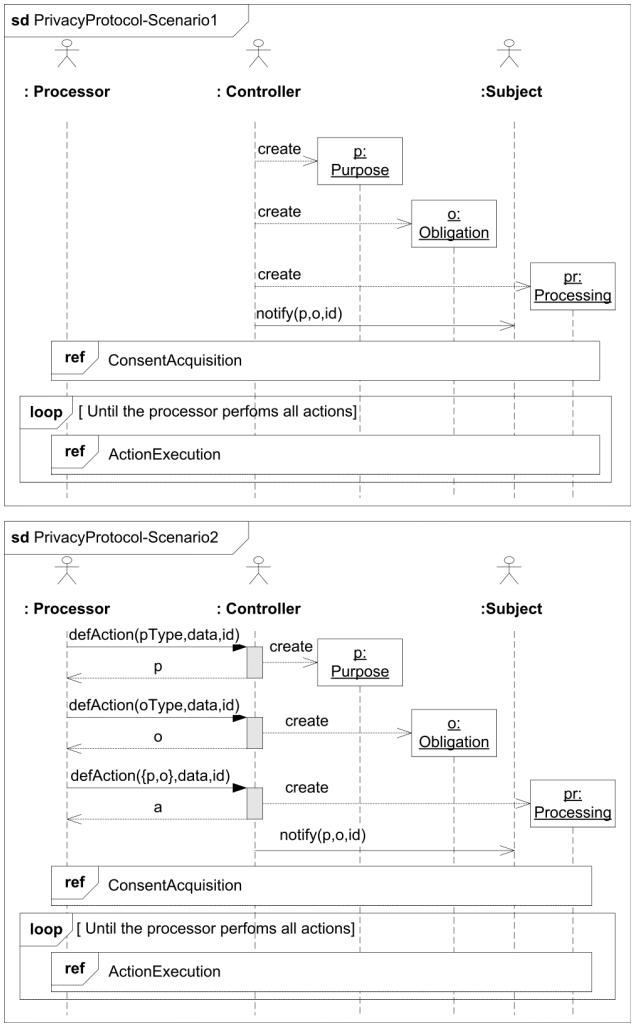

Figure 3: General interaction scenarios

Finally, both scenarios go on specifying the execution of the authorized actions (if any). As specified by the loop construct, depending on the processor needs, actions once authorized, can be executed multiple times.

In both scenarios of Figure 3 the consent acquisition and the action execution phases are not detailed. Instead, this is done by providing two more sequence diagrams, shown in Figure 4, that are referenced by those of Figure 3 using the *ref* construct.

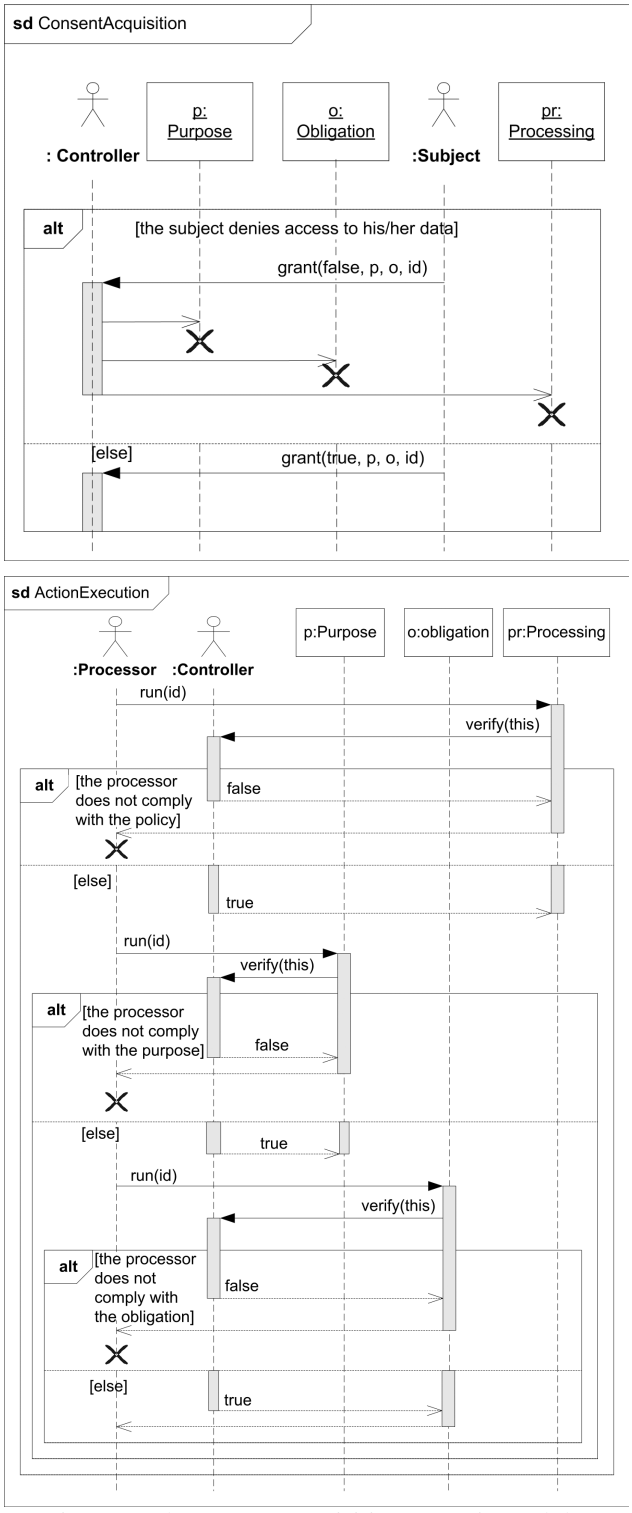

Figure 4: The consent acquisition scenario, and the scenario describing the execution of authorized actions

Thus, the first scenario of Figure 4, named *ConsentAcquisition*, specifies the possible behaviors of the subject by means of an *alt* frame, which allows one to describe alternative behaviors. In particular, if the owner denies the consent, no processor can process the data.

Similarly the second sequence diagram of Figure 4, named *ActionExecution*, shows that controller has to verify whether the actions performed by the processor are

correctly executed and whether they are compliant with the privacy policy. This is expressed by means of several *alt* frames that describe the different possibilities that may occur. In particular, if the controller verifies that the privacy policy is fulfilled, the processor is authorized to perform his/her operations, otherwise he/she cannot perform any action.

2.2 Privacy Policy Enforcement

Privacy policy enforcement consists in verifying the compliance of the actions performed by users with a given privacy policy. In general, there are two different ways in which such verification can be carried out: the first way consists in providing *ex-post* enforcement mechanisms that is, all the controls are done after all the actions related to a policy are performed (e.g., audit-based mechanisms). The second one consists in having run-time enforcement mechanisms that is, the effect of every action is checked before actual execution.

The model presented in this paper aims at conceptually defining the steps to be done in order to check at run-time the compliance of the behavior of a processor with a given privacy policy rather than proposing *ad-hoc* techniques for enforcing such compliance.

In other words, the model supports the integration of enforcement mechanisms in any privacy policy fitting therein or in an extended model derived using the usual object-oriented mechanisms (e.g., generalization). In fact, all actions required by a privacy policy are defined as instances of classes *Purpose*, *Obligation* and *Processing*, which are extensions of the abstract class *Action* (see Figure 2). Class *Action* requires the interface *Control* that, in turn, defines the method *verify()* to represent the verification of the compliance of any instance of *Action* with a given policy. In particular, the verification is carried out by *Controller*, while actions are executed by *Processor* (see Figure3 and Figure4). Thus, when *Processor* executes an action that is, it invokes the method *run()*, the method *verify()* is, in turn, invoked thus allowing the *Controller* to verify that *Action* is compliant with the policy. As a consequence, in case of non-compliance, the *Controller* can prevent *Processor* from executing any further action, as described in the *ActionExecution* scenario shown in Figure 4.

Notice that the enforcement mechanism cannot oblige the *Processor* to perform the required obligation.

3. An application example

This section provides an example concerning the definition of a privacy policy for data management in the context of a hospital.

A hospital is a complex organization that manages both identifiable and sensible data of patients, doctors and possibly other employees. Privacy is a fundamental issue since, for instance, an unauthorized data access may cause serious legal actions. Notice that privacy policies can be bypassed in case of life threatening situations when the impossibility to access essential data may prevent doctors to perform the required medical emergency procedures. However, for the sake of simplicity, emergency situations are not taken into account.

Data processing is managed by policies that specify 1) who is allowed to process data, and 2) what can be done with such data. Notice that it is fundamental that subjects receive clear and complete information concerning the kind of processing that will occur, so that they may consent to data processing.

The example makes use of a role based policy that is, the involved actors (processors and/or subjects[1], according to privacy terminology) are identified and classified depending on their role.

The following three roles have been identified: doctor, administrators and patient. Other possible roles, such as technical staff, security staff and so on, are not taken into account, for the sake of simplicity.

More specifically:

- Patients are hospitalized and therefore they need to be medically assisted. They should grant consent to doctors to access their case history allowing them to retrieve and store information concerning their health conditions.
- Doctors examine patients, access and modify their case histories, prescribe therapies, communicate to patients and to their relatives, manage emergencies and decide on patients discharge.
- Administrators perform bureaucratic activities: such as registering patients, sending written communication to relatives, preparing purchase orders and so on.

In order to avoid any privacy violation, it is necessary to define a privacy policy that limits the kind of data processing that can occur along with who can perform such processing. More specifically, administrators can access identifiable data of patients and identifiable and sensible data of employees, while doctors may access both identifiable and sensible data of patients and of their relatives.

The subject (i.e., a patient, a doctor) has to grant consent to process his/her data for specific purposes and obligations. For example, patients grant consent (when hospitalized) to allow doctors to access their case histories. The consent is constrained by the obligation that if a contagious disease is diagnosed, doctors will notify a qualified supervisor that in turn will notify Health Authorities.

3.1 Modelling privacy

The hospital domain is defined by means of a UML composite structure diagram showing the interested entities and their relationships. Notice that composite structure diagrams allow one to define how the entities introduced in a class diagram are instantiated, specifying their role and how they communicate one to another.

In the example the involved entities are represented by means of instances of the conceptual elements defined in

[1] A person can be at the same time a processor and a subject. For instance a doctor plays the role of a processor when he/she tries to access data of one of his/her patients, but he/she is a subject when the hospital administration has to prepare pay-rolls.

the class diagram of Figure 2, and are introduced in the composite structure diagram shown in Figure 5.
More specifically:

- *patient* is an instance of the class *Subject*
- *caseHistory* is an instance of *Data* and represents the medical information associated with a patient.
- *doctor* is an instance of *Processor*. The doctor needs to perform the following set of actions:

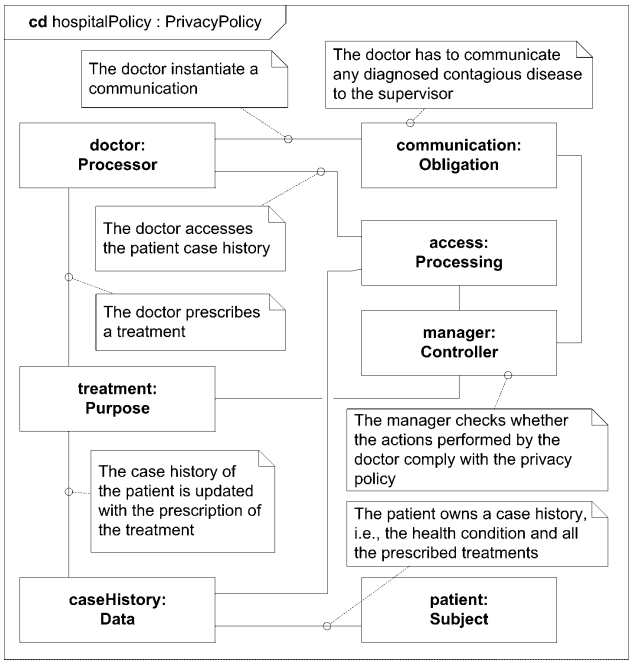

Figure 5: The composite structure diagram

  - accessing the patient case history; *access* is an instance of *Processing*;
  - once identified the disease, the doctor treats the patient by prescribing a treatment; *treatment* is an instance of *Purpose*;
  - reporting any contagious disease to a qualified supervisor; *communication* is an instance of *Obligation*.
- *manager* is an instance of *Controller* that verifies the correctness of the operations performed by *doctor*.

The relationship among the above entities is given by the following privacy policy:
"The doctor may access the case history of the patient in order to evaluate his/her health condition and to possibly prescribe a therapy, under the obligation to report any contagious disease to the supervisor. The manager verifies that all the actions are compliant with the privacy policy of the hospital."
Figure 6 shows a sequence diagrams describing the privacy policy in which:

- *doctor* requests to access *caseHistory*, under a privacy policy that specifies *treatment* as purpose and *communication* as obligation.
- Upon the reception of the access request *manager* checks whether the request is compliant with the stated privacy policy. Assuming that the patient has previously granted the consent, *manager* replies to doctor allowing him/her to carry out the request.
- *doctor* accesses the *caseHistory*, and after examining the patient, diagnoses a contagious disease. Then he/she prescribes a treatment and requests to update the *caseHistory*.
- *manager* checks the request against the policy and allows the *doctor* to proceed.
- *doctor*, once modified the case history, goes on carrying out the obligation; he/she notifies the diagnosis to a qualified supervisor.
- Finally *manager* checks whether the doctor communicated to the supervisor the patient case.

Notice that if the verification performed by *manager* fails, *doctor* is stopped and cannot operate anymore. In this way the enforcement of privacy policy is supported.

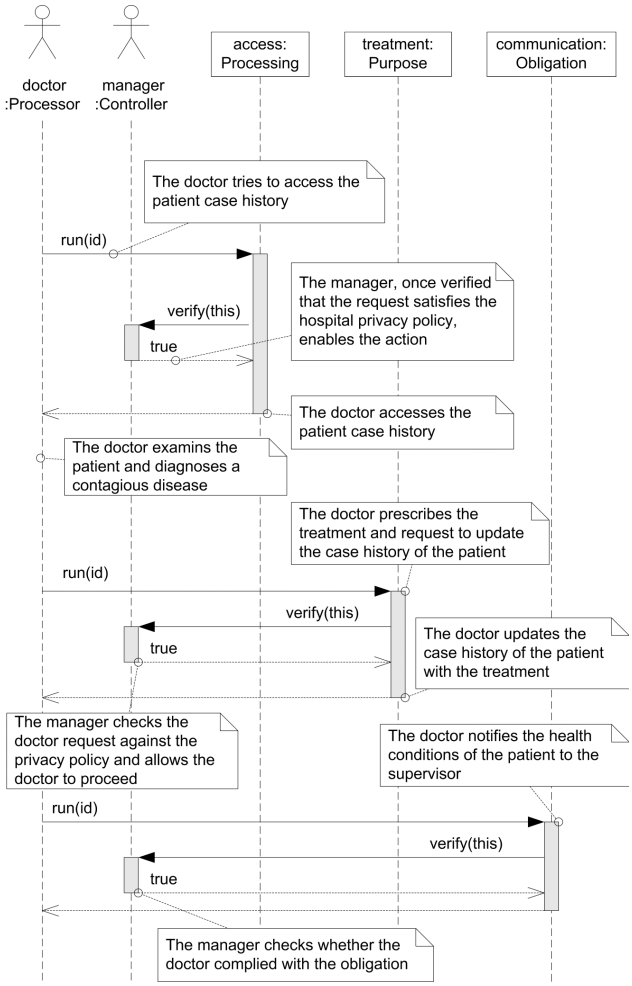

Figure 6: The main scenario of the hospital privacy policy

4. Related Works

While research on security is a well-established field, the problems and issues that arise when dealing with privacy have been under thorough investigation only in the recent years. The research efforts aiming at the protection of user privacy can be partitioned in two broad categories: Security-oriented Requirement Engineering (SRE) methodologies and Privacy Enhancing Technologies (PETs). The former focuses on methods for taking into account security issues (including privacy) during the

early stages of systems development, while the latter describes techniques to ensure privacy.

Several requirement engineering methodologies have been proposed to manage security issue at design level such as Kaos [9], Tropos [10], [11], [12], NFR [13], [14], and GBRAM [15]. In [16], the authors present a methodology, called PRIS, to incorporate privacy requirements into systems design process. PRIS is a requirement engineering methodology focused on privacy issues. It provides a set of concepts to model privacy requirements and a set of rules to transform requirements into implementation techniques.

Anton [17] uses both requirements engineering and goal-oriented analysis. Goal analysis [9], [15], [17] is a procedure to define goals and refine their meaning; it includes a set of logical mechanisms to identify, organize and justify software requirements. Anton's goal based approach includes a compliance activity to ensure that all policies are reflected into the actual system requirements. The approach focuses on the initial specification of security and privacy policies and their transformation into system requirements.

All of the above methodologies address the problem of how to state as clearly as possible the requirements that an information system should satisfy in order to be considered secure (with respect to a set of given security policies). This is different from our goal, which is to define a conceptual model for representing privacy policies. As explained in Section 2, this is achieved through the deployment of a model that represents all the relevant concepts of privacy related policies.

Enforcement is usually implemented by means of audit-based mechanisms. In particular, it is commonly assumed that an application providing privacy management support is backed up by some (usually relational) DBMS. This implies that the large majority of actions performed by the application (and by the processors interacting with it) are logged by the DBMS. In this context, Oracle's "fine-grained auditing" and IBM's Hippocratic DB-based compliance [18] are the most well-known approaches. The former allows one to log query information (e.g. user context, database state, etc.) every time a query involves some tables. In this way it is possible to rerun the query using the appropriate database state and check the retrieved information. The latter offers an advanced support for auditing purposes minimizing the need for large actions' logs.

This work differs from the above ones since it does not address a specific technique, being at a higher level of abstraction.

In [19] extensions to a RBDMS are provided in order to express P3P privacy policies, at schema definition level. Furthermore, the authors define mechanisms for translating P3P privacy policies into a properly extended SQL-like data definition language. This is different from our approach, since what we propose is a conceptual model for the definition of privacy policies (not to be necessarily expressed in P3P language) and for the specification of the needed functional modules of an application in order to enforce such policies.

Finally, in the field SRE methodologies, several techniques have been proposed in order to protect private data from unauthorized users. Typical examples are anonymizing techniques based on data suppression or randomization [20], [21]. However, these techniques do not require the definition of any privacy policy; rather they can be used as building blocks for realizing them.

5. Conclusions

This work presented an UML-based conceptual model suited for the definition of general privacy policies. The model allows one to define the concepts involved when dealing with the management of privacy-related information.

Furthermore, the proposed model takes into account the enforcement of such policies by defining when the controls are to be done and what are the actions to be performed in the case something goes wrong.

Instances of the model can be easily integrated at design time in new or existing systems taking advantage of the visibility and usability of UML. In addition UML supports automatic code generation in different programming languages.

Future works concern both the development of a prototype application for the management of privacy-related information, ranging from the definition of privacy policies to their corresponding enforcement, and the application of the proposed model for the definition of policies in a real system in order to analyze the scalability of the approach.

References

[1] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities of 23 November 1995 No L. 281 p. 31*

[2] http://www.hipaa.org

[3] http://www.glba.org

[4] W3C. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, November 2006. http://www.w3.org/P3P/.

[5] OMG. *Unified Modeling Language: Infrastructure*, 2007. Ver. 2.1.1, formal/07-02-04.

[6] OMG. *Unified Modeling Language: Superstructure*, 2007. Ver. 2.1.1, formal/07-02-03.

[7] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. Privacy-aware role-based access control. In *Proc. of ACM Symp. on Access Control Methods And Technologies (SACMAT'07)*, 2007.

[8] Decreto Legislativo n. 196, 30 Giugno 2003, Codice in materia di protezione dei dati personali, *Gazzetta Ufficiale n. 174 del 29-7-2003 - Suppl. Ord. n. 123*

[9] A. V. Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirement engineering. *IEEE Trans. Soft. Eng*, 26:978–1005, 2000.

[10] L. Liu, E. Yu, and J. Mylopoulos. Analyzing security requirements as relationships among strategic actors. In *SREIS'02, e-proceedings*, Raleigh, 2002.

[11] H. Mouratidis, P. Giorgini, and G. Mason. Integrating security and systems engineering towards the modelling of secure information system. In *15th Int. Conf. of Advanced Info. System Engineering (CAiSE'03)*, volume 2681 of *LNCS*, pages 63–78. Springer-Verlang, Berlin, 2003.

[12] H. Mouratidis, P. Giorgini, and G. A. Manson. An ontology for modelling security: The tropos approach. In V. Palade, R. J. Howlett, and L. C. Jain, editors, *KES*, volume 2773 of *Lecture Notes in Computer Science*, pages 1387–1394. Springer, 2003.

[13] L. Chung. Dealing with security requirements during the development of information system. In *5th Int. Conf. of Advanced Info.System Engineering (CaiSE'93)*, Paris (France).

[14] J. Mylopolulos, L. Chung, and B. Nixon. Representing and using non functional requirements: a process oriented approach. *IEEE Trans. Soft. Eng.*, 18:483–497, 1992.

[15] A. Anton. Goal-based requirements analysis. In *2nd IEEE Int. Conf. on Requirements Engineering (ICRE'96)*, pages 136–144, Colorado Springs Co, 1996.

[16] E. Kavakli, C. Kalloniatis, P. Loucopoulos, and S. Gritzalis. Incorporating privacy requirements into the system design process. the pris conceptual framework. *Internet research*, 16:978–1005, 2006.

[17] A. Anton and J. Earp. Strategies for developing policies and requirements for secure electronic commerce systems. In *1st ACM Workshop on Security and Privacy in E-commerce (CCS 2000)*, 2000.

[18] R. Agrawal, R. J. B. Jr., C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In M. A. Nascimento, M. T. Цzsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 516–527. Morgan Kaufmann, 2004.

[19] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan, and W. Rjaibi. Extending relational database systems to automatically enforce privacy policies. In *ICDE*, pages 1013–1022. IEEE Computer Society, 2005.

[20] T. Mielikinen. Privacy problems with anonymized transaction databases. In *7th Int. Conf. Discovery Science (DS 2004)*, Lecture Notes in Computer Science.

[21] A. Narayanan and V. Shmatikov. Obfuscated databases and group privacy. In *12th ACM conference on Computer and communications security (CCS '05)*, pages 102–111, New York, NY, USA, 2005. ACM Press.