From a Formal Privacy Model to its Implementation

Simone Fischer-Hübner, Amon Ott
Faculty for Informatics, University of Hamburg, Vogt-Kölln-Str. 30, D-20251 Hamburg
{fischer | 1ott}@informatik.uni-hamburg.de

Abstract: Privacy technologies are becoming more relevant, because individual privacy is at risk in the Global Information Society. In this paper, an updated version of a formal task-based privacy-model which can be used to technically enforce legal privacy requirements is presented. It is shown, how the privacy policy has been specified and implemented according to the Generalized Framework for Access Control (GFAC)- approach.

1. Introduction

On the way to the Global Information Society, individual privacy is at risk (see [Fischer-Hübner 1997a]). The EU-directive on data protection as well as privacy laws of many western states require basic privacy principles to be guaranteed when personal data are collected or processed, such as:

- *purpose binding* (personal data obtained for one purpose should not be used for another purpose without informed consent)
- *necessity of data collection and processing* (the collection and processing of personal data shall only be allowed, if it is necessary for tasks falling within the responsibility of the data processing agency).

In the Global Information Society, privacy cannot be efficiently implemented solely by legislative means. Data protection commissioners are therefore demanding that legal privacy requirements should be technically enforced and should be a design criteria for information systems. A recent study by the Dutch Data Protection Authority and the Information and Privacy Commissioner for the Province of Ontario / Canada [Registratiekamer et al. 1995] is exploring privacy enhancing technologies that are providing anonymity or pseudonymity for the users. Also the Privacy Class of the Common Criteria [Common Criteria 1998] is focused on anonymity, pseudonymity, unlinkability and unobservability of users.

However, in addition to the privacy protection of the users, there is also a need for privacy enhancing technologies to protect the data subjects (the so-called usees). Unfortunately, today's security models are mostly not appropriate to enforce basic privacy requirements, such as necessity of data processing or purpose binding.

The well-known Bell LaPadula model was designed to protect secrecy of classified information. Its Mandatory Access Control (MAC) policy restricts the access to objects based on the sensitivity of the information contained in the objects and the clearance of the subjects. However, personal data cannot be classified accurately by its sensitivity *per se*, because the sensitivity of personal data is related to the purpose and context of its use. In order to enforce privacy, it should be checked whether the purpose of the task, currently performed by the user who wants to access personal data, corresponds to the purpose for which that personal data were obtained (**requirement of purpose binding**).

Discretionary Access Control (DAC) restricts access to objects based on the identity of subjects and/or subject groups. DAC permits the granting and revoking of access privileges to data to be left to the discretion of a user with a certain access permission that has control over the data. However, under privacy aspects, personal data about a data subject should not be "owned" or "controlled" by another person. In order to protect privacy, an access control decision should not be determined by the user´s identity, but by the task that the individual user is currently performing. Personal data should only be accessible to a user, if such an access is necessary to perform his/her current task and if he/she is authorised to perform this task (**requirement of necessity of data processing**).

More recent security models, such as the Clark Wilson model [Clark/Wilson 1987] or Role-based Access Control models (e.g. [Ferraiolo/Kuhn 1992]) can enforce security aspects and requirements (e.g. integrity of personal data, users/roles are granted only such access rights as needed to perform their duties) that are more appropriate for the privacy protection of personal data. However, they are not especially designed to directly enforce privacy principles such as purpose binding.

In this paper, an updated and extended version of a formal task-based privacy model that was introduced in [Fischer-Hübner 1994], [Fischer-Hübner 1995] shall be presented which can be used to technically enforce legal privacy requirements in a system. It will be shown, how the privacy policy has been implemented together with other security policies according to the Generalized Framework for Access Control (GFAC, see [Abrams et al 1990], [LaPadula 1995]).

3. A Formal Privacy Model

The following section describes the concept of a formal security model that directly enforces basic legal privacy requirements, such as *purpose binding* or *necessity of data processing.* The **privacy policy** which is to be enforced by this model can be informally described as follows:

A user may only have access to personal data, if this access is necessary to perform his/her current task and only, if the user is authorised to perform this task. The user may only access data in a controlled manner by performing a (certified) transformation procedure for which the user´s current task is authorised. Besides, the purpose of his/her current task must correspond to the purposes for which the personal data were obtained or there has to be consent by the data subjects.

This **formal task-based privacy model** is defined as a state machine model. The privacy model contains the following *state variables, invariants, constraints (privacy properties)* and *state transition functions (model rules):*

3.1 State variables

In the privacy model, the following security-relevant (or better: privacy-relevant) state variables are defined:

Subjects S: Subjects are the active entities of the system (e.g., processes).

Active subject subj: identity of the subject that is currently active and is invoking a transition functions.

Objects O: Objects are passive entities (e.g. files, records).

Personal data objects $OP \subseteq O$: Personal data objects are objects containing personal data. Personal data are data about an identified or identifiable person.

Object-classes O-class: An object containing personal data can normally be classified by a certain class, e.g. patient record in a hospital, accounting data. **O-class = set of the different object-classes = {o-class$_1$, o-class$_2$, none...}.**
The class of non personal data is set to the predefined value "none".

Object-class function Class: Each object is classified by an object-class. A function
Class: O -> O-class is defined, where Class(O_j) is the object-class of the object O_j.
$\forall$ **$O_j \in O \setminus OP$: Class(O_j) = none.**

Tasks T: A subject shall be allowed to access an object only by performing a task. The tasks have to be defined for each application.

Current Task CT: The task that is currently performed by a subject is called its current task. If a subject is not currently performing a certain task, its current task is defined by the value Nil. A function **CT: S -> T $\cup$ {Nil}** is defined, where CT(S_i) is the current task of subject S_i.

Authorised Tasks AT: AT is a function that defines a non-empty set of tasks for a subject that this subject is authorised to perform. **AT: S -> $2^{T \cup \{Nil\}}$** $\setminus \emptyset$, where AT(S_i) is the set of tasks that S_i is authorised to perform.$\forall$ $S_i \in S$: Nil $\in$ AT(S_i). For an easier administration, authorised tasks could alternatively be defined for user roles instead of individual users.

User Role: Subjects can be categorised by certain security-relevant roles which users can play. **Role: S -> user-role,** where **user-role = {user, sec-officer, data-protection-officer, tp-manager,...}**. Further user roles should be defined, if authorised tasks are defined for user roles instead of individual users.

Users, who are defined as "responsible" for a tasks, shall be allowed to request to delegate this task. Therefore, a function "responsible" is defined: **Responsible: T -> 2^{S},** where Responsible(T_i) is the set of subjects, who can request to delegate the task T_i.

Purposes P: Every task has to serve a certain purpose. Besides, personal data have to be collected for certain purposes. Purposes have to be defined suitable for the system´s applications.

Purpose function for tasks T-Purpose: Each task serves exactly one purpose. (However, each purpose can be achieved by the performance of different tasks). A function
T-Purpose: T -> P is defined, where T-Purpose(T_i) is the purpose of task T_i.

Purpose function for object class O-Purposes: Each object-class has to have specified purposes for which the personal data of this class are collected.

A function **O-Purposes: O-class -> 2^{P}** $\setminus \emptyset$ is defined, where O-Purposes(O-class$_i$) are the purposes for which the object class O-class$_i$ was obtained. As is practice, non personal data should be usable for all different possible purposes: O-Purposes(none) = P.

Transformation Procedures TRANS: A subject cannot access an object arbitrarily. If it performs a task, it can execute certain transformation procedures, that are accessing objects in a controlled manner.

Current Transformation Procedure CTP: The transformation procedure that is currently executed by a subject is called its current transformation procedure. If a subject is currently not executing a transformation procedure, its current transformation procedure is defined by the value Nil. **CTP:** $\mathbf{S \text{ -> } TRANS \cup \{Nil\}}$**.**

Authorised Transformation Procedures ATP: While performing a task, a subject is authorised to run certain transformation procedures. **ATP:** $\mathbf{T \text{ -> } 2^{Trans\cup \{Nil\}}} \setminus \emptyset$,

$\forall T_i \in T$: Nil $\in ATP(T_i)$.

Access rights A: The access rights that a subject can have to a personal data object are defined by access attributes **A = {read, write, append, delete, create}**.

Necessary accesses NA: For any task, it has to be defined in advance which accesses to which object-classes by running which transformation procedure are needed to perform this task. This is done by defining the set $\mathbf{NA \subseteq T \times O\text{-}Class \times TRANS \times A}$ which consists of triples of the form $\mathbf{(T_i, o\text{-}class_j, transp_k, x)}$**,** where $transp_k \neq$ Nil. $(T_i, o\text{-}class_j, transp_k, x) \in NA$ means that for the performance of task T_i it is necessary to have x-access to objects of the object-class o-class$_j$ by running transformation procedure transp$_k$.

Current access set CA: A current x-access, where x $\in$ {read, write, append}, by a subject S_i to an object O_j in the current state is represented by a triple $\mathbf{(S_i, O_j, x)}$. The current access set $\mathbf{CA \subseteq S \times O \times A}$ is a set of triples representing all current accesses.

Consent C: According to most national privacy laws, the processing and use of personal data shall also be admissible, if the data subject has consented. A set $\mathbf{C \subseteq P \times O}$ is defined as a set of tuples (p_i, O_j). The tuple (p_i, O_j) means that the data subjects have consented that their personal data contained in the object O_j are processed for the purpose p_i.

3.1 Invariants

The following invariants define conditions for a system state to meet specific privacy principles (a so-called privacy-oriented state). They formally define the privacy policy stated above. To enforce this privacy policy, it has to be guaranteed that these invariants are fulfilled in each system state defined by the state variables.

1. ***A subject´s current task has to be authorised for the subject (task authorisation property):***
$\forall S_i: S : CT(S_i) \in AT(S_i)$.

2. A subject´s current transformation procedure has to be authorised for the subject´s current task (TP authorisation property): $\forall S_i: S : CTP(S_i) \in ATP(CT(S_i))$.

3. ***A subject may only have current access to a personal data object, if the access by executing a transformation procedure to objects of this class is needed to perform the current task (necessity of data processing):***
$\forall S_i:S, O_j:OP: (S_i,O_j, x) \in CA \Rightarrow (CT(S_i), Class(O_j), CTP(S_i) , x) \in NA$.

4. ***A subject may only have current access to a personal data object, if the purpose of its current task corresponds to the purposes for that objects of this class are obtained, or if there is a consent from the data subjects (purpose binding):***

$\forall\ S_i{:}S,\ O_j{:}OP{:}\ \ (S_i,O_j,\ x)\ \in\ CA\quad \Rightarrow$

T-Purpose $(CT\ (S_i)) \in$ O-Purposes $(Class(O_j)) \vee$ (T-Purpose$(\ CT(S_i)),O_j\ ) \in\ C$

If authorised tasks are defined for user roles, $AT(S_i)$ in the first privacy invariant has to be replaced by $AT(role(S_i))$.

Example: The following example demonstrates the differences between the concept of necessity of data processing and the concept of purpose binding: In a hospital, "diagnosis data" are collected for the purpose "medical treatment". For a researcher it might be necessary to do a statistical analysis by running a statistical program with read-access on diagnosis data. Thus, a corresponding necessary access $\in$ NA could be defined. However, the task "statistical analysis" serves the purpose "research". Consequently, the principle of purpose binding allows an access to diagnosis data of a patient for research purposes only, if the patient has consented to it.

3.3 Constraints

The privacy principles of necessity of data processing and purpose binding should also be enforced, if personal data are created or deleted. These principles can be formulated by adding constraints which are properties of sequences of states (a constraint differs from an invariant, because it takes into account the relationships between values in two successive states - before and after each state transition function). In the following notation the convention of placing the symbol * behind a state variable is used to refer to the new state.

1. A subject may create a personal data object, only if it is necessary for its current task:

$(CT(subj),\ o\text{-}class_k,\ CTP(subj),\ create) \notin\ NA\ \wedge\ O_j \notin\ OP \Rightarrow$

$O_j \notin\ OP^*\ \vee\ class^*(O_j) \neq o\text{-}class_k$

2. A subject may delete a personal date object, only if it is necessary for its current task:

$(CT(subj),\ Class(O_j),\ CTP(subj),\ delete) \notin\ NA\ \ \wedge\ \ O_j \in\ OP\ \Rightarrow O_j \in\ OP^*$

3. A subject may create a personal data object, only if the purpose of its current task corresponds to the purposes of the object´s class o-class$_k$ (purpose binding):

T-Purpose $(CT\ (S_i)) \notin$ O-Purposes $(o\text{-}class_k)\ \wedge\ \ O_j \notin\ OP \Rightarrow$

$O_j \notin\ OP^*\ \vee\ class^*(O_j) \neq o\text{-}class_k$

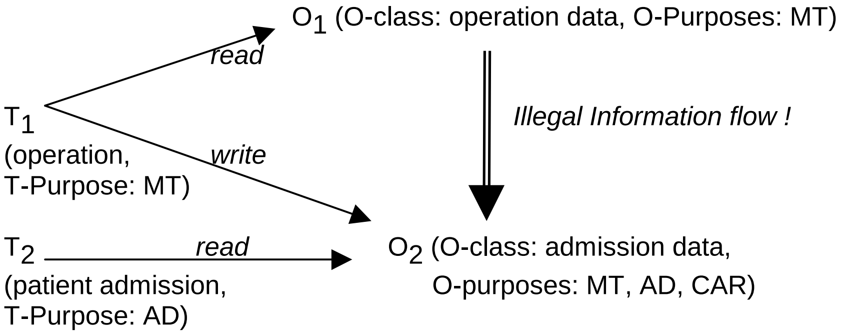
4. A subject may delete a personal data object, only if the purpose of its current task corresponds to the purposes of the object´s class, or if there is a consent from the data subjects (purpose binding):

T-Purpose $(CT\ (S_i)) \notin$ O-Purposes $(Class(O_j))\ \wedge$ (T-Purpose$(\ CT(S_i)),O_j\ ) \notin\ C\ \wedge O_j \in\ OP\ \Rightarrow$

$O_j \in\ OP^*$.

3.4 Information flow Control:

In the privacy model, a subject may access an object, if the purpose of his/her current task is contained in the set of purposes for which data of the object-class are obtained. The following attack-scenario (in a hospital environment), shown in figure 1, may allow illegal information flow.



MT: medical treatment, AD: administration, CAR: intensive care

Figure 1: Illegal Information Flow

A subject performing task T_1 could read object O_1 and write sensitive data from O_1 to object O_2. The consequence is that another subject that is performing T_2 (with T-Purpose AD) could then read data from O_1 (with O-Purposes(class(O1) = {MT}) which was written to O_2. This could violate the principle of purpose binding !
In particular, it has to be prevented that a subject can read a personal data object O_1 (with O-Purposes(Class(O_1)) $\subset$ P) and write it to a non-personal data object O_2 (with O-Purposes(Class(O_2)) = P).

Such illegal information flow can be prevented, if the following condition is guaranteed:
In any state, if a subject S_i has simultaneous read-access to object O_1 and write- or append-access to object O_2, then O-Purposes (class (O_1)) $\supseteq$ O-Purposes (class (O_2)).

Information flow control mechanisms by access control or by program certification of the transformation procedures could be used to satisfy this condition.

A simple flow control can be integrated into the access control mechanism of the operating system. A function **Input-Purposes: S -> $2^{\mathbf{P}}$** is defined, where Input-Purposes(S_i) is the set of purposes for that the data, which S_i has read, were obtained. Initially (after process creation) Input-Purpose (S_i) is set to P. If S_i gets read-access to an object O_j, Input-Purposes(S_i)* in the new system state is set to Input-Purpose (S_i) $\cap$ O-Purposes (class (O_j)).
Illegal information flow can be prevented, if it is guaranteed that S_i may not write to an object which was obtained for purposes not contained in Input-Purposes (S_i). The following information flow invariant shall help to prevent illegal information flow:

Information flow control invariant:

$\forall$ S_i: S, O_j:O, x $\in$ {write, append} : (S_i, O_j, x) $\in$ CA =>
O-Purposes (class (O_j)) $\subseteq$ Input-Purposes (S_i).

A "privacy-oriented state" is defined as a system state that satisfies the four privacy invariants1.-4. as well as the information flow invariant. A state sequence is defined as a "privacy-oriented state sequence", if each state of the state sequence is privacy-oriented and all successive states of the sequence satisfy the four privacy-constraints 1.-4.

3.5 State transition functions

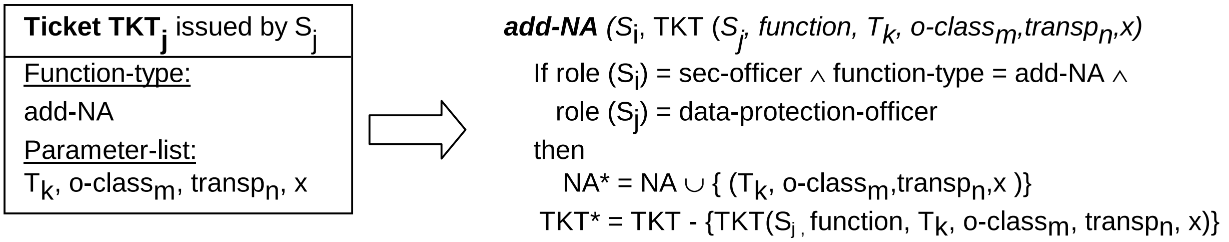
State transition functions, which describe all possible changes of state variables, are defined for the actions *get access*, *release access, create object, delete object, change current task*, *execute transformation procedure, exit transformation procedure*.

Besides, privileged functions are needed to define or change access control information, such as tasks, purposes, authorised tasks for a subject, authorised transformation procedures for a task, object classes and their purposes, necessary accesses and consents. These privileged functions can only be executed by the security officer. However, in order to support the "4-eyes principle", administration of access control information should be done in co-operation with another person, who cares for the privacy interests of the data subjects. This other person could for instance be the data protection officer in an organisation or the works council (according to the German data protection act, each organisation engaged in processing of personal data has to appoint a data protection officer. In the EU directive on data protection a similar role of a "data protection official" is defined). The data protection officer, who is responsible for enforcing privacy regulations in an organisation and for setting up a privacy policy, defines first all needed access control information. The security officer, who is responsible for enforcing security policies, then has to set the access control information according to these requirements.

In order to define and implement such a joint action scheme, another system-variable for a “one-time” ticket is introduced: A Ticket **TKT(S_i, function-type, parameter-list):** is issued by a Subject S_i and sent to the security officer (user in the role “sec-officer”). It means that S_i requests the security officer to perform a certain function with certain parameters. The issuer of a ticket is normally a user in the role "data-protection-officer" or a responsible user of a task, if authorisation to this task shall be granted to or revoked from another subject. TKT is defined as the set of all issued tickets. With the appropriate ticket the security officer can execute a corresponding privileged function:

Data Protection Officer S_j issues ticket:

Security Officer S_i can use this ticket to perform a privileged function:



Ticket TKT_j issued by S_j

Function-type:
add-NA
Parameter-list:
T_k, o-class$_m$, transp$_n$, x

```
add-NA ($S_i$, TKT ($S_j$, function, $T_k$, o-class$_m$,transp$_n$,x)
   If role ($S_i$) = sec-officer $\wedge$ function-type = add-NA $\wedge$
      role ($S_j$) = data-protection-officer
   then
      NA* = NA $\cup$ { ($T_k$, o-class$_m$,transp$_n$,x )}
   TKT* = TKT - {TKT($S_j$, function, $T_k$, o-class$_m$, transp$_n$, x)}
```

Figure 2: Joint action scheme for the execution of privileged functions.

For all transition functions it has been formally proven that they preserve the privacy invariants, constraints as well as information flow invariant. Thus, by mathematical induction it can be shown that, if a system that enforces the privacy model, starts in a privacy oriented state, then all the state sequences of the system will be privacy-oriented.

4. Specification and Implementation according to the Generalized Framework for Access Control Approach

In a project, it has been specified, how the privacy policy can be enforced according to the Generalized Framework for Access Control (GFAC) Approach in Unix System V (see [Fischer-Hübner 97b]). GFAC (see [Abrams et al. 1990], [LaPadula 1995]) is a framework for expressing and integrating multiple policy components to make it feasible to configure (and extend) a system with security policies chosen from a vendor provided set of options with confidence that the resulting system´s security policies will be properly enforced.

A draft top-level specification describing how GFAC enforcing the Bell LaPadula policy, the Clark Wilson policy and two supporting policies (Functional Control (FC) policy, Security Information Modification (SIM) policy) can be implemented in Unix System V, was published in [LaPadula 1995]. This top-level specification was further elaborated and extended with the policy rules of the privacy model. It was then used and adapted for the implementation and integration of the privacy policy according to the GFAC-approach together with other security policies (Bell LaPadula, FC, SIM) in the Linux operating system (see [Ott 1997]). Linux, although it is not designed for security, was chosen as a demonstration system, because it is a robust system, its source code is available and because it has functionalities of System V (LaPadula´s reference system). Furthermore, it keeps most common Unix standards. The results of the project (in particular, the privacy policy-specific Access Decision Facility- and Access Control Information modules) can be easily transferred to more secure Unix versions.
The GFAC approach was chosen, because it easily allows to combine the privacy policy that defines general privacy rules of national privacy legislation together with other more specific privacy regulations (e.g., privacy provisions of a hospital information law) which can be granted a higher priority. This is important, because according to the German Federal and state data protection laws, in so far as other legal provisions are applicable to personal data, such provisions shall take precedence. In addition, Linux was extended with pseudonymous audit mechanisms. Pseudonymous audit (see [Sobirey et al. 1997]) is a privacy-enhancing security audit technique, where user identifying data in audit records are pseudonymized.

4.1 The GFAC Approach

According to the GFAC approach, the Trusted Computer Base (TCB) consists of an access enforcement facility (AEF) and an access decision facility (ADF). ADF enforces the system´s mandatory security policies and a metapolicy to decide whether processes´ requests satisfy those security policies. AEF uses the ADF-decisions to implement the operations of system call functions.

For the GFAC implementation, the access control system of the system kernel is divided into the AEF and ADF components and the ACI-module which administrates Access Control Information (ACI, e.g. security attributes). Figure 3 shows the interactions between the system components. For each security-relevant system call, e.g. if a process requests to access an object (file, directory, security control data (scd) or interprocess communication data (ipc)), or if a process wants to clone itself or to send a signal, AEF sends a decision request to ADF. Parameters of the decision request are the request type, describing the desired type of functionality, the identification of the calling process and possibly the id of the target of access (a subject or an object). ADF evaluates its security policies by using the policy rules for the request type and the ACI needed for these rules. It then evaluates its metapolicy, which uses the decisions of the different security policies to finally decide about the process´ request. AEF then enforces the decision, by either performing the system call functionality or returning an error to the calling process. In the first case, after successful execution, ADF is notified, so that the attributes can be set accordingly. Finally, control is returned to the process.

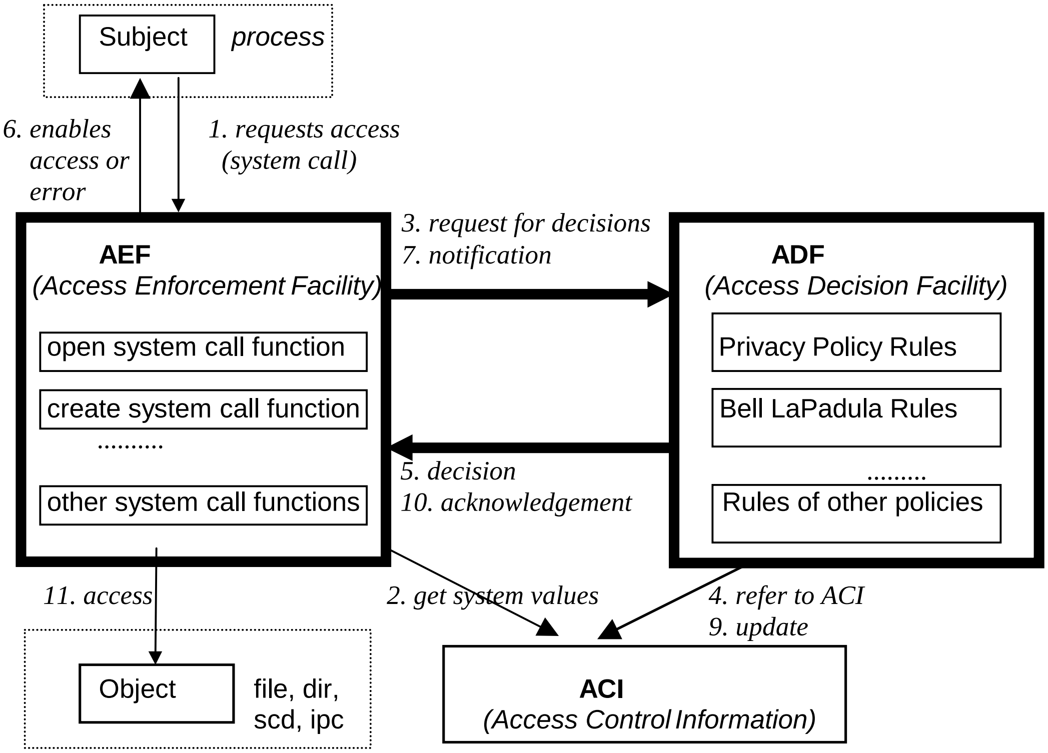


Figure 3: *Implementation of the GFAC Concept*

4.2 Access Enforcement Facility (AEF)

Since all security relevant accesses in a Unix system require system calls to the system kernel, AEF is enforced by extending all security-relevant system calls with ADF-requests and notification messages. For each system call, there is at least one ADF-request that relates to its functionality, but also further requests might be necessary. For example, the open system call has to be extended with ADF-requests for reading a directory, creating a directory/file, truncating a file and for append-/read-/write/read-write-opening a file.

Each state transition function of the privacy model is enforced by a system call. Therefore, AEF was extended with system calls (change-current-task, create-personal-data as well as all privileged system calls to add or delete ACI) which are exclusively needed for the privacy model.

4.3 ACI - Module

The ACI administration and ADF are implemented as independent modules. The ACI module is responsible for a reliable administration of security attributes of processes (process-ACI), of users (user-ACI) and of all resources that are needed and controlled by the security policies (object-ACI). Besides, it administrates additional access control information, such as the lists of necessary accesses or of defined tasks and their security attributes (in contrast to [Abrams et al. 1990], we do not differentiate between ACI which is associated with subjects or objects, and ACC (Access Control Context) information, which is additional access control information used in access control decision making). Access to ACI is only possible by defined function calls. Storage format of user and file ACI on secondary storage is independent from the used file system.

The tables below are listing the access control information needed for the system specification and implementation of the privacy policy, the corresponding variables of the privacy model and their value domains. Predefined values are printed in bold letters. ID is the abbreviation for identifier. The ACI, which is corresponding to state variables of the privacy model, can be explained by the meaning of their corresponding state variables. Other ACI attributes and data structures are explained below.

User-ACI	**Model variable**	**Values**
authorised-tasks	AT	a list of task-IDs including **NIL**
role	role	**sec-officer, user, data-protection-officer, tp-manager, system-admin**

Process-ACI	**Model variable**	**Values**
owner (pointer to user)		
transformation procedure	CTP	a transformation-procedure-ID or **NIL**
current_task	CT	a task-ID or **NIL**
process-type		**NIL, TP**
Input-Purposes	Input-Purposes	a list of purpose-IDs

Each Unix process is pointing to a user, who is the owner of the process. A process that is executing a transformation procedure is of the type TP. All other processes are of the type NIL.

Object-ACI	Model variable	Values
class	class	an object-class-ID or **none**
transformation-procedure	TRANS	a transformation-procedure-ID or **NIL**
object-type		**file, dir, ipc, scd**
data-type		**NIL, TP, personal data, non-personal-data**

The object-type values defined for the specification of the privacy policy are **file**, **dir** (directory), **ipc** and **scd**. **file** and **dir** have their obvious Unix meanings. **ipc** means "inter process communication" (e.g., message queues, shared memory in Unix System V, sockets in BSD Unix). **scd** means "system control data" (e.g., system time). The data-type of a file containing personal data is **personal-data**. The data-type of a file containing non-personal data is **non-personal-data**. The data-type containing the executable code of a (certified) transformation procedure is **TP**. All other file (e.g., other executable programs) have the data-type **NIL**.

Object-class-ACI	Model variable	Values
purposes	O-Purposes	a list of purpose-IDs

Further ACI	Model variable	Values
Necessary-Accesses	NA	list with entries of the form (task-ID, class-ID, transformation-procedure-ID, access-right)
Consent	C	list with entries of the form (purpose-ID, object-ID)
Purpose-list	P	list of purpose-IDs
Task-list	T	list of task-IDs
Ticket	TKT	records of the form (ticket-ID, issuer, function-type, parameter-list, timestamp)
O-Class-list	O-Class	list of o-class-IDs

Task-ACI	Model variable	Values
purpose	T-Purpose	a purpose-ID
authorised-TP	ATP	list of transformation-procedure-IDs including **NIL**
responsible	responsible	list of user-IDs

Ticket-ACI	Values
ticket-issuer	a user-ID
function-type	**add_authorised_tasks, delete_authorised_tasks, add_task, delete_task, add_NA, delete_NA add_purpose, delete_purpose, add_object-class, delete_object_class , add_authorised-TP, delete_authorised_TP, add_consent, delete_consent, add_responsible_user, delete_responsible_user, set-role, set-object-class**
parameter-list	(depending on the function-type)
timestamp	system time value

4.4 Access Decision Facility (ADF)

The ADF module receives access requests from AEF which are evaluated by the rules of the different policies. Figure 4., for example, specifies the privacy policy rule of ADF for the append-open request. The specification language should be intuitively understandable to a broad audience, but is also explained in [LaPadula 1995]. In this specification, the following predicates have the following meaning:

Necessary (task, class, transp, right) <=> (task, class, transp, right) is element of the table Necessary-Accesses.

Purpose-binding(task, class) <=> Purpose(task) is element of Purposes(class).

Consent(task, object) <=> (purpose(task), object) is element of the table consent.

```
CASE append-open
        SELECT CASE target[input-argument]
                CASE file
                        SELECT CASE data-type(object)
                                CASE personal-data
                                        IF  [Necessary (current-task (process), class (object),
                                               transformation-procedure (process), append )
                                                AND
                                                 Purpose-binding (current-task (process), class (object))
                                                OR
                                                consent ( current-task (process), object)]
                                                AND
                                                purposes(class(object)) in input-purposes(process)
                                        THEN
                                                return(YES)
                                        ELSE
                                                return(NO)
                                CASE TP (* it is not allowed to directly modify TPs *)
                                        return  (NO)
                                CASE non-personal-data
                                        IF purposes(class(object)) in input-purposes(process)
                                        THEN (* in this case input-purposes(process) is equal
                                                 to the set of all possible purposes *)
                                                return(YES)
                                        ELSE
                                        return(NO)
                                CASE ELSE
                                        return(NO);
                CASE ipc
                        IF class(object) is none (*object contains non-personal-data*)
                        THEN
                                IF purposes(class(object)) in input-purposes(process)
                                THEN
                                        return(YES)
                                ELSE
                                        return(NO)
                        ELSE                (* object contains personal data *)
                                IF Necessary (current-task(process), class(object),
                                                transformation-procedure (process), append)
                                AND
                                Purpose-binding (current-task (process), class(object))
                                AND
                                purposes(class(object)) in input-purposes(process)
                                THEN    (* consents are not defined for ipc-objects *)
                                        return (YES)
                                ELSE
                                        return (NO)
                CASE ELSE
                        return (UNDEFINED);
```

Figure 4: Specification of the ADF- privacy policy rule for the append-open request

In correspondence to the privacy model, personal data files can be created by using the *create_personal_data s*ystem call, for which the class of the new file has to be specified as a parameter. In order to preserve functionality with the Unix system, it should be allowed for a TP-type process to create an object also by using the ordinary Unix *creat* (create) system call. If a TP-type process creates a file or an ipc-object, these objects have to be treated as personal data to prevent illegal information flow. However, if the ordinary *creat* system call is invoked, no object class for the new file or ipc-type object is specified as a parameter. Therefore, for each purpose p_j a unique default-class with the purpose p_j is defined, with: $\text{purpose}(\text{default-class}(p_j)\ ) = p_j$.
If a TP-type process is creating a file or ipc-type object by using the ordinary creat system call, the object-class of the new object will be set to the default-class of the purpose of the processes´ current task.

5. Final Remarks

This project has shown how legal privacy requirements can be technically enforced in a system. The first tests of our implemented system with an imaginary hospital scenario, have demonstrated how privacy can be enhanced. The GFAC concept allows to combine the privacy policy with policies protecting more specific privacy requirements of certain application areas. In future, it is planned to analyse how easy it is to transfer the ADF- and ACI-modules to other system platforms. Besides, the system concept shall be tested in a real application environment.

Literature:

[Abrams et al. 1990] Marshall Abrams, K.Eggers, L.LaPadula, I.Olson, "A Generalized Framework for Access Control: An Informal Description", *Proceedings of the 13th National Computer Security Conference*, Washington, October 1990.

[Clark/ Wilson 1987] David Clark, David Wilson, "A Comparison of Commercial and Military Computer Security Policies", *Proceedings of the IEEE Computer Society Symposium on Security and Privacy,* Oakland, 1987.

[Common Criteria 1998] Common Criteria Editorial Board: Common Criteria for Information Technology Security Evaluation, Version 2.0, May 1998.

[Ferraiolo/Kuhn 1992] David Ferraiolo, Richard Kuhn, "Role-Based Access Controls", *Proceedings of the 15th National Computer Security Conference*, Baltimore MD, October 1992.

[Fischer-Hübner 1994] Simone Fischer-Hübner, “Towards a Privacy-Friendly Design and Usage of IT-Security Mechanisms”, *Proceedings of the 17th National Computer Security Conference*, Baltimore MD, October 1994.

[Fischer-Hübner 1995] Simone Fischer-Hübner, “Considering Privacy as a Security-Aspect: A Formal Privacy-Model”, DASY-Papers No. 5/95, Institute of Computer and System Sciences, Copenhagen Business School, 1995.

[Fischer-Hübner 1997a] Simone Fischer-Hübner, "Privacy at Risk in the Global Information Society", in: Jacques Berleur and Diane Whitehouse, Eds., 'An ethical global information society: Culture and democracy revisited', *Proceedings of the IFIP-WG9.2/9.5 Corfu International Conference*, May 8-10, 1997, Chapman &Hall, 1997.

[Fischer-Hübner 1997b] Simone Fischer-Hübner, "A Formal Task-based Privacy Model and its Implementation: An updated Report", *Proceedings of the Second Nordic Workshop on Secure Computer Systems NORDSEC*'97, Helsiniki, November 6-7, 1997.

[LaPadula 1995] Leonard LaPadula, “Rule-Set Modelling of Trusted Computer System”, Essay 9 in: M.Abrams, S.Jajodia, H. Podell, “Information Security - An integrated Collection of Essays”, IEEE Computer Society Press, 1995.

[Ott 1997] Amon Ott, "Regel-basierte Zugriffskontrolle nach dem Generalized Framework for Access Control-Ansatz am Beispiel Linux", Universität Hamburg, Fachbereich Informatik,
http://agn-www.informatik.uni-hamburg.de/people/1ott/rsbac/eng.htm.

[Registratiekamer et al. 1995] Registratiekamer, The Netherlands & Information and Privacy Commissioner /Ontario, Canada: “Privacy-Enhancing Technologies: The Path to Anonymity”, Vol.1, August 1995.

[Sobirey et al. 1997] Michael Sobirey, Simone Fischer-Hübner, Kai Rannenberg, "Pseudonymous Auditing for a Privacy-Enhanced Intrusion Detection", *Proceedings of the IFIP TC-11 Sec'97-Conference "Information Security in Research and Business"*, Copenhagen, May 14-16, Eds: L.Yngstroem, J.Carlsen, Capman&Hall, 1997.