UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Especificación formal de un modelo de consentimiento para el control de acceso a datos personales

Informe de Proyecto de Grado presentado por

Bruno Lartigau

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Carlos Luna

Montevideo, Octubre de 2024

Agradecimientos

Agradezco a mi madre, a mi padre, a mi hermana y a toda mi familia cercana por apoyarme incondicionalmente a lo largo de mi vida, tanto en los buenos como en los malos momentos.

También quiero agradecer a mis amigos cercanos, aquellos que he tenido desde la primaria y secundaria, así como a los que he conocido en el ámbito laboral, quienes me han acompañado no solo en momentos de ocio, sino también en muchos otros aspectos de mi vida.

A su vez, agradezco a la UdelaR por brindarme la oportunidad de cursar esta carrera, así como a los compañeros y docentes que he tenido en el recorrido.

En particular, quiero expresar mi gratitud a Carlos Luna por ser mi tutor, por guiarme y aconsejarme a lo largo de este trabajo, incluso en mis momentos de mayor incertidumbre. Asimismo, agradezco los aportes de Gustavo Betarte durante el tiempo que también estuvo acompañándome en la tutoría.

Resumen

En 2018, en el proyecto de grado “Formalización y validación de consentimiento para el control de acceso a datos personales”, se propuso un modelo semiformal de control de acceso denominado PCA-RBAC (Purpose and Consent Aware RBAC), que extiende el modelo de control de acceso basado en roles (RBAC) incorporando nociones de propósito y consentimiento. PCA-RBAC se desarrolló en base a requerimientos extraídos de la Ley 18.331 de Protección de Datos Personales, promulgada en Uruguay en 2008. Dicho modelo les permite a las personas otorgar diferentes consentimientos según el tipo de datos, garantizando que éstos sean respetados cuando se accede a su información personal.

Por otra parte, en la tesina de grado de la Universidad de Rosario (Argentina) “Especificación Formal del Modelo RBAC en el Cálculo de Construcciones Inductivas” (2008) se realizó una especificación formal del modelo RBAC utilizando el asistente de pruebas Coq y el Cálculo de Construcciones Inductivas. Este trabajo se motivó por el hecho de que el National Institute of Standards and Technology (NIST), que previamente estandarizó el modelo, no llevó a cabo ningún análisis sobre su corrección ni sobre sus propiedades.

En el presente proyecto de grado, siguiendo la metodología aplicada en el trabajo anterior, se especifica formalmente el modelo PCA-RBAC. Se define su estructura y especifica un estado del sistema junto con una serie de comandos, para evaluar su comportamiento ante determinadas acciones. Además, se realiza un análisis formal de algunas propiedades esenciales de seguridad del modelo; en particular, que no se permita el acceso a información sensible si no se cumplen los requisitos necesarios en cuanto a propósito y consentimiento. La especificación del sistema como una máquina de estados resulta fundamental para estudiar sus propiedades rigurosamente.

Palabras clave: Privacidad, Datos Personales, RBAC, PCA-RBAC, Propósito, Consentimiento, Formalización, CCI, Coq, Análisis, Propiedades.

Índice general

Capítulo 1: Introducción

Un concepto que ha sido relevante a lo largo de la historia, especialmente en los últimos tiempos debido a los avances en la informática, es el concepto de privacidad. Guillermo Rodríguez aborda este tema en su tesis de grado “Formalización y validación de consentimiento para el control de acceso a datos personales” [1]. El trabajo se enfoca en la definición propuesta por Westin [2], quien establece la privacidad como el derecho de cada individuo, grupo o institución a determinar por sí mismo en qué momento, de qué forma y hasta qué punto desea compartir su información personal a terceros.

Dado que la privacidad de la información puede verse comprometida por la posibilidad de deducir patrones y comportamientos [3, 4, 5], o por el uso de la información con fines distintos a los inicialmente previstos [4], según Solove, es necesario regular la protección de los datos personales. En Uruguay, la Ley N° 18.331 de Protección de Datos Personales [6], promulgada en 2008, establece que toda persona tiene derecho a la protección de sus datos personales (Art. 1), y que este derecho se extiende también a las personas jurídicas (Art. 2).

Ann Cavoukian [7] sostiene que la privacidad debe estar integrada en los sistemas de datos y tecnologías desde su diseño, así como de los protocolos y objetivos de las empresas. Por esta razón, en su artículo plantea los siete principios fundamentales de Privacy by Design para lograr dicho objetivo. Sin embargo, debido a la complejidad para implementar estos principios [8], es necesario proponer un modelo que represente de manera más efectiva las políticas de privacidad.

El primero en considerarse es Role-Based Access Control (RBAC) [9, 10], un modelo de control de acceso que define roles asociados a permisos y asignados a los usuarios correspondientes. Este enfoque evita una asignación directa entre usuarios y permisos, simplificando el modelado de políticas de seguridad. Los roles tienden a ser más estables que los usuarios y permisos, ya que suelen describir responsabilidades dentro de una organización, lo que permite reasignar permisos a diferentes roles según las necesidades.

Con el objetivo de estandarizar el modelo RBAC, el National Institute of Standards and Technology (NIST) [11, 12] propone su especificación en tres niveles acumulativos: Core RBAC, Hierarchical RBAC y Constrained RBAC.

En Core RBAC, las relaciones entre roles, usuarios y permisos son de tipo muchos-a-muchos (un rol puede tener muchos usuarios y permisos, y un usuario o permiso puede asociarse con muchos roles). Los permisos se representan mediante operaciones que ejecutan tareas sobre objetos que contienen información. Además, el modelo contempla sesiones en las que los usuarios se conectan, permitiéndoles activar y desactivar los roles asignados.

Hierarchical RBAC extiende la especificación de Core RBAC añadiendo una jerarquía de roles. Esta jerarquía se basa en una relación de orden parcial, donde un rol r considerado superior (o $senior$) a otro rol r' hereda sus permisos, mientras que r' adquiere los usuarios de r, es decir, todos los usuarios asignados a r se heredan a r'. En la figura 1 se ilustra este modelo conceptual de Hierarchical RBAC.

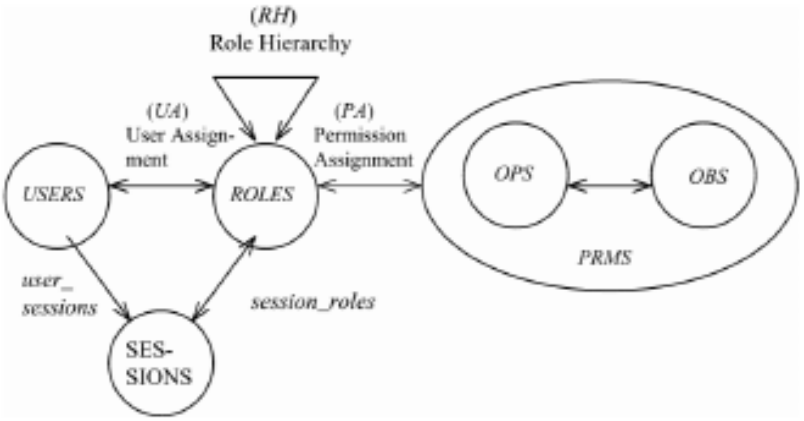

Figura 1: Modelo conceptual de Hierarchical RBAC [12]

Por otro lado, Constrained RBAC incluye restricciones adicionales en las asignaciones, con el objetivo de evitar que los usuarios sobrepasen sus niveles de permisos. Las restricciones pueden ser estáticas (SSD), limitando la cantidad de roles que se pueden asignar simultáneamente, o dinámicas (DSD), restringiendo qué roles pueden activarse al mismo tiempo en una sesión.

Un punto importante a considerar es que la estandarización propuesta por el NIST no profundiza en la corrección del modelo ni en el análisis de sus propiedades. Por esta razón, Cristian Daniel Rosa, en su tesina de grado “Especificación Formal del Modelo RBAC en el Cálculo de Construcciones Inductivas” [13], realiza un análisis detallado de RBAC, utilizando el Cálculo de Construcciones Inductivas (CCI) [14, 15] y el asistente de pruebas Coq [16, 17]. Esta formalización permite estudiar el modelo en profundidad, verificando su semántica, estudiando posibles ambigüedades o imprecisiones, y demostrando formalmente algunas propiedades básicas de seguridad del modelo. Este trabajo es de gran utilidad para el desarrollo de este proyecto.

Una problemática que Guillermo Rodríguez plantea en su proyecto [1] es cómo asegurar que los sistemas informáticos, después de recolectar la información que necesitan, cumplan con las políticas y regulaciones de privacidad. Aunque se han propuesto modelos que extienden RBAC con este fin, como Purpose-aware RBAC (PuRBAC) [18] o Privacy-aware RBAC (P-RBAC) [19], ninguno logra resolver completamente el problema. Además, se ha propuesto un framework para gestionar la información personal y los consentimientos de cada individuo según las políticas de privacidad vigentes. Este enfoque, conocido como Sticky Policy Paradigm [20], busca controlar el uso de la información en función de los propósitos definidos y los consentimientos otorgados.

Con el fin de abordar este problema de manera más efectiva, se elaboró un documento [1] en el que se identificaron algunos de los requerimientos más relevantes derivados de la Ley 18.331 [6]. Entre los principios destacados se encuentran el previo consentimiento (Art. 4 literal C y Art. 9) y de finalidad del tratamiento (Art. 8), lo que implica requerimientos como almacenar los consentimientos de los titulares de los datos junto con sus respectivas finalidades, y

verificar que los datos se utilicen exclusivamente para los fines establecidos en dichos consentimientos.

Con base en lo planteado hasta ahora, sumado al análisis realizado por Guillermo Rodríguez en su tesis [1], se busca un modelo que funcione como nexo entre el ámbito informático y el jurídico. Para ello, se propone el modelo PCA-RBAC, que busca abordar los problemas mencionados anteriormente, incorporando elementos clave que RBAC no contempla, como el propósito, el consentimiento y el manejo de la información personal. Este nuevo modelo tiene la finalidad de permitir que los usuarios otorguen consentimientos para diferentes tipos de datos, de manera que éstos sean considerados a la hora de acceder a datos personales.

Sin embargo, aunque el modelo fue presentado de manera semiformal, no se analizó con la profundidad que Cristian Rosa lo hizo en su trabajo con RBAC [13]. Por ello, el objetivo principal de este proyecto es especificar formalmente el modelo PCA-RBAC, y someterlo a un análisis riguroso utilizando el CCI y el asistente de pruebas Coq, con el fin de llegar a un modelo formal que incluya un estudio adecuado y riguroso de sus propiedades de seguridad.

Este documento se organiza en capítulos, desarrollando en cada uno aspectos fundamentales del trabajo.

El capítulo 2 presenta el modelo PCA-RBAC propuesto por Guillermo Rodríguez. Se analiza la especificación semiformal realizada y se la compara con su predecesor, RBAC. Se destacan las diferencias significativas y evalúa cómo se representa la política de privacidad, junto con el análisis de algunas limitaciones del modelo.

El capítulo 3 se enfoca en la especificación formal de PCA-RBAC utilizando el CCI y Coq. Se explican sus elementos y relaciones, y se estudia su comportamiento al definir una serie de comandos que puedan afectar su estado en el sistema.

El capítulo 4 aborda la verificación del modelo, analizando sus propiedades más relevantes mediante el asistente de pruebas. En particular, se busca demostrar que no se permite el acceso a la información personal si no se cumplen los requisitos necesarios respecto al propósito y consentimiento.

Finalmente, el capítulo 5 presenta las conclusiones de este proyecto y propone posibles trabajos a realizar en un futuro para complementarlo.

En el enlace https://www.fing.edu.uy/~cluna/pca-rbac-coq.v está disponible el código completo desarrollado en este proyecto, con formalizaciones y el análisis de propiedades, utilizando el asistente de pruebas Coq.

Capítulo 2: Contexto

Este capítulo aborda el contexto del proyecto, centrándose en la descripción y análisis del modelo PCA-RBAC [1]. En primer lugar, se presenta de forma general, destacando sus principales contribuciones en comparación con otros modelos RBAC. Luego, se describen en detalle sus elementos y relaciones, incluyendo la restricción no estructural, aunque sin profundizar en su formalización, que se desarrollará en el Capítulo 3. Finalmente, se discute la representación matemática de la política de privacidad y se señalan algunas de las limitaciones del modelo.

2.1. Contribuciones del modelo PCA-RBAC

El modelo de control de acceso PCA-RBAC (Purpose and Consent Aware RBAC) se basa principalmente en RBAC, teniendo en cuenta su estandarización propuesta por el NIST [11, 12], y en P-RBAC [19], incorporando aspectos que lo hacen más abarcativo que estos modelos previos.

Una de sus principales contribuciones es que considera explícitamente al titular de los datos, quien generalmente no es tenido en cuenta, como sí ocurre con los usuarios del sistema, los recursos accedidos y las operaciones sobre estos. En PCA-RBAC, el titular es responsable de autorizar o denegar el acceso a su información personal mediante el otorgamiento de consentimientos.

Además, PCA-RBAC cumple con el framework sticky policy paradigm [20], ya que los datos están regidos por los consentimientos otorgados por el titular, incluso si la política de privacidad de la organización sufre modificaciones.

Otro aporte significativo es la representación detallada de los consentimientos, lo que permite que los individuos autoricen el uso de diferentes tipos de información para distintos fines, en lugar de otorgar un consentimiento genérico para todos los propósitos. De esta manera, los usuarios pueden seleccionar de forma precisa los fines para los que desean que se utilice su información.

Un aspecto importante es que el modelo puede gestionar tanto la información personal, que requiere de un propósito y consentimiento para su uso, como la información no personal, que no los necesita. Este enfoque hace que PCA-RBAC no solo funcione como un modelo de control de acceso general, sino que también integre principios de privacidad, alineándose con los principios de Privacy by Design [7].

Por último, PCA-RBAC establece un vínculo sólido entre las estructuras de datos de los sistemas TI y las políticas de privacidad, garantizando su cumplimiento en estos sistemas.

2.2. Elementos y relaciones de PCA-RBAC

A continuación, se describe la estructura y los componentes del modelo PCA-RBAC, cuya especificación formal se abordará a partir del capítulo 3. En la figura 2 se ilustra su representación, en la que los elementos se representan mediante nodos y las relaciones entre ellos a través de flechas, de manera similar a la figura 1 de Hierarchical RBAC [12]. En este caso, además de las relaciones muchos-a-muchos vistas anteriormente, también existen algunas relaciones uno-a-muchos, donde un elemento x puede asociarse con varios elementos y, pero cada y sólo puede asociarse con un único x.

Sus elementos principales son aquellos heredados de Core RBAC [12], es decir $USERS$ (usuarios), $ROLES$ (roles) y $SESSIONS$ (sesiones). Los usuarios tienen roles asignados mediante la relación $User\ Assignment\ (UA)$, y pueden crear sesiones en tiempo real, activando o desactivando subconjuntos de roles asignados. La relación entre usuarios y sesiones se denomina $SessionUser$, mientras que la relación entre roles y sesiones se llama $SessionRoles$. Asimismo, la jerarquía de roles heredada de Hierarchical RBAC se representa mediante la relación $Role\ Hierarchy\ (RH)$.

Por otro lado, se tienen los elementos $OPERS$ (operaciones) y $OBJS$ (objetos), también heredados de RBAC. A partir de estos elementos se define $PERMISSIONS$ (permisos), obtenido luego de ejecutar una operación sobre un objeto. Estos permisos se asignan a roles mediante la relación $Permission\ Assignment\ (PA)$.

En cuanto a los nuevos elementos que introduce PCA-RBAC, se incluye un nuevo tipo de permiso llamado $PRIVACY\ PERMISSIONS\ (PP)$, utilizado para gestionar la información personal. Para formar este permiso de privacidad, se necesita un permiso convencional y un elemento $PURPOSES$ (propósitos), que especifica el uso permitido de la información. Al igual que los permisos convencionales, los permisos de privacidad se pueden asignar a roles a través de la relación $Privacy\ Permission\ Assignment\ (PPA)$, permitiendo que un rol ejerza tanto un permiso convencional como un permiso de privacidad.

Otro elemento distintivo del modelo son los $PRIVACY\ DATATYPES$ (tipos de privacidad), que modelan a los tipos de datos que representan información personal. Estos tipos se especifican en las políticas de privacidad en lenguaje natural, y cuando se vinculan con propósitos de uso específicos, generan los elementos denominados $CONSENTS$ (consentimientos). Los consentimientos, que son otorgados por el titular de los datos ($OWNERS$), se representan mediante la relación $Owner\ Consent\ (OC)$.

Finalmente, los objetos que contienen información personal se vinculan a uno o más tipos de privacidad a través de la relación $Data\ Mapping\ (DM)$. Además, cada objeto pertenece a un titular de datos mediante la relación $ObjectOwner$.

A modo de esquema, los conjuntos y relaciones del modelo especificados formalmente son los siguientes:

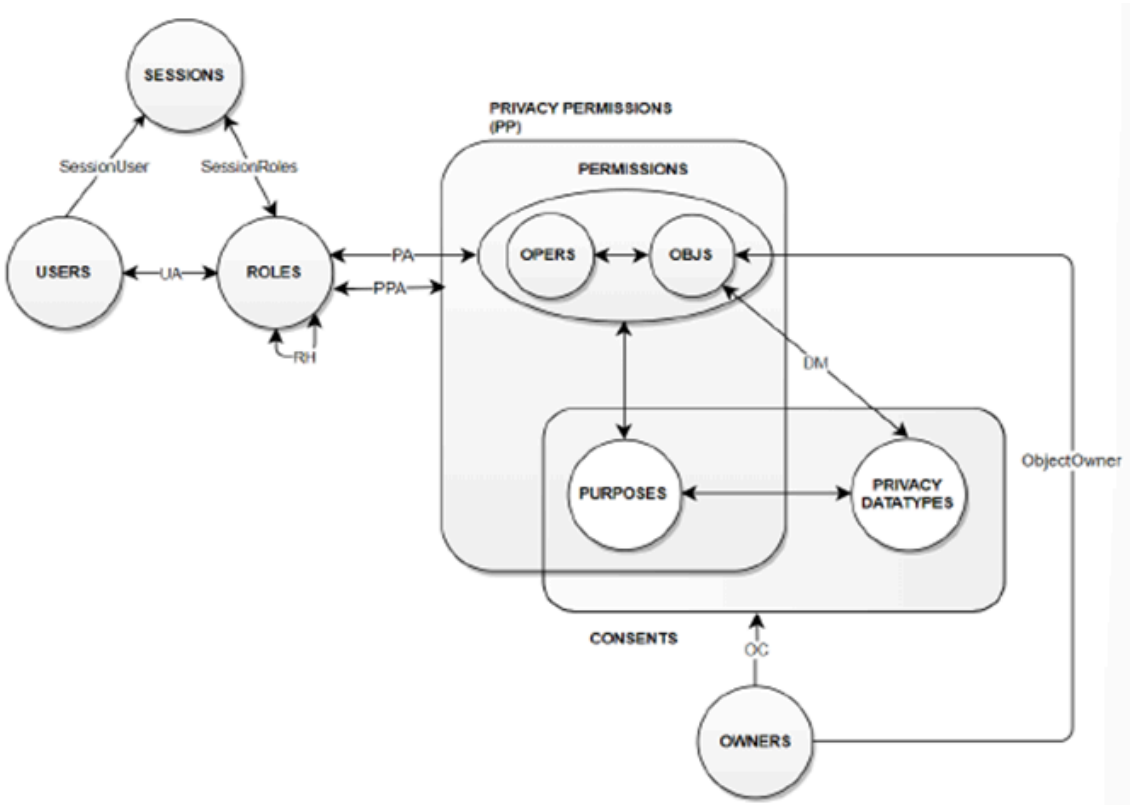

Figura 2: Modelo conceptual de PCA-RBAC [1]

- $User\ Assignment: UA \subseteq USERS\ x\ ROLES.$
  Relación de asignación entre los usuarios y sus roles.
- $Role\ Hierarchy: RH \subseteq ROLES\ x\ ROLES.$
  Relación que define la jerarquía de roles.
- $Permissions: PERMISSIONS \subseteq OPERS\ x\ OBJS.$
  Conjunto de permisos obtenidos al ejecutar una operación sobre un objeto.
- $Permission\ Assignment: PA \subseteq PERMISSIONS\ x\ ROLES.$
  Asignación de permisos a roles.
- $Privacy\ Permissions: PP = \{(prm, prp) \mid prm \in PERMISSIONS, prp \in PURPOSES\}.$
  Permisos de privacidad formados por un permiso y un propósito de uso.
- $Privacy\ Permission\ Assignment: PPA \subseteq PP\ x\ ROLES.$
  Asignación de permisos de privacidad a roles.
- $Data\ Mapping: DM \subseteq OBJS\ x\ PRIVACY\ DATATYPES.$
  Asociación entre objetos y sus correspondientes tipos de privacidad.
- $Consents: CONSENTS = \{(prp, pdt) \mid prp \in PURPOSES, pdt \in PRIVACY\ DATATYPES\}.$ Consentimiento conformado por un propósito y un tipo de privacidad.
- $Owner\ Consent: OC \subseteq OWNERS\ x\ CONSENTS.$
  Asociación entre los titulares de datos y sus consentimientos

En PCA-RBAC existe una restricción que no puede representarse directamente en este modelo conceptual. Esta establece que los objetos que representan información personal deben formar parte de permisos incluidos en permisos de

privacidad y estar asociados a roles mediante la relación PPA. Para capturar esta limitación, se introduce una restricción no estructural:

"Si un objeto tiene un tipo de privacidad asociado, no puede pertenecer a permisos de tipo $PERMISSIONS$ asignados a algún rol mediante PA".

A continuación, se definen las funciones utilizadas para el proceso de autorización:

- $ObjectOwner: OBJS\_PI \rightarrow OWNERS$.
  Asocia un objeto que representa información personal con el identificador de su titular. El conjunto $OBJS\_PI$ contiene los objetos que representan información personal.
- $SessionUser: SESSIONS \rightarrow USERS$.
  Asocia una sesión con su único usuario.
- $SessionRoles: SESSIONS \rightarrow ACTIVE\ ROLES$.
  Mapea una sesión a los roles activos en ella, siendo $ACTIVE\ ROLES$ el conjunto de dichos roles.
- $SessionPermissions: SESSIONS \rightarrow PERMISSIONS\ EXERCISED$.
  Asocia una sesión con los permisos que pueden ejercerse en ella, siendo $PERMISSIONS\ EXERCISED$ el conjunto de estos permisos.
- $SessionPrivacyPermissions: SESSIONS \rightarrow PP\ EXERCISED$.
  Asocia una sesión y los permisos de privacidad que pueden ejercerse en ella, siendo $PP\ EXERCISED$ el conjunto de dichos permisos de privacidad.

Al realizar una solicitud de acceso a los datos se requiere el identificador de la sesión y el permiso solicitado (compuesto por el objeto y la operación). No obstante, para acceder a la información personal, es necesario también el propósito del acceso. Esta solicitud se define como $Access\ Request\ (AR)$, la cual contempla ambos escenarios.

Por otro lado, se define una función para evaluar las solicitudes de acceso (AR) y retornar una decisión afirmativa $(permit)$ o negativa $(deny)$, dependiendo del cumplimiento de determinados requisitos. Esta función, denominada $Access\ Decision\ Function\ (ADF)$, verifica si el usuario que invoca la operación tiene asignado el permiso solicitado en el caso de información no personal. En el caso de información personal, además de verificar el permiso, se evalúa que el propósito solicitado esté asignado al usuario mediante un permiso de privacidad y que exista el consentimiento del titular de los datos para ese propósito.

Estos requisitos se denominarán a lo largo del documento como los requisitos necesarios respecto al propósito y al consentimiento. En el capítulo 4 se estudiará un teorema que valide su cumplimiento.

2.3. Política de privacidad

En una política de privacidad organizacional, es habitual especificar tanto el propósito para el cual se utilizarán los datos como el tipo que estos poseen [20]. Por ello, resulta de utilidad representar matemáticamente esta parte de la política que aborda dichos aspectos.

Para tal fin, se define la política de privacidad como una función que, dado un propósito perteneciente al conjunto $PURPOSES$ y un tipo de datos dentro de $PRIVACY\ DATATYPES$, devuelve un valor que indica la obligatoriedad de que las personas otorguen su consentimiento para esa combinación. Estos valores pueden ser:

- 1, si el consentimiento es obligatorio.
- 0, si es opcional, en cuyo caso la persona puede optar por aceptar o rechazar esa combinación.
- $-\ 1$, si el propósito nunca se aplica a ese tipo de datos.

Formalmente, esto se define como:

$$Policy:\ (PURPOSES\ x\ PRIVACY\ DATATYPES) \rightarrow \{-\ 1,\ 0,\ 1\}$$

Una vez que los propósitos opcionales son aceptados o rechazados, el titular de los datos dispone de un conjunto de consentimientos aceptados, conformados por un propósito y un tipo de datos. Estos consentimientos, junto con los obligatorios, forman el elemento $CONSENTS$ descrito en el modelo, el cual se utiliza para controlar el acceso a la información personal.

2.4. Limitaciones de PCA-RBAC

Este modelo de control de acceso, a pesar de ser más abarcativo que RBAC y algunas extensiones vistas en el capítulo 1, posee algunas limitaciones.

En primer lugar, no se exploran distintas alternativas para modelar condiciones u obligaciones adicionales dentro del esquema de PCA-RBAC. Los consentimientos de los titulares se representan con el único objetivo de reflejar su voluntad, sin considerar si aceptan o no la política de privacidad en su totalidad.

Otra limitación del modelo es que no permite establecer relaciones entre los usuarios del sistema y los titulares de la información. Evaluar la viabilidad de representar dichas relaciones podría ser un área de investigación a futuro, aunque no se aborda en este trabajo.

Asimismo, el modelo no contempla aquellos casos en los que se requiere un propósito de uso, pero no un consentimiento explícito para acceder a la información, dado que la regulación de privacidad en Uruguay exige ambos elementos para el acceso.

Además, los propósitos de uso carecen de una semántica formal definida, ya que se representan únicamente como cadenas de texto (strings) o etiquetas. Aunque esta decisión simplifica el modelo, constituye un obstáculo para asegurar el cumplimiento preciso de las políticas de privacidad relacionadas con los propósitos [21].

Por último, pero no menos importante, es posible que varios permisos de privacidad estén asociados a un mismo propósito y a un mismo rol. Esto provoca un incremento en la cantidad de permisos de privacidad y en las asignaciones correspondientes, lo cual podría complicar la gestión del sistema con el tiempo.

Capítulo 3: Formalización

En este capítulo se presenta la especificación formal del modelo PCA-RBAC introducido en el capítulo anterior. Esta incluye gran parte de la formalización del modelo RBAC desarrollada en la tesina de Cristian Rosa [13], con las extensiones propias de PCA-RBAC, como los propósitos y consentimientos.

Siguiendo un enfoque similar al del trabajo mencionado , se introduce primero la notación formal utilizada, para luego exponer los elementos y relaciones que definen el modelo, el estado del sistema y sus propiedades de validez, así como los comandos, sus posibles errores y su ejecución. Además, se formalizan los conceptos de $Access\ Request$ y $Access\ Definition\ Function$, ya discutidos en el capítulo 2, los cuales son fundamentales para demostrar algunas propiedades de interés posteriormente.

La formalización presentada en este capítulo, basada en el CCI, se detalla de manera rigurosa mediante el asistente de pruebas Coq. El código Coq completo está disponible en https://www.fing.edu.uy/~cluna/pca-rbac-coq.v.

3.1. Notación

La notación utilizada en este informe se inspira principalmente en el proyecto de Cristian Rosa [13], que a su vez se basa en el artículo “A Formal Specification of the MIDP 2.0 Security Model” [22], escrito por Santiago Zanella Béguelin con la colaboración de Gustavo Betarte y Carlos Luna.

Para representar los conectivos lógicos, se emplea su notación convencional
(∧: and, ∨: or, →: implica, ↔: sí y sólo sí, ¬: not, ∃: existe, ∀: para todo).

En cuanto a los tipos de registros (records), se definen como:

$$R := [\, field_0 : A_0, \ldots, field_n : A_n \,]$$

Esto se resume en un tipo inductivo no recursivo con un único constructor denominado $Constructor_R$, y en n funciones de proyección $field_i$: $R \rightarrow A_i$. Cuando el tipo R es evidente, se usa $< a_0, \ldots, a_n >$ en lugar de $Constructor_R\ a_0, \ldots, a_n$. Para las aplicaciones de funciones de producción, se utiliza la notación punto como abreviación (ej. $field_i\ r = r.field_i$).

Para denotar los conjuntos de tipo A, se utiliza $set\ A$, siendo set un tipo inductivo que representa los conjuntos como listas, usando $cons : A \rightarrow set\ A \rightarrow set\ A$ y $nil : set$ como sus constructores (se puede usar :: en lugar de $cons$ y [] en lugar de nil).

Para representar funciones parciales de un tipo A a otro B, se utiliza el tipo $option$, de la forma $f : A \rightarrow option\ B$, cuyos constructores para un tipo T son $Some : T \rightarrow option\ T$ y $None : option\ T$.

En lo que respecta a los operadores de conjuntos, también se utiliza su notación convencional (∪: unión, ∩: intersección, ∈: pertenece, ⊆: incluido).

Por otro lado, los constructores de tipos inductivos representados como:

$$I : t_0, \ldots, t_n := C_i\,(a_0 : k_0), \ldots, (a_j : k_j) : I\,(b_0 : t_0), \ldots, (b_n : t_n) \qquad \forall\, i, j, n \in N_0$$

Esto se puede simplificar de la siguiente manera:

$$C_i \frac{a0, \ldots, aj}{I\ b0, \ldots, bn}$$

De esta forma, las variables libres son universalmente cuantificadas y sus tipos están definidos por el contexto.

Finalmente, se introducen predicados anónimos utilizando la notación lambda, expresado de la forma $(\lambda\, x\,.\ t)$, donde un predicado aplicado a x devuelve $true$ si y sólo si el término t se satisface (la variable x está incluida en el término t).

3.2. Elementos y relaciones

A partir de la notación definida, se lleva a cabo la especificación formal del modelo PCA-RBAC presentado en el capítulo 2.

En primer lugar, se representan los elementos básicos del modelo RBAC, incluyendo sesiones, usuarios, roles, operaciones y objetos, denominados $SESSIONS$, $USERS$, $ROLES$, $OPERS$ y $OBJS$, respectivamente. A estos se les añaden otros tipos básicos específicos de PCA-RBAC, siendo $PRIVACY\_DATATYPES$, $PURPOSES$ y $OWNERS$, los que corresponden a los tipos de privacidad, propósitos y titulares de datos, respectivamente. Todos ellos se representan utilizando el tipo Set.

Luego se tienen los registros provenientes de RBAC. En primer lugar, la asignación entre usuarios y roles representada como:

$$UA := [\, u_{UA} : USERS,\ r_{UA} : ROLES \,]$$

Luego, se definen los permisos formados por operaciones y objetos:

$$PERMISSIONS := [\, op_{PRM} : OPERS,\ ob_{PRM} : OBJS \,]$$

Y, finalmente, la asignación de permisos a roles se expresa de la forma:

$$PA := [\, prm_{PA} : PERMISSIONS,\ r_{PA} : ROLES \,]$$

Se añaden también otros registros relacionados a elementos de PCA-RBAC, que se detallan a continuación.

Por un lado, se tienen los permisos de privacidad, que se forman con permisos y con propósitos de uso:

$$PP :=\ [\, prm_{\mathsf{PP}} :\ PERMISSIONS,\ prp_{\mathsf{PP}} :\ PURPOSES\,]$$

Además, se establece la asignación de permisos de privacidad a roles con:

$$PPA :=\ [\, pp_{\mathsf{PPA}} :\ PP,\ r_{\mathsf{PPA}} :\ ROLES\,]$$

En el caso de los objetos mapeados con tipos de privacidad, se representan como:

$$DM :=\ [\, ob_{\mathsf{DM}} :\ OBJS,\ pdt_{\mathsf{DM}} :\ PRIVACY\_DATATYPES\,]$$

Mientras que los consentimientos conformados por los propósitos y los tipos de privacidad se especifican como:

$$CONSENTS :=\ [\, prp_{\mathsf{CONS}} :\ PURPOSES,\ pdt_{\mathsf{CONS}} :\ PRIVACY\_DATATYPES\,]$$

Finalmente, la asociación entre titulares y consentimientos se representa de la siguiente manera:

$$OC :=\ [\, own_{\mathsf{OC}} :\ OWNERS,\ consent_{\mathsf{OC}} :\ CONSENTS\,]$$

Para representar las funciones utilizadas en el proceso de autorización, primero se deben definir algunos conjuntos pertinentes.

En el caso de la función que asocia un objeto que representa información personal con el identificador de su titular, es necesario representar este tipo de objeto con el tipo:

$$OBJS\_PI :=\ [\, ob_{\mathsf{PI}} :\ OBJS\,]$$

Para la función que asocia una sesión con los roles activos en ella, primero se expresan dichos roles con el tipo:

$$ACTIVE\_ROLES :=\ [\, r_{\mathsf{AR}} :\ set\ ROLES\,]$$

En el caso de la función que asocia una sesión con los permisos que pueden ejercerse en ella, se expresan estos permisos de la siguiente manera:

$$PERMISSIONS\_EXERCISED :=\ [\, prm_{\mathsf{EX}} :\ set\ PERMISSIONS\,]$$

Luego, se expresa la función que asocia una sesión con los permisos de privacidad que pueden ejercerse en ella, representando esos permisos como:

$$PP\_EXERCISED :=\ [\, pp_{\mathsf{EX}} :\ set\ PP\,]$$

Para poder representar las funciones mencionadas, así como la restricción no estructural de PCA-RBAC y otras propiedades relevantes, se debe definir el estado del sistema, el cual se abordará en la siguiente sección.

3.3. Estado del sistema

3.3.1. Definición

Con el fin de poder especificar PCA-RBAC de la mejor manera posible y definir algunas propiedades vinculadas a este modelo, se establece el estado global del sistema que contiene todos sus elementos, utilizando el siguiente tipo de registro:

$$
\begin{aligned}
State := [\ & u_S : set\ USERS, \\
& r_S : set\ ROLES, \\
& ua_S : set\ UA, \\
& \ll_S : ROLES \rightarrow ROLES \rightarrow Prop, \\
& \preccurlyeq_S : ROLES \rightarrow ROLES \rightarrow Prop, \\
& op_S : set\ OPERS, \\
& ob_S : set\ OBJS, \\
& prm_S : set\ PERMISSIONS, \\
& pa_S : set\ PA, \\
& prp_S : set\ PURPOSES, \\
& pp_S : set\ PP, \\
& ppa_S : set\ PPA, \\
& pdt_S : set\ PRIVACY\_DATATYPES, \\
& dm_S : set\ DM, \\
& consent_S : set\ CONSENTS, \\
& own_S : set\ OWNERS, \\
& oc_S : set\ OC, \\
& objectOwner_S : OBJS\_PI \rightarrow option\ OWNERS, \\
& sid_S : set\ SESSIONS, \\
& sessionUser_S : SESSIONS \rightarrow option\ USERS, \\
& sessionRoles_S : SESSIONS \rightarrow option\ ACTIVE\_ROLES, \\
& sessionPermissions_S : SESSIONS \rightarrow option\ PERMISSIONS\_EXERCISED, \\
& sessionPrivacyPermissions_S : SESSIONS \rightarrow option\ PP\_EXERCISED\ ]
\end{aligned}
$$

En esta representación del estado, u_S es el conjunto todos los usuarios válidos, r_S es el conjunto de los roles, ua_S es el conjunto de asignaciones usuario-rol, op_S es el conjunto de operaciones, ob_S es el conjunto de objetos sobre los cuales se pueden realizar estas operaciones, prm_S es el conjunto de permisos, pa_S es el conjunto de asignaciones permiso-rol, prp_S es el conjunto de propósitos de uso, pp_S es el conjunto de permisos de privacidad, ppa_S es el conjunto de asignaciones permiso de privacidad-rol, pdt_S es el conjunto de tipos de privacidad que representan información personal, dm_S es el conjunto de asignaciones objeto-tipo de privacidad, $consent_S$ es el conjunto de consentimientos otorgados por el titular de los datos, own_S es el conjunto de dichos titulares de datos, oc_S es el conjunto de asignaciones titular-consentimiento, y sid_S es el conjunto de los identificadores de sesión de las sesiones establecidas en el sistema.

Asimismo, se define la jerarquía de roles de manera análoga a lo realizado para el modelo RBAC, donde $\ll_S$ representa la relación de herencia inmediata entre roles, y $\preccurlyeq_S$ denota la relación de clausura reflexo-transitiva entre roles.

Para representar las funciones que se utilizan para el proceso de autorización, se utiliza el tipo $option$, definiéndose como funciones parciales.

La función $objectOwner_S$ asocia un objeto que representa información personal con el identificador de su titular; $sessionUser_S$ establece la relación entre una sesión y su único usuario; $sessionRoles_S$ vincula una sesión y sus roles activos; $sessionPermissions_S$ relaciona una sesión con sus permisos ejercidos; y $sessionPrivacyPermissions_S$ conecta una sesión y sus permisos de privacidad ejercidos.

3.3.2. Algunas propiedades

A continuación, se definen algunas propiedades relevantes que podrán ser utilizadas como hipótesis para demostrar otras proposiciones en el siguiente capítulo.

Dadas las dos relaciones provenientes de la jerarquía de roles $\ll_S$ y $\preccurlyeq_S$, se definen las siguientes propiedades para todo rol r, r' y r'' pertenecientes a $ROLES$:

$$rt\_refl\ \frac{}{r \preccurlyeq r} \qquad rt\_step\ \frac{r \ll r'}{r \preccurlyeq r'} \qquad rt\_symm\ \frac{r \preccurlyeq r' \quad r' \preccurlyeq r}{r = r'} \qquad rt\_trans\ \frac{r \preccurlyeq r' \quad r' \preccurlyeq r''}{r \preccurlyeq r''}$$

Para representar la restricción no estructural de PCA-RBAC, se define la siguiente propiedad:

$$\begin{aligned} nonStructuralRestriction :=\ & \forall\ (s \in State)\ (ob \in OBJS),\ \exists\ (pdt \in PRIVACY\_DATATYPES), \\ & ob \in s.ob_S \wedge pdt \in s.pdt_S \wedge (ob,\ pdt) \in DM\ \rightarrow \\ & \neg\ (\exists\ (op \in OPERS) \wedge \exists\ (r \in ROLES),\ op \in s.op_S \wedge r \in s.r_S \wedge ((op,\ ob),\ r) \in PA) \end{aligned}$$

Dados los objetos que representan información personal, se establece la siguiente afirmación:

$$\begin{aligned} obAndPdtInDM :=\ & \forall\ (s \in State)\ (ob \in OBJS),\ \exists\ (ob\_pi \in OBJS\_PI), \\ & ob \in ob\_pi\ \rightarrow\ \exists\ (pdt \in PRIVACY\ DATATYPES),\ ob \in s.ob_S \wedge pdt \in s.pdt_S \wedge (ob,\ pdt) \in DM \end{aligned}$$

Los roles activos en una sesión cumplen la siguiente propiedad:

$$\begin{aligned} mapSessionActiveRole :=\ & \forall\ (s \in State)\ (sid \in SESSIONS)\ (r \in ROLES),\ \exists\ (ar \in ACTIVE\_ROLES), \\ & r \in ar\ \rightarrow\ sessionUser_S\ (sid) \in s.u_S \wedge r \in s.r_S \wedge (sessionUser_S\ (sid),\ r) \in UA \end{aligned}$$

En el caso de los permisos que pueden ser ejercidos en una sesión, se establece lo siguiente:

$$\begin{aligned} prmExercisedInSession :=\ & \forall\ (s \in State)\ (sid \in SESSIONS)\ (prm \in PERMISSIONS), \\ & \exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\ prm \in prm\_ex\ \rightarrow \\ & \exists\ (r\ r' \in ROLES)\ (ar \in ACTIVE\_ROLES), \\ & prm \in s.prm_S \wedge r' \in s.r_S \wedge (prm,\ r') \in PA \wedge r \in s.r_S \wedge r \in ar \wedge ar \in sessionRoles_S\ (sid) \wedge r\ \ll_S r' \end{aligned}$$

Finalmente, los permisos de privacidad que pueden ser ejercidos en una sesión cumplen la siguiente propiedad:

$$ppExercisedInSession := \forall\ (s \in State)\ (sid \in SESSIONS)\ (prm \in PERMISSIONS)\ (prp \in PURPOSES),$$
$$\exists\ (pp\_ex \in PP\_EXERCISED),\ prm \in s.prm_S \wedge prp \in s.prp_S \wedge (prm,\ prp) \in PP \wedge (prm,\ prp) \in pp\_ex\ \rightarrow$$
$$\exists\ (r\ r' \in ROLES)\ (ar \in ACTIVE\_ROLES),$$
$$r' \in s.r_S \wedge ((prm,\ prp),\ r') \in PPA \wedge r \in s.r_S \wedge r \in ar \wedge ar \in sessionRoles_S\ (sid) \wedge r\ \ll_S r'$$

3.3.3. Observadores del estado

Se definen los observadores del estado como predicados que se utilizan para abstraer otros de mayor complejidad, con el objetivo de mejorar su legibilidad y comprensión.

Los predicados definidos son los siguientes:

- $user\_roles : USERS \rightarrow set\ ROLES$
- $role\_users : ROLES \rightarrow set\ USERS$
- $role\_of\_user : State \rightarrow ROLES \rightarrow USERS \rightarrow Prop$
- $authorizedUser : State \rightarrow USERS \rightarrow ROLES \rightarrow Prop$
- $r\_is\_active\_role : State \rightarrow SESSIONS \rightarrow USERS \rightarrow ROLES \rightarrow Prop$

$user\_roles$ devuelve la lista de roles asignados a un usuario en el sistema:

$$user\_roles\ u := foldl\ (\lambda\ rl\ ua\ .\ if\ ua.u_{UA} = u\ then\ ua.r_{UA} :: rl\ else\ rl)\ s.ua_S\ []$$

$role\_users$ devuelve la lista de usuarios asignados a un rol en el sistema:

$$role\_users\ r := foldl\ (\lambda\ ul\ ua\ .\ if\ ua.r_{UA} = r\ then\ ua.u_{UA} :: ul\ else\ ul)\ s.ua_S\ []$$

$role\_of\_user$ relaciona a los usuarios y sus roles asignados en el sistema:

$$role\_of\_user\ s\ r\ u := (u,\ r) \in UA\ \wedge\ (u,\ r) \in s.ua_S$$

$authorizedUser$ establece que un usuario del sistema está autorizado para un rol si está asignado directamente a este, o si lo está a un rol que hereda del mismo:

$$authorizedUser\ s\ u\ r := \exists\ (r' \in ROLES),\ (role\_of\_user\ s\ r'\ u)\ \wedge\ (r\ \preccurlyeq_S r')$$

$r\_is\_active\_role$ establece que un rol es activo si está asociado a una sesión y a un usuario que esté autorizado para él:

$$r\_is\_active\_role\ s\ sid\ u\ r := \exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \rightarrow\ authorizedUser\ s\ u\ r$$

3.3.4. Propiedades de validez de estado

Debido a que no cualquier elemento perteneciente al estado del sistema es un estado válido, es fundamental definir ciertas restricciones que las componentes del estado deben poder cumplir para satisfacer la validez. A continuación, se definen las propiedades de validez de estado a partir de un estado s.

“Toda sesión tiene un dueño que además es miembro del conjunto de usuarios del sistema."

$$existsSessionOwner\ s := \ \forall\ (sid \in SESSIONS),\\ sid \in s.sid_S \rightarrow \exists\ (u \in USERS),\ u \in s.u_S\ \wedge\ u \in sessionUser_S\ (sid)$$

“El usuario dueño de una sesión es único."

$$uniqueSessionOwner\ s := \ \forall\ (sid \in SESSIONS)\ (u\ u' \in USERS),\\ sid \in s.sid_S\ \wedge\ u \in sessionUser_S\ (sid)\ \wedge\ u' \in sessionUser_S\ (sid)\ \rightarrow\ u\ =\ u'$$

“Los roles activos de una sesión son un subconjunto del conjunto de roles para los que el usuario dueño de dicha sesión está autorizado."

$$activeSessionRoles\ s := \ \forall\ (sid \in Session)\ (u \in USERS)\ (r \in ROLES),\ \exists\ (ar \in ACTIVE\_ROLES),\\ sid \in s.sid_S\ \wedge\ sessionUser_S\ (sid)\ \wedge\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \rightarrow\ authorized\_user\ s\ u\ r$$

“La relación entre roles ($\preccurlyeq_S$) que modela la jerarquía de roles es un orden parcial."

$$isOrder\ s := \ \forall\ (r\ r'\ r'' \in ROLES),\\ (r\ \preccurlyeq_S r)\ \wedge\\ (r\ \preccurlyeq_S r'\ \wedge\ r'\ \preccurlyeq_S r\ \rightarrow\ r\ =\ r')\ \wedge\\ (r\ \preccurlyeq_S r'\ \wedge\ r'\ \preccurlyeq\ r''\ \rightarrow\ r\ \preccurlyeq_S r'')$$

“Los usuarios y roles relacionados a través de la asignación usuario-rol son un subconjunto de los conjuntos de usuarios y roles del sistema respectivamente."

$$UA\_integrity\ s := \ \forall\ (u \in USERS)\ (r \in ROLES),\\ (u,\ r) \in s.ua_S\ \wedge\ ((u,\ r) \in UA\ \leftrightarrow\ u \in s.u_S\ \wedge\ r \in s.r_S)$$

“Los roles pertenecientes a la jerarquía de roles son un subconjunto del conjunto de roles del sistema."

$$H\_integrity\ s := \ \forall\ (r\ r' \in ROLES),\ r\ \ll_S r'\ \leftrightarrow\ r \in s.r_S\ \wedge\ r' \in s.r_S$$

“Los objetos y operaciones relacionados a través del conjunto de permisos son un subconjunto de los conjuntos de objetos y operaciones del sistema respectivamente."

$$Prm\_integrity\ s := \ \forall\ (op \in OPERS)\ (ob \in OBJS),\\ (op,\ ob) \in s.prm_S\ \wedge\ ((op,\ ob) \in PERMISSIONS\ \leftrightarrow\ op \in s.op_S\ \wedge\ ob \in s.ob_S)$$

“Los permisos y roles relacionados a través de la asignación rol-permiso son un subconjunto de los conjuntos de permisos y roles del sistema respectivamente."

$$PA\_integrity\ s := \ \forall\ (prm \in PERMISSIONS)\ (r \in ROLES),$$
$$(prm,\ r) \in s.pa_{\mathrm{S}}\ \wedge\ ((prm,\ r) \in PA\ \leftrightarrow\ prm \in s.prm_{\mathrm{S}}\ \wedge\ r \in s.r_{\mathrm{S}})$$

“Los permisos y propósitos relacionados a través del conjunto de permisos de privacidad son un subconjunto de los conjuntos de permisos y propósitos del sistema respectivamente."

$$PP\_integrity\ s := \ \forall\ (prm \in PERMISSIONS)\ (prp \in PURPOSES),$$
$$(prm,\ prp) \in s.pp_{\mathrm{S}}\ \wedge\ ((prm,\ prp) \in PP\ \leftrightarrow\ prm \in s.prm_{\mathrm{S}}\ \wedge\ prp \in s.prp_{\mathrm{S}})$$

“Los permisos de privacidad y roles relacionados a través de la asignación rol-permiso de privacidad son un subconjunto de los conjuntos de permisos de privacidad y roles del sistema respectivamente."

$$PPA\_integrity\ s := \ \forall\ (pp \in PP)\ (r \in ROLES),$$
$$(pp,\ r) \in s.ppa_{\mathrm{S}}\ \wedge\ ((pp,\ r) \in PPA\ \leftrightarrow\ pp \in s.pp_{\mathrm{S}}\ \wedge\ r \in s.r_{\mathrm{S}})$$

“Los objetos y tipos de privacidad relacionados a través de la asignación objeto-tipo de privacidad son un subconjunto de los conjuntos de objetos y tipos de privacidad del sistema respectivamente."

$$DM\_integrity\ s := \ \forall\ (ob \in OBJS)\ (pdt \in PRIVACY\_DATATYPES),$$
$$(ob,\ pdt) \in s.dm_{\mathrm{S}}\ \wedge\ ((ob,\ pdt) \in DM\ \leftrightarrow\ ob \in s.ob_{\mathrm{S}}\ \wedge\ pdt \in s.pdt_{\mathrm{S}})$$

“Los propósitos y tipos de privacidad relacionados a través del conjunto de consentimientos son un subconjunto de los conjuntos de propósitos y tipos de privacidad del sistema respectivamente."

$$Consent\_integrity\ s := \ \forall\ (prp \in PURPOSES)\ (pdt \in PRIVACY\_DATATYPES),$$
$$(prp,\ pdt) \in s.consent_{\mathrm{S}}\ \wedge\ ((prp,\ pdt) \in CONSENTS\ \leftrightarrow\ prp \in s.prp_{\mathrm{S}}\ \wedge\ pdt \in s.pdt_{\mathrm{S}})$$

“Los titulares de datos y consentimientos relacionados a través de la asignación titular-consentimiento son un subconjunto de los conjuntos de titulares de datos y consentimientos del sistema respectivamente."

$$OC\_integrity\ s := \ \forall\ (own \in OWNERS)\ (consent \in CONSENTS),$$
$$(own,\ consent) \in s.oc_{\mathrm{S}}\ \wedge\ ((own,\ consent) \in OC\ \leftrightarrow\ own \in s.own_{\mathrm{S}}\ \wedge\ consent \in s.consent_{\mathrm{S}})$$

“Toda operación conforma un permiso junto a un objeto, el cual también se mapea a un tipo de privacidad.”

$$existsObInPermissionAndDM\ s := \ \forall\ (op \in OPERS)\ (pdt \in PRIVACY\_DATATYPES),$$
$$op \in s.op_{\mathrm{S}}\ \wedge\ pdt \in s.pdt_{\mathrm{S}}\ \rightarrow$$
$$\exists\ (ob \in OBJS),\ ob \in s.ob_{\mathrm{S}}\ \wedge\ (op,\ ob) \in PERMISSIONS\ \wedge\ (ob,\ pdt) \in DM$$

“Todo permiso conforma un permiso de privacidad junto a un propósito, el cual también conforma un consentimiento junto a un tipo de privacidad.”

$$existsPrpInPPAndConsent\ s := \ \forall\ (prm \in PERMISSIONS)\ (pdt \in PRIVACY\_DATATYPES),$$
$$prm \in s.prm_{\mathrm{S}}\ \wedge\ pdt \in s.pdt_{\mathrm{S}}\ \rightarrow$$
$$\exists\ (prp \in PURPOSES),\ prp \in s.prp_{\mathrm{S}}\ \wedge\ (prm,\ prp) \in PP\ \wedge\ (prp,\ pdt) \in CONSENTS$$

“Todo propósito conforma un consentimiento junto a un tipo de privacidad, el cual también se mapea a un objeto.”

$$existsPdtInConsentAndDM\ s := \ \forall\ (prp \in PURPOSES)\ (ob \in OBJS),$$
$$prp \in s.prp_{\mathrm{S}}\ \wedge\ ob \in s.ob_{\mathrm{S}}\ \rightarrow$$
$$\exists\ (pdt \in PRIVACY\_DATATYPES),\ pdt \in s.pdt_{\mathrm{S}}\ \wedge\ (prp,\ pdt) \in CONSENTS\ \wedge\ (ob,\ pdt) \in DM$$

“Todo consentimiento tiene un dueño, el cual también es miembro del conjunto de titulares de datos del sistema.”

$$existsConsentOwner\ s := \ \forall\ (consent \in CONSENTS),$$
$$consent \in s.consent_{\mathrm{S}}\ \rightarrow\ \exists\ (own \in OWNERS),\ own \in s.own_{\mathrm{S}}\ \wedge\ (own,\ consent) \in OC$$

“Todo objeto que representa información personal tiene un dueño, el cual también es miembro del conjunto de titulares de datos del sistema.”

$$existsObjectOwner\ s := \ \forall\ (ob \in OBJS)\ (ob\_pi \in OBJS\_PI),$$
$$ob \in s.ob_{\mathrm{S}}\ \wedge\ ob \in ob\_pi\ \rightarrow\ \exists\ (own \in OWNERS),\ own \in s.own_{\mathrm{S}}\ \wedge\ own \in objectOwner_{\mathrm{S}}\ (ob\_pi)$$

“El titular de un objeto que representa información personal es único.”

$$uniqueObjectOwner\ s := \ \forall\ (ob \in OBJS)\ (ob\_pi \in OBJS\_PI)\ (own\ own' \in OWNERS),$$
$$ob \in s.ob_{\mathrm{S}}\ \wedge\ ob \in ob\_pi\ \wedge\ own \in objectOwner_{\mathrm{S}}\ (ob\_pi)\ \wedge\ own' \in objectOwner_{\mathrm{S}}\ (ob\_pi)\ \rightarrow$$
$$own\ =\ own'$$

“Existe un objeto perteneciente a los objetos del sistema el cual representa información personal y a su vez tiene un dueño.”

$$Valid\_objectOwner\ s := \ \forall\ (ob \in OBJS),\ \exists\ (ob\_pi \in OBJS\_PI)\ (own \in OWNERS),$$
$$ob \in ob\_pi\ \wedge\ own \in s.own_{\mathrm{S}}\ \wedge\ own \in objectOwner_{\mathrm{S}}\ (ob\_pi)$$

Finalmente, para que un estado sea válido se deben cumplir todas las propiedades descritas anteriormente. Esto se representa definiendo la propiedad $Valid$ para un estado s de la siguiente forma:

$$
\begin{aligned}
Valid\ s := \ & existsSessionOwner\ s\ \wedge \\
& uniqueSessionOwner\ s\ \wedge \\
& activeSessionRoles\ s\ \wedge \\
& isOrder\ s\ \wedge \\
& UA\_integrity\ s\ \wedge \\
& H\_integrity\ s\ \wedge \\
& Prm\_integrity\ s\ \wedge \\
& PA\_integrity\ s\ \wedge \\
& PP\_integrity\ s\ \wedge \\
& PPA\_integrity\ s\ \wedge \\
& DM\_integrity\ s\ \wedge \\
& Consent\_integrity\ s\ \wedge \\
& OC\_integrity\ s\ \wedge \\
& existsObInPermissionAndDM\ s\ \wedge \\
& existsPrpInPPAndConsent\ s\ \wedge \\
& existsPdtInConsentAndDM\ s\ \wedge \\
& existsConsentOwner\ s\ \wedge \\
& existsObjectOwner\ s\ \wedge \\
& uniqueObjectOwner\ s\ \wedge \\
& Valid\_objectOwner\ s
\end{aligned}
$$

3.4. Comandos

Con el objetivo de interactuar con el sistema y generar acciones que modifiquen un estado válido, se definieron comandos para cumplir con este propósito. A continuación, se presenta la firma de cada uno, las pre y postcondiciones que deben cumplirse, así como las posibles respuestas y mensajes de error que pueden generarse. También se formaliza la ejecución de los comandos del sistema.

3.4.1. Sintaxis

Los comandos se dividen en tres tipos, dependiendo del tipo de acción que realizan sobre el sistema.

- Comandos administrativos: Agregan o eliminan elementos básicos del sistema.
- Funciones del sistema: Pueden añadir o quitar elementos del sistema, así como realizar otro tipo de acciones.
- Funciones de revisión: Devuelven listas de elementos que cumplen con ciertas características.

Para representar estos comandos, se utiliza el conjunto inductivo $Funcs$.

Comandos administrativos:

- $AddUser : USERS \rightarrow Funcs$
- $DeleteUser : USERS \rightarrow Funcs$
- $AddRole : ROLES \rightarrow Funcs$
- $DeleteRole : ROLES \rightarrow Funcs$
- $AssignUser : USERS \rightarrow ROLES \rightarrow Funcs$
- $DeassignUser : USERS \rightarrow ROLES \rightarrow Funcs$
- $AddOperation : OPERS \rightarrow Funcs$
- $DeleteOperation : OPERS \rightarrow Funcs$
- $AddObject : OBJS \rightarrow Funcs$
- $DeleteObject : OBJS \rightarrow Funcs$
- $AddPermission : OPERS \rightarrow OBJS \rightarrow Funcs$
- $DeletePermission : OPERS \rightarrow OBJS \rightarrow Funcs$
- $GrantPermission : OPERS \rightarrow OBJS \rightarrow ROLES \rightarrow Funcs$
- $RevokePermission : OPERS \rightarrow OBJS \rightarrow ROLES \rightarrow Funcs$
- $AddInheritance : ROLES \rightarrow ROLES \rightarrow Funcs$
- $DeleteInheritance : ROLES \rightarrow ROLES \rightarrow Funcs$
- $AddAscendant : ROLES \rightarrow ROLES \rightarrow Funcs$
- $AddDescendant : ROLES \rightarrow ROLES \rightarrow Funcs$
- $AddPurpose : PURPOSES \rightarrow Funcs$
- $DeletePurpose : PURPOSES \rightarrow Funcs$
- $AddPrivacyPermission : PERMISSIONS \rightarrow PURPOSES \rightarrow Funcs$
- $DeletePrivacyPermission : PERMISSIONS \rightarrow PURPOSES \rightarrow Funcs$
- $GrantPrivacyPermission : PERMISSIONS \rightarrow PURPOSES \rightarrow ROLES \rightarrow Funcs$
- $RevokePrivacyPermission : PERMISSIONS \rightarrow PURPOSES \rightarrow ROLES \rightarrow Funcs$
- $AddPrivacyDataType : PRIVACY\_DATATYPES \rightarrow Funcs$
- $DeletePrivacyDataType : PRIVACY\_DATATYPES \rightarrow Funcs$
- $AddDataMapping : OBJS \rightarrow PRIVACY\_DATATYPES \rightarrow Funcs$
- $DeleteDataMapping : OBJS \rightarrow PRIVACY\_DATATYPES \rightarrow Funcs$
- $GrantConsent : PURPOSES \rightarrow PRIVACY\_DATATYPES \rightarrow Funcs$
- $RevokeConsent : PURPOSES \rightarrow PRIVACY\_DATATYPES \rightarrow Funcs$
- $AddOwner : OWNERS \rightarrow Funcs$
- $DeleteOwner : OWNERS \rightarrow Funcs$
- $AddOwnerConsent : OWNERS \rightarrow CONSENTS \rightarrow Funcs$
- $DeleteOwnerConsent : OWNERS \rightarrow CONSENTS \rightarrow Funcs$

Funciones del sistema:

- $AddObjectPersonalInformation : OBJS \rightarrow PRIVACY\_DATATYPES \rightarrow OWNERS \rightarrow Funcs$
- $DropObjectPersonalInformation : OBJS \rightarrow PRIVACY\_DATATYPES \rightarrow OWNERS \rightarrow Funcs$
- $CreateSession : USERS \rightarrow set\ ROLES \rightarrow SESSIONS \rightarrow Funcs$
- $DeleteSession : USERS \rightarrow SESSIONS \rightarrow Funcs$
- $AddActiveRole : USERS \rightarrow SESSIONS \rightarrow ROLES \rightarrow Funcs$
- $DropActiveRole : USERS \rightarrow SESSIONS \rightarrow ROLES \rightarrow Funcs$
- $AddPermissionExercised : SESSIONS \rightarrow ROLES \rightarrow PERMISSIONS \rightarrow Funcs$
- $DropPermissionExercised : SESSIONS \rightarrow ROLES \rightarrow PERMISSIONS \rightarrow Funcs$
- $AddPrivacyPermissionExercised : SESSIONS \rightarrow ROLES \rightarrow PP \rightarrow Funcs$
- $DropPrivacyPermissionExercised : SESSIONS \rightarrow ROLES \rightarrow PP \rightarrow Funcs$
- $CheckAccess : SESSIONS \rightarrow OPERS \rightarrow OBJS \rightarrow Funcs$

Funciones de revisión:

- $AssignedUsers : ROLES \rightarrow Funcs$
- $AssignedRoles : USERS \rightarrow Funcs$
- $AuthorizedUsers : ROLES \rightarrow Funcs$
- $AuthorizedRoles : USERS \rightarrow Funcs$

3.4.2. Precondiciones

Una vez definidos los comandos, se deben especificar tanto sus pre como sus postcondiciones. A continuación, se describen las precondiciones para cada comando, que son predicados sobre el estado que deben satisfacerse para que el comando se pueda ejecutar. Estos se representan mediante el conjunto inductivo Pre.

Comandos administrativos:

El comando $(AddUser\ u)$ tiene como objetivo crear un nuevo usuario en el sistema. Para lograr este cometido, este usuario no debe existir en el sistema.

$$Pre\ s\ (AddUser\ u) :=\ \neg\ (u \in s.u_S)$$

El comando $(DeleteUser\ u)$ busca eliminar un usuario del sistema. Para ello, este usuario debe existir en el sistema.

$$Pre\ s\ (DeleteUser\ u) :=\ u \in s.u_S$$

Con $(AddRole\ r)$ se crea un nuevo rol en el sistema. Para que esto ocurra, este rol no debe existir previamente.

$$Pre\ s\ (AddRole\ r) :=\ \neg\ (r \in s.r_S)$$

El comando $(DeleteRole\ r)$ elimina un rol del sistema. Para lograrlo, es necesario que el rol exista.

$$Pre\ s\ (DeleteRole\ r) :=\ r \in s.r_S$$

El comando $(AssignUser\ u\ r)$ asigna el rol r al usuario u. Tanto el usuario como el rol deben ser válidos y no estar asignados previamente.

$$Pre\ s\ (AssignUser\ u\ r) :=\ u \in s.u_S\ \wedge\ r \in s.r_S\ \wedge\ \neg\ (role\_of\_user\ s\ r\ u)$$

Por otro lado, $(DeassignUser\ u\ r)$ desasigna al usuario u del rol r. En este caso, ambos deben ser válidos y deben estar previamente asignados.

$$Pre\ s\ (DeassignUser\ u\ r) :=\ u \in s.u_S\ \wedge\ r \in s.r_S\ \wedge\ (role\_of\_user\ s\ r\ u)$$

El comando $(AddOperation\ op)$ agrega una nueva operación al sistema. Para lograrlo, esta operación no debe existir con anterioridad.

$$Pre\ s\ (AddOperation\ op) :=\ \neg\ (op \in s.op_S)$$

En el caso de $(DeleteOperation\ op)$, se elimina una operación del sistema. Para esto se requiere que op exista.

$$Pre\ s\ (DeleteOperation\ op) :=\ op \in s.op_S$$

El comando $(AddObject\ ob)$ busca agregar un objeto al sistema. Esto requiere que el objeto en cuestión no exista previamente.

$$Pre\ s\ (AddObject\ ob) := \neg\ (ob \in s.ob_S)$$

$(DeleteObject\ ob)$ elimina un objeto del sistema. Se requiere que ob exista.

$$Pre\ s\ (DeleteObject\ ob) :=\ ob \in s.ob_S$$

El comando $(AddPermission\ op\ ob)$ busca agregar un nuevo permiso al sistema, a partir de una operación op y un objeto ob. Para conseguirlo, tanto op como ob deben existir en el sistema y no pueden estar previamente asociados a un permiso.

$$Pre\ s\ (AddPermission\ op\ ob) :=$$
$$op \in s.op_S\ \wedge\ ob \in s.ob_S\ \wedge\ \neg\ ((op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_S)$$

En el caso de $(DeletePermission\ op\ ob)$, se quiere eliminar un permiso del sistema. Para que esto ocurra, la operación op y el objeto ob que conforman el permiso deben existir, al igual que dicho permiso.

$$Pre\ s\ (DeletePermission\ op\ ob) :=$$
$$op \in s.op_S\ \wedge\ ob \in s.ob_S\ \wedge\ (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_S$$

Con $(GrantPermission\ op\ ob\ r)$ se busca asignar el permiso formado por la operación op y el objeto ob al rol r. Para que esto funcione, $(op,\ ob)$ debe ser un permiso válido y r debe existir. Además, el permiso no debe estar asignado previamente al rol.

$$Pre\ s\ (GrantPermission\ op\ ob\ r) :=$$
$$(op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_S\ \wedge$$
$$r \in s.r_S\ \wedge$$
$$\neg\ (((op,\ ob),\ r) \in PA\ \wedge\ ((op,\ ob),\ r) \in s.pa_S)$$

El comando $(RevokePermision\ op\ ob\ r)$, en cambio, desasigna al permiso formado por la operación op y el objeto ob del rol r. Para conseguirlo $(op,\ ob)$ tiene que ser un permiso válido, r debe existir y el permiso debe estar asignado previamente al rol.

$$Pre\ s\ (RevokePermission\ op\ ob\ r) :=$$
$$(op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_S\ \wedge$$
$$r \in s.r_S\ \wedge$$
$$((op,\ ob),\ r) \in PA\ \wedge\ ((op,\ ob),\ r) \in s.pa_S$$

El comando $(AddInheritance\ r_{Asc}\ r_{Desc})$ tiene como objetivo agregar una nueva herencia inmediata de roles $r_{Desc} \ll r_{Asc}$. Para ello, tanto r_{Asc} como r_{Desc} deben existir en el sistema, la herencia no puede existir previamente y la jerarquía debe continuar siendo un orden parcial al agregar la herencia. Esto implica conservar las propiedades de reflexividad, transitividad y antisimetría. Dado que la jerarquía de roles está definida como la clausura reflexo-transitiva de la relación de herencia inmediata, las primeras dos propiedades se cumplen trivialmente. Para la

antisimetría, basta con verificar que no se generen ciclos tras agregar la nueva herencia, lo que se representa como $\neg\,(r_{\text{Asc}} \preccurlyeq r_{\text{Desc}})$.

$$Pre\ s\ (AddInheritance\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ r_{\text{Asc}} \in s.r_{\text{S}}\ \wedge\ r_{\text{Desc}} \in s.r_{\text{S}}\ \wedge\ \neg\,(r_{\text{Desc}}\ \ll_{\text{S}} r_{\text{Asc}})\ \wedge\ \neg\,(r_{\text{Asc}}\ \preccurlyeq_{\text{S}} r_{\text{Desc}})$$

En el caso de $(DeleteInheritance\ r_{\text{Asc}}\ r_{\text{Desc}})$, se busca eliminar la relación de herencia inmediata $r_{\text{Desc}} \ll r_{\text{Asc}}$. Es necesario que tanto r_{Asc} como r_{Desc} existan en el sistema y que éstos estén vinculados por la relación de herencia $\ll$.

$$Pre\ s\ (DeleteInheritance\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ r_{\text{Asc}} \in s.r_{\text{S}}\ \wedge\ r_{\text{Desc}} \in s.r_{\text{S}}\ \wedge\ r_{\text{Desc}}\ \ll_{\text{S}} r_{\text{Asc}}$$

La operación $(AddAscendant\ r_{\text{Asc}}\ r_{\text{Desc}})$ agrega la herencia inmediata $r_{\text{Desc}} \ll r_{\text{Asc}}$ a la jerarquía de roles, creando el rol r_{Asc} en el sistema. Para conseguirlo, el rol r_{Asc} no puede existir en el sistema, pero el rol r_{Desc} debe existir. A modo de observación, se cumple que $r_{\text{Desc}} \ll r_{\text{Asc}}$ no pertenece al sistema, dado que el estado s válido. En caso contrario, por la propiedad $H\_integrity$, se tendría que cumplir $r_{\text{Asc}} \in s.r_{\text{S}}$, lo que contradice la primera precondición.

$$Pre\ s\ (AddAscendant\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ \neg\,(r_{\text{Asc}} \in s.r_{\text{S}})\ \wedge\ r_{\text{Desc}} \in s.r_{\text{S}}$$

La operación $(AddDescendant\ r_{\text{Asc}}\ r_{\text{Desc}})$ también agrega la herencia inmediata $r_{\text{Desc}} \ll r_{\text{Asc}}$ en la jerarquía de roles, pero esta vez crea el rol r_{Desc} en el sistema. Para ello, el rol r_{Desc} no puede existir, mientras que r_{Asc} debe existir.
En este caso también se cumple la observación anterior.

$$Pre\ s\ (AddDescendant\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ r_{\text{Asc}} \in s.r_{\text{S}}\ \wedge\ \neg\,(r_{\text{Desc}} \in s.r_{\text{S}})$$

El comando $(AddPurpose\ prp)$ tiene como objetivo agregar un propósito al sistema. Para poder hacerlo, este no puede existir previamente.

$$Pre\ s\ (AddPurpose\ prp) :=\ \neg\,(prp \in s.prp_{\text{S}})$$

En el caso de $(DeletePurpose\ prp)$, se elimina un propósito del sistema, para lo cual éste debe existir previamente.

$$Pre\ s\ (DeletePurpose\ prp) :=\ prp \in s.prp_{\text{S}}$$

Para $(AddPrivacyPermission\ prm\ prp)$ se busca agregar un nuevo permiso de privacidad al sistema, a partir de un permiso prm y de un propósito prp. Para conseguirlo, tanto prm como prp deben existir en el sistema y no pueden estar previamente asociados a un permiso de privacidad.

$$\begin{aligned} &Pre\ s\ (AddPrivacyPermission\ prm\ prp) :=\\ &\quad prm \in s.prm_{\text{S}}\ \wedge\ prp \in s.prp_{\text{S}}\ \wedge\ \neg\,((prm,\ prp) \in PP\ \wedge\ (prm,\ prp) \in s.pp_{\text{S}}) \end{aligned}$$

$(DeletePrivacyPermission\ prm\ prp)$, en cambio, pretende eliminar un permiso de privacidad del sistema. Para que esto ocurra, el permiso prm y el propósito prp, que conforman el permiso de privacidad, deben existir en el sistema.

$$\begin{aligned} &Pre\ s\ (DeletePrivacyPermission\ prm\ prp) :=\\ &\quad prm \in s.prm_{\text{S}}\ \wedge\ prp \in s.prp_{\text{S}}\ \wedge\ (prm,\ prp) \in PP\ \wedge\ (prm,\ prp) \in s.pp_{\text{S}} \end{aligned}$$

Con $(GrantPrivacyPermission\ prm\ prp\ r)$ se busca asignar el permiso de privacidad formado por el permiso prm y el propósito prp al rol r. Para que esto funcione, $(prm,\ prp)$ debe ser un permiso de privacidad válido y r debe existir en el sistema. Además, dicho permiso de privacidad no puede estar asignado previamente al rol.

$$
\begin{aligned}
&Pre\ s\ (GrantPrivacyPermission\ prm\ prp\ r) :=\\
&\quad (prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\mathrm{S}} \wedge\\
&\quad r \in s.r_{\mathrm{S}} \wedge\\
&\quad \neg\ (((prm,\ prp),\ r) \in PPA\ \ \wedge\ \ ((prm,\ prp),\ r) \in s.ppa_{\mathrm{S}})
\end{aligned}
$$

El comando $(RevokePrivacyPermision\ prm\ prp\ r)$, en cambio, desasigna al permiso de privacidad formado por prm y prp asignado al rol r. Para conseguirlo, $(prm,\ prp)$ tiene que ser un permiso de privacidad válido, r debe existir en el sistema y el permiso de privacidad debe estar asignado previamente al rol.

$$
\begin{aligned}
&Pre\ s\ (RevokePrivacyPermission\ prm\ prp\ r) :=\\
&\quad (prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\mathrm{S}} \wedge\\
&\quad r \in s.r_{\mathrm{S}} \wedge\\
&\quad ((prm,\ prp),\ r) \in PPA\ \ \wedge\ \ ((prm,\ prp),\ r) \in s.ppa_{\mathrm{S}}
\end{aligned}
$$

La operación $(AddPrivacyDataType\ pdt)$ busca agregar un nuevo tipo de privacidad al sistema. Para poder hacerlo, este no puede existir previamente.

$$Pre\ s\ (AddPrivacyDataType\ pdt) :=\ \neg\ (pdt \in s.pdt_{\mathrm{S}})$$

En el caso de $(DeletePrivacyDataType\ pdt)$, se quiere eliminar un tipo de privacidad del sistema. Para ello, este debe existir en el sistema.

$$Pre\ s\ (DeletePrivacyDataType\ pdt) :=\ pdt \in s.pdt_{\mathrm{S}}$$

El comando $(AddDataMapping\ ob\ pdt)$ construye una nueva relación a partir de un objeto ob y un tipo de privacidad pdt. Para conseguirlo, tanto ob como pdt deben existir en el sistema y no pueden estar previamente asociados.

$$
\begin{aligned}
&Pre\ s\ (AddDataMapping\ ob\ pdt) :=\\
&\quad ob \in s.ob_{\mathrm{S}}\ \ \wedge\ \ pdt \in s.pdt_{\mathrm{S}}\ \ \wedge\ \ \neg\ ((ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_{\mathrm{S}})
\end{aligned}
$$

$(DeleteDataMapping\ ob\ pdt)$, en cambio, desasigna un objeto ob y un tipo de privacidad pdt previamente relacionados. Para ello, tanto ob como pdt deben existir en el sistema y deben estar relacionados previamente.

$$
\begin{aligned}
&Pre\ s\ (DeleteDataMapping\ ob\ pdt) :=\\
&\quad ob \in s.ob_{\mathrm{S}}\ \ \wedge\ \ pdt \in s.pdt_{\mathrm{S}}\ \ \wedge\ \ (ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_{\mathrm{S}}
\end{aligned}
$$

La operación $(GrantConsent\ prp\ pdt)$ crea un nuevo consentimiento a partir de un propósito prp y un tipo de privacidad pdt. Esto ocurre si tanto prp como pdt existen previamente en el sistema y, además, no puede existir un consentimiento a partir de dicho propósito y tipo de privacidad.

$$Pre\ s\ (GrantConsent\ prp\ pdt) :=$$
$$prp \in s.prp_S \ \wedge \ pdt \in s.pdt_S \ \wedge \ \neg\ ((prp,\ pdt) \in CONSENTS \ \wedge \ (prp,\ pdt) \in s.consent_S)$$

Por otro lado, $(RevokeConsent\ prp\ pdt)$ elimina un consentimiento conformado de un propósito prp y un tipo de privacidad pdt. Para ello, ambos deben existir previamente en el sistema, así como el consentimiento que se desea revocar.

$$Pre\ s\ (RevokeConsent\ prp\ pdt) :=$$
$$prp \in s.prp_S \ \wedge \ pdt \in s.pdt_S \ \wedge \ (prp,\ pdt) \in CONSENTS \ \wedge \ (prp,\ pdt) \in s.consent_S$$

El comando $(AddOwner\ own)$ tiene como objetivo crear un nuevo titular de datos en el sistema. Para lograr esto, el titular no puede existir previamente.

$$Pre\ s\ (AddOwner\ own) :=\ \neg\ (own \in s.own_S)$$

En cambio $(DeleteOwner\ own)$, busca eliminar un titular de datos del sistema. Para ello, este titular debe existir en el sistema.

$$Pre\ s\ (DeleteOwner\ own) :=\ own \in s.own_S$$

La operación $(AddOwnerConsent\ own\ consent)$ asigna un titular de datos own a un consentimiento $consent$. Para conseguirlo, tanto own como $consent$ deben existir en el sistema, y además no debe existir una asignación previa entre ellos.

$$Pre\ s\ (AddOwnerConsent\ own\ consent) :=$$
$$own \in s.own_S \ \wedge \ consent \in s.consent_S \ \wedge \ \neg\ ((own,\ consent) \in OC \ \wedge \ (own,\ consent) \in s.oc_S)$$

En el caso de $(DeleteOwnerConsent\ own\ consent)$, se quiere desasignar a un titular de datos own de un consentimiento $consent$. Para esto, ambos deben existir en el sistema y debe haber una asignación previa entre ellos.

$$Pre\ s\ (DeleteOwnerConsent\ own\ consent) :=$$
$$own \in s.own_S \ \wedge \ consent \in s.consent_S \ \wedge \ (own,\ consent) \in OC \ \wedge \ (own,\ consent) \in s.oc_S$$

Funciones del sistema:

Con el comando $(AddObjectPersonalInformation\ ob\ pdt\ own)$, un objeto ob pasa a representar información personal, pudiendo luego ser asociado con el identificador de su titular own. Para ello, deben existir en el sistema tanto el objeto ob como el tipo de privacidad pdt (que también debe estar mapeado con el objeto) y el titular own. Además, no debe existir una asignación entre ob y own.

$$\begin{aligned} &Pre\ s\ (AddObjectPersonalInformation\ ob\ pdt\ own) := \\ &\quad ob \in s.ob_S \wedge \\ &\quad pdt \in s.pdt_S \wedge \\ &\quad own \in s.own_S \wedge \\ &\quad (ob,\ pdt) \in DM \ \wedge \ (ob,\ pdt) \in s.dm_S \wedge \\ &\quad \neg\ (\exists\ (ob\_pi \in OBJS\_PI),\ ob \in ob\_pi \ \wedge \ own \in objectOwner_S\ (ob\_pi)) \end{aligned}$$

En cambio, $(DropObjectPersonalInformation\ ob\ pdt\ own)$ hace que el objeto ob deje de representar información personal. Para esto, deben existir en el sistema tanto el objeto ob como el tipo de privacidad pdt (que también debe estar mapeado

con el objeto) y el titular own. Asimismo, debe existir una asignación entre el objeto y su titular.

$$
\begin{aligned}
&Pre\ s\ (DropObjectPersonalInformation\ ob\ pdt\ own) := \\
&\quad ob \in s.ob_S \wedge \\
&\quad pdt \in s.pdt_S \wedge \\
&\quad own \in s.own_S \wedge \\
&\quad (ob,\ pdt) \in DM\ \wedge\ (ob,\ pdt) \in s.dm_S \wedge \\
&\quad \exists\ (ob\_pi \in OBJS\_PI),\ ob \in ob\_pi\ \wedge\ own \in objectOwner_S\ (ob\_pi)
\end{aligned}
$$

El comando $(CreateSession\ u\ rl\ sid)$ crea una nueva sesión sid para el usuario u, activando por defecto todos los roles pertenecientes al conjunto rl. Para lograr esto, el usuario debe ser válido, la sesión no debe existir previamente y todos los roles en rl deben estar autorizados para el usuario u.

$$
\begin{aligned}
&Pre\ s\ (CreateSession\ u\ rl\ sid) := \\
&\quad u \in s.u_S\ \wedge\ \neg\ (sid \in s.sid_S)\ \wedge\ (\forall\ (r \in ROLES),\ r \in rl\ \rightarrow\ (authorizedUser\ s\ u\ r))
\end{aligned}
$$

Por otro lado, $(DeleteSession\ u\ sid)$ finaliza la sesión sid del usuario u. Para ello, tanto el usuario como la sesión deben ser válidos y, además, el usuario debe ser el dueño de la sesión.

$$
Pre\ s\ (DeleteSession\ u\ sid) :=\ u \in s.u_S\ \wedge\ sid \in s.sid_S\ \wedge\ u \in sessionUser_S\ (sid)
$$

Con $(AddActiveRole\ u\ sid\ r)$ se agrega el rol r a la lista de roles activos de la sesión sid cuyo usuario dueño es u. Para esto, u, sid y r deben ser elementos válidos del sistema, y u debe ser el dueño de la sesión sid. Además, r debe estar autorizado para u y no debe estar activo en dicha sesión.

$$
\begin{aligned}
&Pre\ s\ (AddActiveRole\ u\ sid\ r) := \\
&\quad u \in s.u_S\ \wedge \\
&\quad r \in s.r_S\ \wedge \\
&\quad sid \in s.sid_S\ \wedge \\
&\quad (authorizedUser\ s\ u\ r)\ \wedge \\
&\quad u \in sessionUser_S\ (sid)\ \wedge \\
&\quad \neg\ (r\_is\_active\_role\ s\ sid\ u\ r)
\end{aligned}
$$

En cambio, $(DropActiveRole\ u\ sid\ r)$ elimina el rol r de la lista de roles activos de la sesión sid perteneciente al usuario u. En este caso, tanto u como sid y r deben ser elementos válidos del sistema; u debe ser el dueño de la sesión sid, y r debe estar activo en dicha sesión.

$$
\begin{aligned}
&Pre\ s\ (DropActiveRole\ u\ sid\ r) := \\
&u \in s.u_S\ \wedge\ r \in s.r_S\ \wedge\ sid \in s.sid_S\ \wedge\ u \in sessionUser_S\ (sid)\ \wedge\ (r\_is\_active\_role\ s\ sid\ u\ r)
\end{aligned}
$$

El comando $(AddPermissionExercised\ sid\ r\ prm)$ agrega el permiso prm a la lista de permisos ejercidos en la sesión sid asignada al rol r. Para ello, sid, r y prm deben ser elementos válidos del sistema; r debe estar activo en la sesión y asignado a prm, y este último no puede haber sido ejercido en sid.

$$
\begin{aligned}
&Pre\ s\ (AddPermissionExercised\ sid\ r\ prm) :=\\
&\quad sid \in s.sid_S\ \wedge\\
&\quad r \in s.r_S\ \wedge\\
&\quad prm \in s.prm_S\ \wedge\\
&\quad (prm,\ r) \in PA\ \wedge\ (prm,\ r) \in s.pa_S\ \wedge\\
&\quad \exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \wedge\\
&\quad \neg\ (\exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\\
&\qquad prm \in prm\_ex\ \wedge\ prm\_ex \in sessionPermissions_S\ (sid))
\end{aligned}
$$

Por otro lado, $(DropPermissionExercised\ sid\ r\ prm)$ elimina el permiso prm de la lista de permisos ejercidos en la sesión sid asignada al rol r. Para esto, sid, r y prm deben ser elementos válidos del sistema; r debe estar activo en la sesión y asignado a prm, y este último tiene que haber sido ejercido en sid.

$$
\begin{aligned}
&Pre\ s\ (DropPermissionExercised\ sid\ r\ prm) :=\\
&\quad sid \in s.sid_S\ \wedge\\
&\quad r \in s.r_S\ \wedge\\
&\quad prm \in s.prm_S\ \wedge\\
&\quad (prm,\ r) \in PA\ \wedge\ (prm,\ r) \in s.pa_S\ \wedge\\
&\quad \exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \wedge\\
&\quad \exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\ prm \in prm\_ex\ \wedge\ prm\_ex \in sessionPermissions_S\ (sid)
\end{aligned}
$$

En el caso de $(AddPrivacyPermissionExercised\ sid\ r\ pp)$, su objetivo es agregar el permiso de privacidad pp a la lista de permisos de privacidad ejercidos en la sesión sid asignada al rol r. Para ejecutar el comando, sid, r y pp deben ser elementos válidos del sistema; r debe estar activo en la sesión y asignado a pp, y este último no puede haber sido ejercido en sid.

$$
\begin{aligned}
&Pre\ s\ (AddPrivacyPermissionExercised\ sid\ r\ pp) :=\\
&\quad sid \in s.sid_S\ \wedge\\
&\quad r \in s.r_S\ \wedge\\
&\quad pp \in s.pp_S\ \wedge\\
&\quad (pp,\ r) \in PPA\ \wedge\ (pp,\ r) \in s.ppa_S\ \wedge\\
&\quad \exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \wedge\\
&\quad \neg\ (\exists\ (pp\_ex \in PP\_EXERCISED),\ pp \in pp\_ex\ \wedge\ pp\_ex \in sessionPrivacyPermissions_S\ (sid))
\end{aligned}
$$

$(DropPrivacyPermissionExercised\ sid\ r\ pp)$, en cambio, elimina el permiso de privacidad pp de la lista de permisos de privacidad ejercidos en la sesión sid asignada al rol r. Para ello, sid, r y pp deben ser elementos válidos del sistema; r debe estar activo en la sesión y asignado a pp, y este último debe haber sido ejercido en sid.

$$
\begin{aligned}
&Pre\ s\ (DropPrivacyPermissionExercised\ sid\ r\ pp) :=\\
&\quad sid \in s.sid_S\ \wedge\\
&\quad r \in s.r_S\ \wedge\\
&\quad pp \in s.pp_S\ \wedge\\
&\quad (pp,\ r) \in PPA\ \wedge\ (pp,\ r) \in s.ppa_S\ \wedge\\
&\quad \exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\ (sid)\ \wedge\\
&\quad \exists\ (pp\_ex \in PP\_EXERCISED),\ pp \in pp\_ex\ \wedge\ pp\_ex \in sessionPrivacyPermissions_S\ (sid)
\end{aligned}
$$

En el caso de $(CheckAccess\ sid\ op\ ob)$, se llevan a cabo las decisiones de control de acceso a los recursos protegidos por el sistema. Tanto la operación como el objeto

que conforman el permiso, así como la sesión, deben existir en el sistema para poder ejecutar el comando.

$$Pre\ s\ (CheckAccess\ sid\ op\ ob) :=\ \ op \in s.op_S\ \ \wedge\ \ ob \in s.ob_S\ \ \wedge\ \ sid \in s.sid_S$$

Funciones de revisión:

La operación $(AssignedUsers\ r)$ devuelve la lista de usuarios asignados al rol r y no modifica el estado. Para que se ejecute correctamente, es necesario que r sea un rol válido del sistema.

$$Pre\ s\ (AssignedUsers\ r) :=\ \ r \in s.r_S$$

En el caso de $(AssignedRoles\ u)$, devuelve la lista de roles asignados al usuario u y no modifica el estado. Para que se ejecute correctamente, es necesario que u sea un usuario válido del sistema.

$$Pre\ s\ (AssignedRoles\ u) :=\ \ u \in s.u_S$$

El comando $(AuthorizedUsers\ r)$ devuelve la lista de usuarios autorizados para el rol r. Para ello, r debe ser un rol válido del sistema.

$$Pre\ s\ (AuthorizedUsers\ r) :=\ \ r \in s.r_S$$

En cambio, $(AuthorizedRoles\ u)$ devuelve la lista de roles autorizados para el usuario u. Para ello, u debe ser un usuario válido del sistema.

$$Pre\ s\ (AuthorizedRoles\ u) :=\ \ u \in s.u_S$$

3.4.3. Errores y posibles respuestas

Para definir las postcondiciones de los comandos, primero se deben establecer sus posibles respuestas. Inicialmente, se definen los posibles errores que un comando puede devolver en caso de que no se cumpla alguna de sus precondiciones. Los posibles códigos de error se definen mediante el tipo enumerado $ErrorCode$:

$$
\begin{aligned}
ErrorCode :=\ & u\_exists \\
& | \ u\_not\_exist \\
& | \ r\_exists \\
& | \ r\_not\_exist \\
& | \ u\_assigned\_to\_r \\
& | \ u\_not\_assigned\_to\_r \\
& | \ op\_exists \\
& | \ op\_not\_exist \\
& | \ ob\_exists \\
& | \ ob\_not\_exist \\
& | \ prm\_exists \\
& | \ prm\_not\_exist \\
& | \ prm\_assigned\_to\_r \\
& | \ prm\_not\_assigned\_to\_r \\
& | \ inh\_defined \\
& | \ rDesc\_parent\_of\_rAsc \\
& | \ inh\_not\_defined \\
& | \ prp\_exists \\
& | \ prp\_not\_exist \\
& | \ pp\_exists \\
& | \ pp\_not\_exist \\
& | \ pp\_assigned\_to\_r \\
& | \ pp\_not\_assigned\_to\_r \\
& | \ pdt\_exists \\
& | \ pdt\_not\_exist \\
& | \ data\_mapped \\
& | \ data\_not\_mapped \\
& | \ consent\_exists \\
& | \ consent\_not\_exist \\
& | \ own\_exists \\
& | \ own\_not\_exist \\
& | \ consent\_granted \\
& | \ consent\_not\_granted \\
& | \ ob\_assigned\_to\_own \\
& | \ ob\_not\_assigned\_to\_own \\
& | \ sid\_exists \\
& | \ sid\_not\_exist \\
& | \ sid\_not\_linked\_to\_u \\
& | \ r\_is\_active \\
& | \ r\_is\_not\_active \\
& | \ prm\_exercised \\
& | \ prm\_not\_exercised \\
& | \ pp\_exercised \\
& | \ pp\_not\_exercised
\end{aligned}
$$

Luego, se define la relación $ErrorMsg$, donde para todo comando f se asocia un código de error por cada error posible, así como la condición del comando asociada a dicho error.

Los errores pueden ocurrir si la precondición no se cumple. Luego, la condición para cada comando se establece como la negación de su precondición.

Para mayor claridad, en los Cuadros 1, 2 y 3 se representa a $ErrorMsg$ para cada uno de los comandos, según sean administrativos, funciones del sistema o de revisión, respectivamente. Las columnas de los cuadros contienen los posibles códigos de error y sus condiciones correspondientes.

Cuadro 1: Representación de $ErrorMsg$ para los comandos administrativos

$AddUser\ u$	
u_exists	$u \in s.u_S$
$DeleteUser\ u$	
u_not_exist	$\neg\ (u \in s.u_S)$
$AddRole\ r$	
r_exists	$r \in s.r_S$
$DeleteRole\ r$	
r_not_exist	$\neg\ (r \in s.r_S)$
$AssignUser\ u\ r$	
u_not_exist	$\neg\ (u \in s.u_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
$u\_assigned\_to\_r$	$role\_of\_user\ s\ r\ u$
$DeassignUser\ u\ r$	
u_not_exist	$\neg\ (u \in s.u_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
$u\_not\_assigned\_to\_r$	$\neg\ (role\_of\_user\ s\ r\ u)$
$AddOperation\ op$	
op_exists	$op \in s.op_S$
$DeleteOperation\ op$	
op_not_exist	$\neg\ (op \in s.op_S)$
$AddObject\ ob$	
ob_exists	$ob \in s.ob_S$

$DeleteObject\ ob$	
ob_not_exist	$\neg\ (ob \in s.ob_S)$
$AddPermission\ op\ ob$	
op_not_exist	$\neg\ (op \in s.op_S)$
ob_not_exist	$\neg\ (ob \in s.ob_S)$
prm_exists	$(op,\ ob) \in PERMISSIONS\ \ \wedge\ \ (op,\ ob) \in s.prm_S$
$DeletePermission\ op\ ob$	
op_not_exist	$\neg\ (op \in s.op_S)$
ob_not_exist	$\neg\ (ob \in s.ob_S)$
prm_not_exist	$\neg\ ((op,\ ob) \in PERMISSIONS\ \ \wedge\ \ (op,\ ob) \in s.prm_S)$
$GrantPermission\ ob\ op\ r$	
prm_not_exist	$\neg\ ((op,\ ob) \in PERMISSIONS\ \ \wedge\ \ (op,\ ob) \in s.prm_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
$prm\_assigned\_to\_r$	$((op,\ ob),\ r) \in PA\ \ \wedge\ \ ((op,\ ob),\ r) \in s.pa_S$
$RevokePermission\ ob\ op\ r$	
prm_not_exist	$\neg\ ((op,\ ob) \in PERMISSIONS\ \ \wedge\ \ (op,\ ob) \in s.prm_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
$prm\_not\_assigned\_to\_r$	$\neg\ (((op,\ ob),\ r) \in PA\ \ \wedge\ \ ((op,\ ob),\ r) \in s.pa_S)$
$AddInheritance\ r_{Asc}\ r_{Desc}$	
r_not_exist	$\neg\ (r_{Asc} \in s.r_S)\ \ \vee\ \ \neg\ (r_{Desc} \in s.r_S)$
$inh\_defined$	$r_{Desc}\ \ll_S r_{Asc}$
$rDesc\_parent\_of\_rAsc$	$r_{Asc}\ \preccurlyeq_S r_{Desc}$
$DeleteInheritance\ r_{Asc}\ r_{Desc}$	
r_not_exist	$\neg\ (r_{Asc} \in s.r_S)\ \ \vee\ \ \neg\ (r_{Desc} \in s.r_S)$
$inh\_not\_defined$	$\neg\ (r_{Desc}\ \ll_S r_{Asc})$
$AddAscendant\ r_{Asc}\ r_{Desc}$	
r_exists	$r_{Asc} \in s.r_S$
r_not_exist	$\neg\ (r_{Desc} \in s.r_S)$
$AddDescendant\ r_{Asc}\ r_{Desc}$	

r_exists	$r_{\text{Desc}} \in s.r_{\text{S}}$
r_not_exist	$\neg\ (r_{\text{Asc}} \in s.r_{\text{S}})$
$AddPurpose\ prp$	
prp_exists	$prp \in s.prp_{\text{S}}$
$DeletePurpose\ prp$	
prp_not_exist	$\neg\ (prp \in s.prp_{\text{S}})$
$AddPrivacyPermission\ prm\ prp$	
prm_not_exist	$\neg\ (prm \in s.prm_{\text{S}})$
prp_not_exist	$\neg\ (prp \in s.prp_{\text{S}})$
pp_exists	$(prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\text{S}}$
$DeletePrivacyPermission\ prm\ prp$	
prm_not_exist	$\neg\ (prm \in s.prm_{\text{S}})$
prp_not_exist	$\neg\ (prp \in s.prp_{\text{S}})$
pp_not_exist	$\neg\ ((prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\text{S}})$
$GrantPrivacyPermission\ prm\ prp\ r$	
pp_not_exist	$\neg\ ((prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\text{S}})$
r_not_exist	$\neg\ (r \in s.r_{\text{S}})$
$pp\_assigned\_to\_r$	$((prm,\ prp),\ r) \in PPA\ \ \wedge\ \ ((prm,\ prp),\ r) \in s.ppa_{\text{S}}$
$RevokePrivacyPermission\ prm\ prp\ r$	
pp_not_exist	$\neg\ ((prm,\ prp) \in PP\ \ \wedge\ \ (prm,\ prp) \in s.pp_{\text{S}})$
r_not_exist	$\neg\ (r \in s.r_{\text{S}})$
$pp\_not\_assigned\_to\_r$	$\neg\ (((prm,\ prp),\ r) \in PPA\ \ \wedge\ \ ((prm,\ prp),\ r) \in s.ppa_{\text{S}})$
$AddPrivacyDataType\ pdt$	
pdt_exists	$pdt \in s.pdt_{\text{S}}$
$DeletePrivacyDataType\ pdt$	
pdt_not_exist	$\neg\ (pdt \in s.pdt_{\text{S}})$
$AddDataMapping\ ob\ pdt$	
ob_not_exist	$\neg\ (ob \in s.ob_{\text{S}})$
pdt_not_exist	$\neg\ (pdt \in s.pdt_{\text{S}})$

$data\_mapped$	$(ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_{\mathrm{S}}$
$DeleteDataMapping\ ob\ pdt$	
ob_not_exist	$\neg\ (ob \in s.ob_{\mathrm{S}})$
pdt_not_exist	$\neg\ (pdt \in s.pdt_{\mathrm{S}})$
$data\_not\_mapped$	$\neg\ ((ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_{\mathrm{S}})$
$GrantConsent\ prp\ pdt$	
prp_not_exist	$\neg\ (prp \in s.prp_{\mathrm{S}})$
pdt_not_exist	$\neg\ (pdt \in s.pdt_{\mathrm{S}})$
$consent\_exists$	$(prp,\ pdt) \in CONSENTS\ \ \wedge\ \ (prp,\ pdt) \in s.consent_{\mathrm{S}}$
$RevokeConsent\ prp\ pdt$	
prp_not_exist	$\neg\ (prp \in s.prp_{\mathrm{S}})$
pdt_not_exist	$\neg\ (pdt \in s.pdt_{\mathrm{S}})$
$consent\_not\_exist$	$\neg\ ((prp,\ pdt) \in CONSENTS\ \ \wedge\ \ (prp,\ pdt) \in s.consent_{\mathrm{S}})$
$AddOwner\ own$	
own_exists	$own \in s.own_{\mathrm{S}}$
$DeleteOwner\ own$	
own_not_exist	$\neg\ (own \in s.own_{\mathrm{S}})$
$AddOwnerConsent\ own\ consent$	
own_not_exist	$\neg\ (own \in s.own_{\mathrm{S}})$
$consent\_not\_exist$	$\neg\ (consent \in s.consent_{\mathrm{S}})$
$consent\_granted$	$(own,\ consent) \in OC\ \ \wedge\ \ (own,\ consent) \in s.oc_{\mathrm{S}}$
$DeleteOwnerConsent\ own\ consent$	
own_not_exist	$\neg\ (own \in s.own_{\mathrm{S}})$
$consent\_not\_exist$	$\neg\ (consent \in s.consent_{\mathrm{S}})$
$consent\_not\_granted$	$\neg\ ((own,\ consent) \in OC\ \ \wedge\ \ (own,\ consent) \in s.oc_{\mathrm{S}})$

Cuadro 2: Representación de $ErrorMsg$ para las funciones del sistema

$AddObjectPersonalInformation\ ob\ pdt\ own$	
ob_not_exist	$\neg\ (ob \in s.ob_S)$
pdt_not_exist	$\neg\ (pdt \in s.pdt_S)$
own_not_exist	$\neg\ (own \in s.own_S)$
$data\_not\_mapped$	$\neg\ ((ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_S)$
$ob\_assigned\_to\_own$	$\exists\ (ob\_pi \in OBJS\_PI),\ ob \in ob\_pi\ \ \wedge\ \ own \in objectOwner_S\ (ob\_pi)$
$DropObjectPersonalInformation\ ob\ pdt\ own$	
ob_not_exist	$\neg\ (ob \in s.ob_S)$
pdt_not_exist	$\neg\ (pdt \in s.pdt_S)$
own_not_exist	$\neg\ (own \in s.own_S)$
$data\_not\_mapped$	$\neg\ ((ob,\ pdt) \in DM\ \ \wedge\ \ (ob,\ pdt) \in s.dm_S)$
$ob\_not\_assigned\_to\_own$	$\neg\ (\exists\ (ob\_pi \in OBJS\_PI),\ ob \in ob\_pi\ \ \wedge\ \ own \in objectOwner_S\ (ob\_pi))$
$CreateSession\ u\ rl\ sid$	
u_not_exist	$\neg\ (u \in s.u_S)$
$u\_not\_assigned\_to\_r$	$\neg\ (\forall\ (r \in ROLES),\ r \in rl\ \rightarrow\ (role\_of\_user\ s\ r\ u))$
sid_exists	$sid \in s.sid_S$
$DeleteSession\ u\ sid$	
u_not_exist	$\neg\ (u \in s.u_S)$
sid_not_exist	$\neg\ (sid \in s.sid_S)$
$sid\_not\_linked\_to\_u$	$\neg\ (u \in sessionUser_S\ (sid))$
$AddActiveRole\ u\ sid\ r$	
u_not_exist	$\neg\ (u \in s.u_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
sid_not_exist	$\neg\ (sid \in s.sid_S)$
$u\_not\_assigned\_to\_r$	$\neg\ (role\_of\_user\ s\ r\ u)$
r_is_active	$r\_is\_active\_role\ s\ sid\ u\ r$
$sid\_not\_linked\_to\_u$	$\neg\ (u \in sessionUser_S\ (sid))$
$DropActiveRole\ u\ sid\ r$	

u_not_exist	$\neg\ (u \in s.u_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
sid_not_exist	$\neg\ (sid \in s.sid_S)$
$r\_is\_not\_active$	$\neg\ (r\_is\_active\_role\ s\ sid\ u\ r)$
$sid\_not\_linked\_to\_u$	$\neg\ (u \in sessionUser_S\ (sid))$
$AddPermissionExcercised\ sid\ r\ prm$	
sid_not_exist	$\neg\ (sid \in s.sid_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
prm_not_exist	$\neg\ (prm \in s.prm_S)$
$prm\_not\_assigned\_to\_r$	$\neg\ ((prm,\ r) \in PA\ \ \wedge\ \ (prm,\ r) \in s.pa_S)$
$r\_is\_not\_active$	$\neg\ (\exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \ \wedge\ \ ar \in sessionRoles_S\ (sid))$
$prm\_exercised$	$\exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),$ $prm \in prm\_ex\ \ \wedge\ \ prm\_ex \in sessionPermissions_S\ (sid)$
$DropPermissionExcercised\ sid\ r\ prm$	
sid_not_exist	$\neg\ (sid \in s.sid_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
prm_not_exist	$\neg\ (prm \in s.prm_S)$
$prm\_not\_assigned\_to\_r$	$\neg\ ((prm,\ r) \in PA\ \ \wedge\ \ (prm,\ r) \in s.pa_S)$
$r\_is\_not\_active$	$\neg\ (\exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \ \wedge\ \ ar \in sessionRoles_S\ (sid))$
$prm\_not\_exercised$	$\neg\ (\exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),$ $prm \in prm\_ex\ \ \wedge\ \ prm\_ex \in sessionPermissions_S\ (sid))$
$AddPrivacyPermissionExcercised\ sid\ r\ pp$	
sid_not_exist	$\neg\ (sid \in s.sid_S)$
r_not_exist	$\neg\ (r \in s.r_S)$
pp_not_exist	$\neg\ (pp \in s.pp_S)$
$pp\_not\_assigned\_to\_r$	$\neg\ ((pp,\ r) \in PPA\ \ \wedge\ \ (pp,\ r) \in s.ppa_S)$
$r\_is\_not\_active$	$\neg\ (\exists\ (ar \in ACTIVE\_ROLES),\ r \in ar\ \ \wedge\ \ ar \in sessionRoles_S\ (sid))$
$pp\_exercised$	$\exists\ (pp\_ex \in PP\_EXERCISED),$ $pp \in pp\_ex\ \ \wedge\ \ pp\_ex \in sessionPrivacyPermissions_S\ (sid)$
$DropPrivacyPermissionExcercised\ sid\ r\ pp$	
sid_not_exist	$\neg\ (sid \in s.sid_S)$

r_not_exist	$\neg\,(r \in s.r_S)$
pp_not_exist	$\neg\,(pp \in s.pp_S)$
$pp\_not\_assigned\_to\_r$	$\neg\,((pp,\ r) \in PPA\ \wedge\ (pp,\ r) \in s.ppa_S)$
$r\_is\_not\_active$	$\neg\,(\exists\,(ar \in ACTIVE\_ROLES),\ r \in ar\ \wedge\ ar \in sessionRoles_S\,(sid))$
$pp\_not\_exercised$	$\neg\,(\exists\,(pp\_ex \in PP\_EXERCISED),$ $pp \in pp\_ex\ \wedge\ pp\_ex \in sessionPrivacyPermissions_S\,(sid))$
$CheckAccess\ sid\ op\ ob$	
op_not_exist	$\neg\,(op \in s.op_S)$
ob_not_exist	$\neg\,(ob \in s.ob_S)$
sid_not_exist	$\neg\,(sid \in s.sid_S)$

Cuadro 3: Representación de $ErrorMsg$ para las funciones de revisión

$AssignedUsers\ r$	
r_not_exist	$\neg\,(r \in s.r_S)$
$AssignedRoles\ u$	
u_not_exist	$\neg\,(u \in s.u_S)$
$AuthorizedUsers\ r$	
r_not_exist	$\neg\,(r \in s.r_S)$
$AuthorizedRoles\ u$	
u_not_exist	$\neg\,(u \in s.u_S)$

Luego, las posibles respuestas del sistema están especificadas mediante el tipo inductivo no recursivo $Answer$:

$$
\begin{aligned}
Answer :=\ & ok : Answer \\
& \mid fail : Answer \\
& \mid user\_list : set\ USERS \rightarrow Answer \\
& \mid role\_list : set\ ROLES \rightarrow Answer \\
& \mid error : ErrorCode \rightarrow Answer
\end{aligned}
$$

Donde:

- ok: Todos los comandos, excepto los de revisión, devuelven ok como respuesta en caso de ejecutarse correctamente.
- $fail$: Es la respuesta asociada cuando existe algún inconveniente en la ejecución del comando.
- $user\_list$: Devuelve una lista de usuarios de tipo $set\ USERS$ como respuesta de los comandos $AssignedUsers$ y $AuthorizedUsers$.
- $role\_list$: Devuelve una lista de roles de tipo $set\ ROLES$ como respuesta de los comandos $AssignedRoles$ y $AuthorizedRoles$.
- $error$: Devuelve un código de error de tipo $ErrorCode$ cuando la precondición del comando no se cumple.

3.4.4. Postcondiciones

Una vez establecidos los posibles errores y respuestas, se procede ahora a detallar las postcondiciones para cada comando. Estas son predicados que relacionan el estado antes y después de ejecutar el comando correspondiente. En cada caso se omiten los campos del estado que permanecen invariantes al ejecutar el comando. Se representan mediante el conjunto inductivo Pos.

Comandos administrativos:

Para el caso de $(AddUser\ u)$, la diferencia entre ambos estados es la inclusión del nuevo usuario u en el sistema.

$$Pos\ s\ s'\ ans\ (AddUser\ u) :=\ s'.u_S\ =\ s.u_S\ \cup\ \{u\}\ \wedge\ ans\ =\ ok$$

En el caso de $(DeleteUser\ u)$, al eliminar el usuario u, también se eliminan todas las asignaciones de roles y las sesiones establecidas que posee.

$$\begin{aligned} &Pos\ s\ s'\ ans\ (DeleteUser\ u) :=\\ &\quad s'.u_S\ =\ s.u_S \setminus \{u\}\ \wedge\\ &\quad (\forall\ (ua \in UA),\ s'.ua_S\ =\ s.ua_S \setminus \{ua\}\ \wedge\ u \in ua.u_{UA}\ \wedge\ ua \in s.ua_S)\ \wedge\\ &\quad (\forall\ (sid \in SESSIONS),\ s'.sid_S\ =\ s.sid_S \setminus \{sid\}\ \wedge\ u \in sessionUser_S\ (sid)\ \wedge\ sid \in s.sid_S)\ \wedge\\ &\quad ans\ =\ ok \end{aligned}$$

Para $(AddRole\ r)$, la diferencia entre ambos estados es la inclusión del nuevo rol r en el sistema.

$$Pos\ s\ s'\ ans\ (AddRole\ r) :=\ s'.r_S\ =\ s.r_S\ \cup\ \{r\}\ \wedge\ ans\ =\ ok$$

En $(DeleteRole\ r)$, además de eliminar el rol r, se finalizan las sesiones que tienen activo dicho rol, así como también todas las asignaciones de usuario, permisos, permisos de privacidad y las herencias en las que este interviene para poder conservar la validez del estado.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeleteRole\ r) :=\\
&\quad s'.r_{\mathsf{S}} = s.r_{\mathsf{S}} \setminus \{r\} \ \wedge\\
&\quad (\forall\ (ua \in UA),\ s'.ua_{\mathsf{S}} = s.ua_{\mathsf{S}} \setminus \{ua\} \ \wedge\ r \in ua.r_{\mathsf{UA}} \ \wedge\ ua \in s.ua_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (pa \in PA),\ s'.pa_{\mathsf{S}} = s.pa_{\mathsf{S}} \setminus \{pa\} \ \wedge\ r \in pa.r_{\mathsf{PA}} \ \wedge\ pa \in s.pa_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (ppa \in PPA),\ s'.ppa_{\mathsf{S}} = s.ppa_{\mathsf{S}} \setminus \{ppa\} \ \wedge\ r \in ppa.r_{\mathsf{PPA}} \ \wedge\ ppa \in s.ppa_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (ar \in ACTIVE\_ROLES),\\
&\qquad s'.sid_{\mathsf{S}} = s.sid_{\mathsf{S}} \setminus \{sid\} \ \wedge\ ar \in sessionRoles_{\mathsf{S}}\ (sid) \ \wedge\ r \in ar \ \wedge\ sid \in s.sid_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (r_{\mathsf{Desc}}\ r_{\mathsf{Asc}} \in ROLES),\\
&\qquad ((r = r_{\mathsf{Desc}} \ \wedge\ \neg\ (r = r_{\mathsf{Asc}})) \ \wedge\ (\neg\ (r \ll_{\mathsf{S}}' r_{\mathsf{Asc}}) \ \wedge\ r \ll_{\mathsf{S}} r_{\mathsf{Asc}})) \ \vee\\
&\qquad ((\neg\ (r = r_{\mathsf{Desc}}) \ \wedge\ r = r_{\mathsf{Asc}}) \ \wedge\ (\neg\ (r_{\mathsf{Desc}} \ll_{\mathsf{S}}' r) \ \wedge\ r_{\mathsf{Desc}} \ll_{\mathsf{S}} r)) \ \vee\\
&\qquad ((\neg\ (r = r_{\mathsf{Desc}}) \ \wedge\ \neg\ (r = r_{\mathsf{Asc}})) \ \wedge\ (s'.\ll_{\mathsf{S}} = s.\ll_{\mathsf{S}}))) \ \wedge\\
&\quad ans = ok
\end{aligned}
$$

Para el comando $(AssignUser\ u\ r)$, se agrega una asignación usuario-rol entre el usuario u y el rol r.

$$
Pos\ s\ s'\ ans\ (AssignUser\ u\ r) :=\ s'.ua_{\mathsf{S}} = s.ua_{\mathsf{S}} \cup \{(u, r)\} \ \wedge\ ans = ok
$$

En el caso de $(DeassignUser\ u\ r)$, además de eliminar la asignación entre el usuario u y el rol r, se finalizan las sesiones de u que tengan activo a r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeassignUser\ u\ r) :=\\
&\quad s'.ua_{\mathsf{S}} = s.ua_{\mathsf{S}} \setminus \{(u, r)\} \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (ar \in ACTIVE\_ROLES),\\
&\qquad s'.sid_{\mathsf{S}} = s.sid_{\mathsf{S}} \setminus \{sid\} \ \wedge\ ar \in sessionRoles_{\mathsf{S}}\ (sid) \ \wedge\ r \in ar \ \wedge\ sid \in s.sid_{\mathsf{S}}) \ \wedge\\
&\quad ans = ok
\end{aligned}
$$

Para el caso de $(AddOperation\ op)$, la diferencia entre ambos estados es la inclusión de la nueva operación op en el sistema.

$$
Pos\ s\ s'\ ans\ (AddOperation\ op) :=\ s'.op_{\mathsf{S}} = s.op_{\mathsf{S}} \cup \{op\} \ \wedge\ ans = ok
$$

En el caso de $(DeleteOperation\ op)$, además de eliminar la operación op, se eliminan los permisos asociados a dicha operación. Al borrarse los permisos, también se eliminan los permisos de privacidad, así como las asignaciones con roles y sesiones asociadas a estos permisos.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeleteOperation\ op) :=\\
&\quad s'.op_{\mathsf{S}} = s.op_{\mathsf{S}} \setminus \{op\} \ \wedge\\
&\quad (\forall\ (prm \in PERMISSIONS),\ s'.prm_{\mathsf{S}} = s.prm_{\mathsf{S}} \setminus \{prm\} \ \wedge\ op \in prm.op_{\mathsf{PRM}} \ \wedge\ prm \in s.prm_{\mathsf{S}} \ \wedge\\
&\quad (\forall\ (pa \in PA),\ s'.pa_{\mathsf{S}} = s.pa_{\mathsf{S}} \setminus \{pa\} \ \wedge\ prm \in pa.prm_{\mathsf{PA}} \ \wedge\ pa \in s.pa_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (pp \in PP),\ s'.pp_{\mathsf{S}} = s.pp_{\mathsf{S}} \setminus \{pp\} \ \wedge\ prm \in pp.prm_{\mathsf{PP}} \ \wedge\ pp \in s.pp_{\mathsf{S}} \ \wedge\\
&\quad (\forall\ (ppa \in PPA),\ s'.ppa_{\mathsf{S}} = s.ppa_{\mathsf{S}} \setminus \{ppa\} \ \wedge\ pp \in ppa.pp_{\mathsf{PPA}} \ \wedge\ ppa \in s.ppa_{\mathsf{S}})) \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\\
&\qquad s'.sid_{\mathsf{S}} = s.sid_{\mathsf{S}} \setminus \{sid\} \ \wedge\\
&\qquad prm\_ex \in sessionPermissions_{\mathsf{S}}\ (sid) \ \wedge\\
&\qquad prm \in prm\_ex \ \wedge\\
&\qquad sid \in s.sid_{\mathsf{S}})) \ \wedge\\
&\quad ans = ok
\end{aligned}
$$

Para el comando $(AddObject\ ob)$, la diferencia entre ambos estados es la inclusión del nuevo objeto ob en el sistema.

$$
Pos\ s\ s'\ ans\ (AddObject\ ob) :=\ s'.ob_{\mathsf{S}} = s.ob_{\mathsf{S}} \cup \{ob\} \ \wedge\ ans = ok
$$

En cambio, para $(DeleteObject\ ob)$, además de eliminar el objeto ob, se eliminan los permisos asociados a dicho objeto y los mapeos que tenga con distintos tipos de privacidad. Además, al borrarse permisos, también se eliminan los permisos de privacidad, las asignaciones con roles y las sesiones asociadas a estos permisos.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeleteObject\ ob) :=\\
&\quad s'.ob_S = s.ob_S \setminus \{ob\} \ \wedge\\
&\quad (\forall\ (prm \in PERMISSIONS),\ s'.prm_S = s.prm_S \setminus \{prm\} \ \wedge\ ob \in prm.ob_{PRM} \ \wedge\ prm \in s.prm_S \ \wedge\\
&\quad (\forall\ (pa \in PA),\ s'.pa_S = s.pa_S \setminus \{pa\} \ \wedge\ prm \in pa.prm_{PA} \ \wedge\ pa \in s.pa_S) \ \wedge\\
&\quad (\forall\ (pp \in PP),\ s'.pp_S = s.pp_S \setminus \{pp\} \ \wedge\ prm \in pp.prm_{PP} \ \wedge\ pp \in s.pp_S \ \wedge\\
&\quad (\forall\ (ppa \in PPA),\ s'.ppa_S = s.ppa_S \setminus \{ppa\} \ \wedge\ pp \in ppa.pp_{PPA} \ \wedge\ ppa \in s.ppa_S)) \ \wedge\\
&\quad (\forall\ (dm \in DM),\ s'.dm_S = s.dm_S \setminus \{dm\} \wedge ob \in dm.ob_{DM} \wedge dm \in s.dm_S) \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\\
&\qquad s'.sid_S = s.sid_S \setminus \{sid\} \ \wedge\\
&\qquad prm\_ex \in sessionPermissions_S\ (sid) \ \wedge\\
&\qquad prm \in prm\_ex \ \wedge\\
&\qquad sid \in s.sid_S)) \ \wedge\\
&\quad ans = ok
\end{aligned}
$$

Para la operación $(AddPermission\ op\ ob)$, se agrega un permiso formado por la operación op y el objeto ob.

$$Pos\ s\ s'\ ans\ (AddPermission\ op\ ob) := s'.prm_S = s.prm_S \cup \{(op,\ ob)\} \ \wedge\ ans = ok$$

En el caso de $(DeletePermission\ op\ ob)$, además de eliminar el permiso asociado a la operación op y al objeto ob, se eliminan los permisos de privacidad, las asignaciones con roles y las sesiones asociadas a este permiso.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeletePermission\ op\ ob) :=\\
&\quad s'.prm_S = s.prm_S \setminus \{(op,\ ob)\} \ \wedge\\
&\quad (\forall\ (pa \in PA),\ s'.pa_S = s.pa_S \setminus \{pa\} \ \wedge\ (op,\ ob) \in pa.prm_{PA} \ \wedge\ pa \in s.pa_S) \ \wedge\\
&\quad (\forall\ (pp \in PP),\ s'.pp_S = s.pp_S \setminus \{pp\} \ \wedge\ (op,\ ob) \in pp.prm_{PP} \ \wedge\ pp \in s.pp_S \ \wedge\\
&\quad (\forall\ (ppa \in PPA),\ s'.ppa_S = s.ppa_S \setminus \{ppa\} \ \wedge\ pp \in ppa.pp_{PPA} \ \wedge\ ppa \in s.ppa_S)) \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (prm\_ex \in PERMISSIONS\_EXERCISED),\\
&\qquad s'.sid_S = s.sid_S \setminus \{sid\} \ \wedge\\
&\qquad prm\_ex \in sessionPermissions_S\ (sid) \ \wedge\\
&\qquad (op,\ ob) \in prm\_ex \ \wedge\\
&\qquad sid \in s.sid_S) \ \wedge\\
&\quad ans = ok
\end{aligned}
$$

En $(GrantPermission\ op\ ob\ r)$, se agrega una asignación permiso-rol entre el permiso formado por la operación op y el objeto ob, y el rol r.

$$Pos\ s\ s'\ ans\ (GrantPermission\ op\ ob\ r) := s'.pa_S = s.pa_S \cup \{((op,\ ob),\ r)\} \ \wedge\ ans = ok$$

En $(RevokePermision\ op\ ob\ r)$, en cambio, se elimina la asignación permiso-rol entre el permiso formado por la operación op y el objeto ob, y el rol r.

$$Pos\ s\ s'\ ans\ (RevokePermision\ op\ ob\ r) := s'.pa_S = s.pa_S \setminus \{((op,\ ob),\ r)\} \ \wedge\ ans = ok$$

Para $(AddInheritance\ r_{Asc}\ r_{Desc})$ se agrega la herencia inmediata de roles $r_{Desc} \ll r_{Asc}$.

$$Pos\ s\ s'\ ans\ (AddInheritance\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ r_{\text{Desc}}\ \ll_{\text{S}}'\ r_{\text{Asc}}\ \wedge\ ans\ =\ ok$$

El comando $(DeleteInheritance\ r_{\text{Asc}}\ r_{\text{Desc}})$ elimina la relación de herencia inmediata de roles $r_{\text{Desc}}\ \ll\ r_{\text{Asc}}$.

$$Pos\ s\ s'\ ans\ (DeleteInheritance\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\ \neg\ (r_{\text{Desc}}\ \ll_{\text{S}}'\ r_{\text{Asc}})\ \wedge\ ans\ =\ ok$$

$(AddAscendant\ r_{\text{Asc}}\ r_{\text{Desc}})$ agrega la herencia inmediata $r_{\text{Desc}}\ \ll\ r_{\text{Asc}}$ a la jerarquía de roles, creando el rol r_{Asc} en el sistema.

$$\begin{aligned} &Pos\ s\ s'\ ans\ (AddAscendant\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\\ &\quad s'.r_{\text{S}}\ =\ s.r_{\text{S}}\ \cup\ \{r_{\text{Asc}}\}\ \wedge\ r_{\text{Desc}}\ \ll_{\text{S}}'\ r_{\text{Asc}}\ \wedge\ \neg\ (r_{\text{Desc}}\ \ll_{\text{S}}\ r_{\text{Asc}})\ \wedge\ ans\ =\ ok \end{aligned}$$

$(AddDescendant\ r_{\text{Asc}}\ r_{\text{Desc}})$ agrega la herencia inmediata $r_{\text{Desc}}\ \ll\ r_{\text{Asc}}$ a la jerarquía de roles, creando el rol r_{Desc} en el sistema.

$$\begin{aligned} &Pos\ s\ s'\ ans\ (AddDescendant\ r_{\text{Asc}}\ r_{\text{Desc}}) :=\\ &\quad s'.r_{\text{S}}\ =\ s.r_{\text{S}}\ \cup\ \{r_{\text{Desc}}\}\ \wedge\ r_{\text{Desc}}\ \ll_{\text{S}}'\ r_{\text{Asc}}\ \wedge\ \neg\ (r_{\text{Desc}}\ \ll_{\text{S}}\ r_{\text{Asc}})\ \wedge\ ans\ =\ ok \end{aligned}$$

Para el comando $(AddPurpose\ prp)$, la diferencia entre ambos estados es la inclusión del nuevo propósito prp en el sistema.

$$Pos\ s\ s'\ ans\ (AddPurpose\ prp) :=\ s'.prp_{\text{S}}\ =\ s.prp_{\text{S}}\ \cup\ \{prp\}\ \wedge\ ans\ =\ ok$$

En el caso de $(DeletePurpose\ prp)$, además de eliminar el propósito prp, se eliminan los permisos de privacidad y los consentimientos asociados al mismo. Al borrar permisos de privacidad, también se eliminan las asignaciones con roles y las sesiones asociadas a estos. Al eliminar consentimientos, también se borran sus asignaciones con titulares de datos.

$$\begin{aligned} &Pos\ s\ s'\ ans\ (DeletePurpose\ prp) :=\\ &\quad s'.prp_{\text{S}}\ =\ s.prp_{\text{S}} \setminus \{prp\}\ \wedge\\ &\quad (\forall\ (pp \in PP),\ s'.pp_{\text{S}}\ =\ s.pp_{\text{S}} \setminus \{pp\}\ \wedge\ prp \in pp.prp_{\text{PP}}\ \wedge\ pp \in s.pp_{\text{S}})\ \wedge\\ &\quad (\forall\ (ppa \in PPA),\ s'.ppa_{\text{S}}\ =\ s.ppa_{\text{S}} \setminus \{ppa\}\ \wedge\ pp \in ppa.pp_{\text{PPA}}\ \wedge\ ppa \in s.ppa_{\text{S}})\ \wedge\\ &\quad (\forall\ (consent \in CONSENTS),\\ &\qquad s'.consent_{\text{S}}\ =\ s.consent_{\text{S}} \setminus \{consent\}\ \wedge\ prp \in consent.prp_{\text{CONS}}\ \wedge\ consent \in s.consent_{\text{S}}\ \wedge\\ &\quad (\forall\ (oc \in OC),\ s'.oc_{\text{S}}\ =\ s.oc_{\text{S}} \setminus \{oc\}\ \wedge\ consent \in oc.consent_{\text{OC}}\ \wedge\ oc \in s.oc_{\text{S}}))\ \wedge\\ &\quad (\forall\ (sid \in SESSIONS),\ \exists\ (pp\_ex \in PP\_EXERCISED),\\ &\qquad s'.sid_{\text{S}}\ =\ s.sid_{\text{S}} \setminus \{sid\}\ \wedge\\ &\qquad pp\_ex \in sessionPrivacyPermissions_{\text{S}}\ (sid)\ \wedge\\ &\qquad pp \in pp\_ex\ \wedge\\ &\qquad sid \in s.sid_{\text{S}}))\ \wedge\\ &\quad ans\ =\ ok \end{aligned}$$

Para $(AddPrivacyPermission\ prm\ prp)$, se agrega un permiso de privacidad formado por el permiso prm y el propósito prp.

$$Pos\ s\ s'\ ans\ (AddPrivacyPermission\ prm\ prp) :=\ s'.pp_{\text{S}}\ =\ s.pp_{\text{S}}\ \cup\ \{(prm,\ prp)\}\ \wedge\ ans\ =\ ok$$

$(DeletePrivacyPermission\ prm\ prp)$, en cambio, elimina el permiso de privacidad formado por el permiso prm y el propósito prp. Además, se eliminan las asignaciones con roles y las sesiones asociadas a este permiso de privacidad.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ \ (DeletePrivacyPermission\ prm\ prp) :=\\
&\quad s'.pp_{\mathsf{S}} \ = \ s.pp_{\mathsf{S}} \setminus \{(prm,\ prp)\} \ \wedge\\
&\quad (\forall\ (ppa \in PPA),\ s'.ppa_{\mathsf{S}} \ = \ s.ppa_{\mathsf{S}} \setminus \{ppa\} \ \wedge\ (prm,\ prp) \in ppa.pp_{\mathsf{PPA}} \ \wedge\ ppa \in s.ppa_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (sid \in SESSIONS),\ \exists\ (pp\_ex \in PP\_EXERCISED),\\
&\qquad s'.sid_{\mathsf{S}} \ = \ s.sid_{\mathsf{S}} \setminus \{sid\} \ \wedge\\
&\qquad pp\_ex \in sessionPrivacyPermissions_{\mathsf{S}}\ (sid) \ \wedge\\
&\qquad (prm,\ prp) \in pp\_ex \ \wedge\\
&\qquad sid \in s.sid_{\mathsf{S}}) \ \wedge\\
&\quad ans \ = \ ok
\end{aligned}
$$

Con $(GrantPrivacyPermission\ prm\ prp\ r)$, se agrega una asignación permiso de privacidad-rol entre el permiso de privacidad formado por el permiso prm y el propósito prp, y el rol r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (GrantPrivacyPermission\ prm\ prp\ r) :=\\
&\qquad s'.ppa_{\mathsf{S}} \ = \ s.ppa_{\mathsf{S}} \ \cup\ \{((prm,\ prp),\ r)\} \ \wedge\ ans \ = \ ok
\end{aligned}
$$

El comando $(RevokePrivacyPermision\ prm\ prp\ r)$ elimina la asignación permiso de privacidad-rol entre el permiso de privacidad formado por el permiso prm y el propósito prp, y el rol r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (GrantPrivacyPermission\ prm\ prp\ r) :=\\
&\qquad s'.ppa_{\mathsf{S}} \ = \ s.ppa_{\mathsf{S}} \setminus \{((prm,\ prp),\ r)\} \ \wedge\ ans \ = \ ok
\end{aligned}
$$

Para la operación $(AddPrivacyDataType\ pdt)$, la diferencia entre ambos estados es la inclusión del nuevo tipo de privacidad pdt en el sistema.

$$
Pos\ s\ s'\ ans\ (AddPrivacyDataType\ pdt) :=\ s'.pdt_{\mathsf{S}} \ = \ s.pdt_{\mathsf{S}} \ \cup\ \{pdt\} \ \wedge\ ans \ = \ ok
$$

En el caso de $(DeletePrivacyDataType\ pdt)$, además de eliminar el tipo de privacidad pdt, se eliminan los mapeos que tenga con distintos objetos, así como los consentimientos asociados al mismo. Al eliminar los consentimientos, también se borran sus asignaciones con titulares de datos.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DeletePrivacyDataType\ pdt) :=\\
&\quad s'.pdt_{\mathsf{S}} \ = \ s.pdt_{\mathsf{S}} \setminus \{pdt\} \ \wedge\\
&\quad (\forall\ (dm \in DM),\ s'.dm_{\mathsf{S}} \ = \ s.dm_{\mathsf{S}} \setminus \{dm\} \ \wedge\ pdt \in dm.pdt_{\mathsf{DM}} \ \wedge\ dm \in s.dm_{\mathsf{S}}) \ \wedge\\
&\quad (\forall\ (consent \in CONSENTS),\\
&\qquad s'.consent_{\mathsf{S}} \ = \ s.consent_{\mathsf{S}} \setminus \{consent\} \ \wedge\ pdt \in consent.pdt_{\mathsf{CONS}} \ \wedge\ consent \in s.consent_{\mathsf{S}} \ \wedge\\
&\quad (\forall\ (oc \in OC),\ s'.oc_{\mathsf{S}} \ = \ s.oc_{\mathsf{S}} \setminus \{oc\} \ \wedge\ consent \in oc.consent_{\mathsf{OC}} \ \wedge\ oc \in s.oc_{\mathsf{S}})) \ \wedge\\
&\quad ans \ = \ ok
\end{aligned}
$$

El comando $(AddDataMapping\ ob\ pdt)$ se agrega una asignación objeto-tipo de privacidad entre el objeto ob, y el tipo de privacidad pdt.

$$
Pos\ s\ s'\ ans\ (AddDataMapping\ ob\ pdt) :=\ s'.dm_{\mathsf{S}} \ = \ s.dm_{\mathsf{S}} \ \cup\ \{(ob,\ pdt)\} \ \wedge\ ans \ = \ ok
$$

$(DeleteDataMapping\ ob\ pdt)$, en cambio, elimina la asignación objeto-tipo de privacidad entre el objeto ob y el tipo de privacidad pdt.

$$
Pos\ s\ s'\ ans\ (DeleteDataMapping\ ob\ pdt) :=\ s'.dm_{\mathsf{S}} \ = \ s.dm_{\mathsf{S}} \setminus \{(ob,\ pdt)\} \ \wedge\ ans \ = \ ok
$$

La operación $(GrantConsent\ prp\ pdt)$ agrega un consentimiento conformado por el propósito prp y tipo de privacidad pdt.

$$Pos\ s\ s'\ ans\ (GrantConsent\ prp\ pdt) :=\ s'.consent_S\ =\ s.consent_S\ \cup\ \{(prp,\ pdt)\}\ \wedge\ ans\ =\ ok$$

Por otro lado, $(RevokeConsent\ prp\ pdt)$ elimina el consentimiento conformado por el propósito prp y tipo de privacidad pdt. Al eliminar el consentimiento, también se borran sus asignaciones con titulares de datos.

$$\begin{aligned}&Pos\ s\ s'\ ans\ (RevokeConsent\ prp\ pdt) :=\\&\quad s'.consent_S\ =\ s.consent_S \setminus \{(prp,\ pdt)\}\ \wedge\\&\quad (\forall\ (oc \in OC),\ s'.oc_S\ =\ s.oc_S \setminus \{oc\}\ \wedge\ (prp,\ pdt) \in oc.consent_{OC}\ \wedge\ oc \in s.oc_S)\ \wedge\\&\quad ans\ =\ ok\end{aligned}$$

Para el comando $(AddOwner\ own)$, la diferencia entre ambos estados es la inclusión del nuevo titular de datos own en el sistema.

$$Pos\ s\ s'\ ans\ (AddOwner\ own) :=\ s'.own_S\ =\ s.own_S\ \cup\ \{own\}\ \wedge\ ans\ =\ ok$$

En cambio, $(DeleteOwner\ own)$ elimina al titular de datos own, así como todas las asignaciones de consentimientos y los objetos que este posee.

$$\begin{aligned}&Pos\ s\ s'\ ans\ (DeleteOwner\ own) :=\\&\quad s'.own_S\ =\ s.own_S \setminus \{own\}\ \wedge\\&\quad (\forall\ (oc \in OC),\ s'.oc_S\ =\ s.oc_S \setminus \{oc\}\ \wedge\ own \in oc.own_{OC}\ \wedge\ oc \in s.oc_S)\ \wedge\\&\quad (\forall\ (ob \in OBJS),\ \exists\ (ob\_pi \in OBJS\_PI),\\&\qquad s'.ob_S\ =\ s.ob_S \setminus \{ob\}\ \wedge\ own \in objectOwner_S\ (ob\_pi)\ \wedge\ ob \in ob\_pi\ \wedge\ ob \in s.ob_S)\ \wedge\\&\quad ans\ =\ ok\end{aligned}$$

La operación $(AddOwnerConsent\ own\ consent)$ agrega una asignación titular-consentimiento entre el titular de datos own y el consentimiento $consent$.

$$Pos\ s\ s'\ ans\ (AddOwnerConsent\ own\ consent) :=\ s'.oc_S\ =\ s.oc_S\ \cup\ \{(own,\ consent)\}\ \wedge\ ans\ =\ ok$$

En el caso de $(DeleteOwnerConsent\ own\ consent)$, se elimina la asignación entre el titular de datos own y el consentimiento $consent$.

$$Pos\ s\ s'\ ans\ (DeleteOwnerConsent\ own\ consent) :=\ s'.oc_S\ =\ s.oc_S \setminus \{(own,\ consent)\}\ \wedge\ ans\ =\ ok$$

Funciones del sistema:

Para $(AddObjectPersonalInformation\ ob\ pdt\ own)$ un objeto ob pasa a representar información personal, quedando asociado con el identificador de su titular own.

$$\begin{aligned}&Pos\ s\ s'\ ans\ (AddObjectPersonalInformation\ ob\ pdt\ own) :=\\&\quad (\forall\ (ob\_pi \in OBJS\_PI),\ ob \in ob\_pi\ \wedge\ own \in objectOwner_S'\ (ob\_pi))\ \wedge\ ans\ =\ ok\end{aligned}$$

Con $(DropObjectPersonalInformation\ ob\ pdt\ own)$, en cambio, el objeto ob deja de representar información personal y de estar asociado con el identificador del titular own.

$$Pos\ s\ s'\ ans\ (DropObjectPersonalInformation\ ob\ pdt\ own) :=$$
$$(\forall\ (ob\_pi \in OBJS\_PI),\ \neg\ (ob \in ob\_pi)\ \wedge\ \neg\ (own \in objectOwner_S'\ (ob\_pi)))\ \wedge\ ans\ =\ ok$$

Para el comando $(CreateSession\ u\ rl\ sid)$, la diferencia entre ambos estados es la nueva sesión agregada en el sistema para el usuario u, activando a su vez todos los roles pertenecientes al conjunto rl.

$$Pos\ s\ s'\ ans\ (CreateSession\ u\ rl\ sid) :=$$
$$(\exists\ (ar \in ACTIVE\_ROLES),$$
$$s'.sid_S\ =\ s.sid_S\ \cup\ \{sid\}\ \wedge\ ar \in sessionRoles_S'\ (sid)\ \wedge\ rl \in ar\ \wedge\ u \in sessionUser_S'\ (sid))\ \wedge$$
$$ans\ =\ ok$$

$(DeleteSession\ u\ sid)$, en cambio, finaliza la sesión sid del usuario u.

$$Pos\ s\ s'\ ans\ (DeleteSession\ u\ sid) :=\ s'.sid_S\ =\ s.sid_S \setminus \{sid\}\ \wedge\ ans\ =\ ok$$

Con $(AddActiveRole\ u\ sid\ r)$ se añade el rol r a la lista de roles activos de la sesión sid, donde u es el usuario dueño.

$$Pos\ s\ s'\ ans\ (AddActiveRole\ u\ sid\ r) :=$$
$$(\exists\ (ar\ ar' \in ACTIVE\_ROLES),$$
$$ar \in sessionRoles_S\ (sid)\ \wedge$$
$$ar'\ =\ ar\ \cup\ \{r\}\ \wedge$$
$$ar' \in sessionRoles_S'\ (sid)\ \wedge$$
$$u \in sessionUser_S'\ (sid))\ \wedge$$
$$ans\ =\ ok$$

Por otro lado, $(DropActiveRole\ u\ sid\ r)$ elimina el rol r de la lista de roles activos de la sesión sid perteneciente al usuario u.

$$Pos\ s\ s'\ ans\ (DropActiveRole\ u\ sid\ r) :=$$
$$(\exists\ (ar\ ar' \in ACTIVE\_ROLES),$$
$$ar \in sessionRoles_S\ (sid)\ \wedge$$
$$ar'\ =\ ar \setminus \{r\}\ \wedge$$
$$ar' \in sessionRoles_S'\ (sid)\ \wedge$$
$$u \in sessionUser_S'\ (sid))\ \wedge$$
$$ans\ =\ ok$$

El comando $(AddPermissionExercised\ sid\ r\ prm)$ agrega el permiso prm a la lista de permisos ejercidos en la sesión sid asignado al rol r.

$$Pos\ s\ s'\ ans\ (AddPermissionExercised\ sid\ r\ prm) :=$$
$$(\exists\ (ar \in ACTIVE\_ROLES)\ (prm\_ex\ prm\_ex' \in PERMISSIONS\_EXERCISED),$$
$$prm\_ex \in sessionPermissions_S\ (sid)\ \wedge$$
$$prm\_ex'\ =\ prm\_ex\ \cup\ \{prm\}\ \wedge$$
$$prm\_ex' \in sessionPermissions_S'\ (sid)\ \wedge$$
$$ar \in sessionRoles_S\ (sid)\ \wedge$$
$$r \in ar)\ \wedge$$
$$ans\ =\ ok$$

Por el lado de $(DropPermissionExercised\ sid\ r\ prm)$, se elimina el permiso prm de la lista de permisos ejercidos en la sesión sid asignado al rol r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DropPermissionExercised\ sid\ r\ prm) :=\\
&\quad (\exists\ (ar \in ACTIVE\_ROLES)\ (prm\_ex\ prm\_ex' \in PERMISSIONS\_EXERCISED),\\
&\qquad prm\_ex \in sessionPermissions_S\ (sid)\ \wedge\\
&\qquad prm\_ex' = prm\_ex \setminus \{prm\}\ \wedge\\
&\qquad prm\_ex' \in sessionPermissions_S'\ (sid)\ \wedge\\
&\qquad ar \in sessionRoles_S\ (sid)\ \wedge\\
&\qquad r \in ar)\ \wedge\\
&\quad ans = ok
\end{aligned}
$$

En el caso de $(AddPrivacyPermissionExercised\ sid\ r\ pp)$, tiene como objetivo agregar el permiso de privacidad pp a la lista de permisos de privacidad ejercidos en la sesión sid asignado al rol r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (AddPrivacyPermissionExercised\ sid\ r\ pp) :=\\
&\quad (\exists\ (ar \in ACTIVE\_ROLES)\ (pp\_ex\ pp\_ex' \in PP\_EXERCISED),\\
&\qquad pp\_ex \in sessionPrivacyPermissions_S\ (sid)\ \wedge\\
&\qquad pp\_ex' = pp\_ex \cup \{pp\}\ \wedge\\
&\qquad pp\_ex' \in sessionPrivacyPermissions_S'\ (sid)\ \wedge\\
&\qquad ar \in sessionRoles_S\ (sid)\ \wedge\\
&\qquad r \in ar)\ \wedge\\
&\quad ans = ok
\end{aligned}
$$

$(DropPrivacyPermissionExercised\ sid\ r\ pp)$, en cambio, elimina el permiso de privacidad pp de la lista de permisos de privacidad ejercidos en la sesión sid asignado al rol r.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (DropPrivacyPermissionExercised\ sid\ r\ pp) :=\\
&\quad (\exists\ (ar \in ACTIVE\_ROLES)\ (pp\_ex\ pp\_ex' \in PP\_EXERCISED),\\
&\qquad pp\_ex \in sessionPrivacyPermissions_S\ (sid)\ \wedge\\
&\qquad pp\_ex' = pp\_ex \setminus \{pp\}\ \wedge\\
&\qquad pp\_ex' \in sessionPrivacyPermissions_S'\ (sid)\ \wedge\\
&\qquad ar \in sessionRoles_S\ (sid)\ \wedge\\
&\qquad r \in ar)\ \wedge\\
&\quad ans = ok
\end{aligned}
$$

El comando $(CheckAccess\ sid\ op\ ob)$ devuelve ok como respuesta si la sesión sid posee algún rol activo que tenga asignado un permiso conformado por la operación op y el objeto ob. En caso contrario, la respuesta devuelta es $fail$, denegando así la autorización.

$$
\begin{aligned}
&Pos\ s\ s'\ ans\ (CheckAccess\ sid\ op\ ob) :=\\
&\quad s = s'\ \wedge\\
&\quad (\exists\ (r \in ROLES),\ r \in s.r_S\ \wedge\\
&\qquad (\exists\ (ar \in ACTIVE\_ROLES),\ ar \in sessionRoles_S\ (sid)\ \wedge\ r \in ar)\ \wedge\\
&\qquad (((op,\ ob),\ r) \in s.pa_S\ \wedge\ (op,\ ob) \in PERMISSIONS\ \wedge\ ((op,\ ob),\ r) \in PA))\ \wedge\\
&\quad ans = ok
\end{aligned}
$$

Funciones de revisión:

$(AssignedUsers\ r)$ es un comando de revisión cuyo objetivo es retornar la lista de usuarios asignados al rol r sin modificar el estado.

$$
Pos\ s\ s'\ ans\ (AssignedUsers\ r) :=\ s = s'\ \wedge\ ans = user\_list\ (role\_users\ r)
$$

En cambio, $(AssignedRoles\ u)$ es un comando de revisión cuyo objetivo es retornar la lista de roles asignados al usuario u sin modificar el estado.

$$Pos\ s\ s'\ ans\ (AssignedRoles\ u) := s = s' \wedge ans = role\_list\ (user\_roles\ u)$$

En el caso de $(AuthorizedUsers\ r)$, es un comando de revisión que devuelve una lista de usuarios autorizados para el rol r.

$$Pos\ s\ s'\ ans\ (AuthorizedUsers\ r) :=$$
$$s = s' \wedge$$
$$\forall\ (u \in USERS),\ (\exists\ (ul \in set\ USERS),\ (authorizedUser\ s\ u\ r) \wedge u \in ul \wedge ans = user\_list\ (ul))$$

Por otro lado, $(AuthorizedRoles\ u)$ es un comando de revisión que devuelve una lista de roles para los cuales el usuario u está autorizado.

$$Pos\ s\ s'\ ans\ (AuthorizedRoles\ u) :=$$
$$s = s' \wedge$$
$$\forall\ (r \in ROLES),\ (\exists\ (rl \in set\ ROLES),\ (authorizedUser\ s\ u\ r) \wedge r \in rl \wedge ans = role\_list\ (rl))$$

3.4.5. Ejecución de los comandos

Finalmente, se representa la ejecución de un comando f como tal utilizando el tipo inductivo $exec$, que produce un nuevo estado s' a partir de un estado previo s y una respuesta ans. La relación entre estos tres parámetros está dada por las postcondiciones del comando, las cuales fueron especificadas anteriormente:

$$exec : State \rightarrow State \rightarrow Answer \rightarrow Funcs \rightarrow Prop$$

Si la precondición $Pre\ s\ f$ se cumple, entonces el estado resultante s' y la respuesta ans se describen mediante la relación $exec$:

$$exec\_pre\ \frac{Pre\ s\ f \quad Pos\ s\ s'\ ans\ f}{exec\ s\ s'\ ans\ f}$$

En cambio, si $Pre\ s\ f$ no se cumple, el estado s no cambia y la respuesta del sistema es el mensaje de error determinado por la relación $ErrorMsg$:

$$exec\_npre\ \frac{\neg\ (Pre\ s\ f) \quad \exists\ (ec \in ErrorCode),\ s = s' \wedge ErrorMsg\ s\ f\ ec \wedge ans = error\ ec}{exec\ s\ s'\ ans\ f}$$

3.5. Access Request y Access Definition Function

Para concluir la formalización de PCA-RBAC, se representa el $Access\ Request$ como un tipo inductivo, en el cual es necesario distinguir si el acceso solicitado involucra información personal o no. En caso afirmativo, también se debe incluir el propósito del acceso, que se refleja en el siguiente tipo inductivo:

$$\begin{aligned} accessPI :=\ & Purpose\ (prp \in PURPOSES) \\ & |\ No\_Access \end{aligned}$$

A continuación, se presenta el $Access\ Request$ de la siguiente forma:

$$\begin{aligned} & AR\ (s \in State)\ (sid \in SESSIONS)\ (ob \in OBJS)\ (op \in OPERS) :\ accessPI \rightarrow Prop := \\ & \quad |\ AR\_No\_Access : \\ & \qquad sid \in s.sid_S \ \wedge\ (op,\ ob) \in PERMISSIONS \ \wedge\ (op,\ ob) \in s.prm_S \rightarrow \\ & \qquad AR\ s\ sid\ ob\ op\ No\_Access \\ & \quad |\ AR\_Purpose : \\ & \qquad \forall\ (prp \in PURPOSES), \\ & \qquad\quad sid \in s.sid_S \ \wedge\ (op,\ ob) \in PERMISSIONS \ \wedge\ (op,\ ob) \in s.prm_S \rightarrow \\ & \qquad\quad AR\ s\ sid\ ob\ op\ (Purpose\ prp) \end{aligned}$$

Para definir la $Access\ Definition\ Function$, también se utiliza un tipo inductivo. En este caso, también es necesario definir previamente otro conjunto, que corresponde al resultado de aplicar esta función al $Access\ Request$. El resultado puede indicar si el acceso es permitido o es denegado:

$$\begin{aligned} decision :=\ & permit \\ & |\ deny \end{aligned}$$

Luego, la $Access\ Definition\ Function$ se representa como:

$$
\begin{aligned}
&ADF\ (s \in State)\ (sid \in SESSIONS)\ (ob \in OBJS)\ (op \in OPERS) :\ accessPI\ \rightarrow\ decision\ \rightarrow\ Prop := \\
&\quad | \ No\_Access\_permit : \\
&\qquad sid \in s.sid_{\mathsf{S}}\ \wedge \\
&\qquad (\exists\ (prm\_ex \in PERMISSIONS\_EXERCISED), \\
&\qquad (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_{\mathsf{S}}\ \wedge \\
&\qquad prm\_ex \in sessionPermissions_{\mathsf{S}}\ (sid)\ \wedge\ (op,\ ob) \in prm\_ex)\ \rightarrow \\
&\qquad ADF\ s\ sid\ ob\ op\ No\_Access\ permit \\
&\quad | \ No\_Access\_deny : \\
&\qquad sid \in s.sid_{\mathsf{S}}\ \wedge \\
&\qquad \neg\ (\exists\ (prm\_ex \in PERMISSIONS\_EXERCISED), \\
&\qquad (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_{\mathsf{S}}\ \wedge \\
&\qquad prm\_ex \in sessionPermissions_{\mathsf{S}}\ (sid)\ \wedge\ (op,\ ob) \in prm\_ex)\ \rightarrow \\
&\qquad ADF\ s\ sid\ ob\ op\ No\_Access\ deny \\
&\quad | \ Purpose\_permit : \\
&\qquad \forall\ (prp \in PURPOSES), \\
&\qquad\quad (\exists\ (pdt \in PRIVACY\_DATATYPES), \\
&\qquad\quad (\exists\ (pp\_ex \in PP\_EXERCISED), \\
&\qquad\quad sid \in s.sid_{\mathsf{S}}\ \wedge \\
&\qquad\quad (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_{\mathsf{S}}\ \wedge \\
&\qquad\quad ((op,\ ob),\ prp) \in PP\ \wedge\ ((op,\ ob),\ prp) \in s.pp_{\mathsf{S}}\ \wedge \\
&\qquad\quad pp\_ex \in sessionPrivacyPermissions_{\mathsf{S}}\ (sid)\ \wedge\ ((op,\ ob),\ prp) \in pp\_ex)\ \wedge \\
&\qquad\quad ((ob,\ pdt) \in DM\ \wedge\ (ob,\ pdt) \in s.dm_{\mathsf{S}})\ \rightarrow \\
&\qquad\quad \exists\ (ob\_pi \in OBJS\_PI)\ (own \in OWNERS), \\
&\qquad\quad ob \in ob\_pi\ \wedge\ own \in objectOwner_{\mathsf{S}}\ (ob\_pi)\ \wedge\ own \in s.own_{\mathsf{S}}\ \wedge \\
&\qquad\quad (prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_{\mathsf{S}})\ \wedge \\
&\qquad\quad (own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_{\mathsf{S}})\ \rightarrow \\
&\qquad\quad ADF\ s\ sid\ ob\ op\ (Purpose\ prp)\ permit \\
&\quad | \ Purpose\_deny : \\
&\qquad \forall\ (prp \in PURPOSES), \\
&\qquad\quad \neg\ (\exists\ (pdt \in PRIVACY\_DATATYPES), \\
&\qquad\quad (\exists\ (pp\_ex \in PP\_EXERCISED), \\
&\qquad\quad sid \in s.sid_{\mathsf{S}}\ \wedge \\
&\qquad\quad (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_{\mathsf{S}}\ \wedge \\
&\qquad\quad ((op,\ ob),\ prp) \in PP\ \wedge\ ((op,\ ob),\ prp) \in s.pp_{\mathsf{S}}\ \wedge \\
&\qquad\quad pp\_ex \in sessionPrivacyPermissions_{\mathsf{S}}\ (sid)\ \wedge\ ((op,\ ob),\ prp) \in pp\_ex)\ \wedge \\
&\qquad\quad ((ob,\ pdt) \in DM\ \wedge\ (ob,\ pdt) \in s.dm_{\mathsf{S}})\ \rightarrow \\
&\qquad\quad \exists\ (ob\_pi \in OBJS\_PI)\ (own \in OWNERS), \\
&\qquad\quad ob \in ob\_pi\ \wedge\ own \in objectOwner_{\mathsf{S}}\ (ob\_pi)\ \wedge\ own \in s.own_{\mathsf{S}}\ \wedge \\
&\qquad\quad (prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_{\mathsf{S}})\ \wedge \\
&\qquad\quad (own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_{\mathsf{S}})\ \rightarrow \\
&\qquad\quad ADF\ s\ sid\ ob\ op\ (Purpose\ prp)\ deny
\end{aligned}
$$

Capítulo 4: Análisis

En el capítulo anterior, se formalizó el modelo PCA-RBAC. Como parte de su verificación, en este capítulo se analizan un conjunto de propiedades de interés.

En primer lugar, se demuestra la invarianza de la política de privacidad; es decir, partiendo de un estado válido, se prueba que solo se permite el acceso a los datos personales si se cumplen los requisitos necesarios en relación con el propósito y el consentimiento.

Posteriormente, se evalúan algunas propiedades heredadas de la tesina de Cristian Rosa [13], adaptadas al contexto de PCA-RBAC. Estas incluyen la invarianza de la validez de estado respecto a la ejecución de los comandos, la intransferibilidad de las sesiones y el comportamiento adecuado de la jerarquía de roles.

Todas estas propiedades y sus demostraciones se encuentran disponibles, en Coq, en: https://www.fing.edu.uy/~cluna/pca-rbac-coq.v.

4.1. Invarianza de la política de privacidad

Con el objetivo de verificar que se cumplen los requisitos respecto al propósito y consentimiento se utiliza la función ADF definida previamente, la cual debería devolver $permit$ como respuesta al recibir un $Access\ Request$ sobre un estado válido. Es decir, en las propiedades de validez de estado se debe poder encontrar la información necesaria que permita chequear el acceso a datos personales para que ADF devuelva $permit$ y, de este modo, cumplir los requisitos.

La formalización corresponde al siguiente teorema:

Teorema 4.1.1

$privacy\_policy\_invariance$:

$$\forall\ (s \in State)\ (sid \in SESSIONS)\ (ob \in OBJS)\ (op \in OPERS)\ (prp \in PURPOSES),$$
$$Valid\ s\ \wedge\ AR\ s\ sid\ ob\ op\ (Purpose\ prp)\ \rightarrow\ ADF\ s\ sid\ ob\ op\ (Purpose\ prp)\ permit$$

Demostración:

Dado que ADF debe devolver $permit$ para un acceso a datos personales, nos centramos en demostrar únicamente el caso $purpose\_permit$. Es decir, dado el estado del sistema s, una sesión sid, un objeto ob, una operación op y un propósito prp, junto con las propiedades de validez de estado y el $Access\ Request$, se busca probar lo siguiente:

$$
\begin{aligned}
&(\exists\ (pdt \in PRIVACY\_DATATYPES),\\
&\quad (\exists\ (pp\_ex \in PP\_EXERCISED),\\
&\qquad sid \in s.sid_S\ \wedge\\
&\qquad (op,\ ob) \in PERMISSIONS\ \wedge\ (op,\ ob) \in s.prm_S\ \wedge\\
&\qquad ((op,\ ob),\ prp) \in PP\ \wedge\ ((op,\ ob),\ prp) \in s.pp_S\ \wedge\\
&\qquad pp\_ex \in sessionPrivacyPermissions_S\ (sid)\ \wedge\ ((op,\ ob),\ prp) \in pp\_ex)\ \wedge\\
&\quad ((ob,\ pdt) \in DM\ \wedge\ (ob,\ pdt) \in s.dm_S)\ \rightarrow\\
&\quad \exists\ (ob\_pi \in OBJS\_PI)\ (own \in OWNERS),\\
&\qquad ob \in ob\_pi\ \wedge\ own \in objectOwner_S\ (ob\_pi)\ \wedge\ own \in s.own_S\ \wedge\\
&\qquad (prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_S)\ \wedge\\
&\qquad (own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_S)
\end{aligned}
$$

Primero, se puede aplicar la propiedad $(nonStructuralRestriction\ s\ ob)$ definida en la sección 3.3.2 para obtener el tipo de privacidad pdt como testigo. Posteriormente, el antecedente de la implicación quedará como una nueva hipótesis, teniendo que demostrar su consecuente, es decir:

$$
\begin{aligned}
&\exists\ (ob\_pi \in OBJS\_PI)\ (own \in OWNERS),\\
&\quad ob \in ob\_pi\ \wedge\ own \in objectOwner_S\ (ob\_pi)\ \wedge\ own \in s.own_S\ \wedge\\
&\quad (prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_S)\ \wedge\\
&\quad (own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_S)
\end{aligned}
$$

Aplicando la propiedad de validez del estado $(Valid\_objectOwner\ ob)$, se obtiene el objeto que representa información personal (ob_pi) y el titular de los datos (own) como testigos. Luego, se necesitan testigos que representen el consentimiento y la asignación entre el titular y dicho consentimiento. Para ello, se utilizan las propiedades de validez $(existsPdtInConsentAndDM\ prp\ ob)$ para obtener $(prp,\ pdt)$ y $(existsConsentOwner\ (prp,\ pdt))$ para obtener $(own,\ (prp,\ pdt))$, respectivamente. Sin embargo, al aplicar estas propiedades se añaden sus antecedentes como pruebas, es decir, que prp, ob y $(prp,\ pdt)$ pertenecen al estado, quedando las demostraciones de la siguiente manera:

Prueba 1:

$$
\begin{aligned}
&ob \in ob\_pi\ \wedge\ own \in objectOwner_S\ (ob\_pi)\ \wedge\ own \in s.own_S\ \wedge\\
&(prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_S)\ \wedge\\
&(own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_S)
\end{aligned}
$$

Prueba 2:

$$(prp,\ pdt) \in s.consent_S$$

Prueba 3:

$$prp \in s.prp_S\ \wedge\ ob \in s.ob_S$$

Aplicando nuevamente $Valid\_objectOwner$, se pueden demostrar las primeras tres afirmaciones de la Prueba 1, quedando pendiente demostrar lo siguiente:

$$
\begin{aligned}
&(prp,\ pdt) \in CONSENTS\ \wedge\ (prp,\ pdt) \in s.consent_S)\ \wedge\\
&(own,\ (prp,\ pdt)) \in OC\ \wedge\ (own,\ (prp,\ pdt)) \in s.oc_S)
\end{aligned}
$$

Para finalizar la prueba se aplican y usan las propiedades de integridad:
$(Consent\_integrity\ prp\ pdt\ (prp,\ pdt))$ y
$(OC\_integrity\ own\ (prp,\ pdt)\ (own,\ (prp,\ pdt)))$, logrando realizar la demostración

con los antecedentes de éstas, agregando sus consecuentes como nuevas pruebas. Ahora, queda pendiente demostrar lo siguiente:

Prueba 1.1:

$$own \in s.own_{\mathrm{S}} \ \wedge \ (prp,\ pdt) \in s.consent_{\mathrm{S}}$$

Prueba 1.2:

$$prp \in s.prp_{\mathrm{S}} \ \wedge \ pdt \in s.pdt_{\mathrm{S}}$$

Prueba 2:

$$(prp,\ pdt) \in s.consent_{\mathrm{S}}$$

Prueba 3:

$$prp \in s.prp_{\mathrm{S}} \ \wedge \ ob \in s.ob_{\mathrm{S}}$$

La prueba 1.1 se demuestra directamente aplicando $Valid\_objectOwner$ y $Consent\_integrity$. Para probar 1.2, se dividirá la demostración en dos:

Prueba 1.2.1:

$$prp \in s.prp_{\mathrm{S}}$$

Prueba 1.2.2:

$$pdt \in s.pdt_{\mathrm{S}}$$

En 1.2.1, se obtienen los testigos del permiso $(op,\ ob)$ y el permiso de privacidad $((op,\ ob),\ prp)$ a partir de las hipótesis obtenidas de los antecedentes de $purpose\_permit$. Con estos testigos, se puede aplicar la propiedad $(PP\_integrity\ (op,\ ob)\ prp\ ((op,\ ob),\ prp))$ para probar 1.2.1. Luego, se debe demostrar su antecedente, $((op,\ ob),\ prp) \in PP$, lo cual se prueba directamente con los antecedentes de $purpose\_permit$.

Para 1.2.2, se realiza una demostración análoga, pero esta vez obteniendo el testigo $(ob,\ pdt)$, que representa el mapeo entre ob y pdt, para luego utilizarlo en la propiedad de integridad $(DM\_integrity\ ob\ pdt\ (ob,\ pdt))$, que se aplica para probar 1.2.2. Tras esto, se genera la prueba $(ob,\ pdt) \in DM$, la cual también se demuestra con los antecedentes de $purpose\_permit$.

La prueba 2 se resuelve directamente utilizando nuevamente $Consent\_integrity$.

En el caso de la última prueba, la demostración de $prp \in s.prp_{\mathrm{S}}$ se realiza de igual manera que en 1.2.1. Luego, para poder probar $ob \in s.ob_{\mathrm{S}}$, se aplica la hipótesis del $Access\ Request$ para volver a obtener el testigo del permiso $(op,\ ob)$ y así aplicarlo a $(Prm\_integrity\ op\ ob\ (op,\ ob))$, demostrando así la afirmación. Finalmente, se genera la prueba de $(op,\ ob) \in PERMISSIONS$, la cual puede demostrarse directamente con el $Access\ Request$, concluyendo así la demostración.

□

4.2. Otras propiedades relevantes

En esta sección se analizan algunas afirmaciones previamente estudiadas en la tesina [13], aplicándolas a nuestro caso de estudio del modelo PCA-RBAC.
En cuanto al teorema de invarianza de la validez de estado, por razones de brevedad, sólo se demostrarán algunas propiedades para ciertos comandos.

4.2.1. Invarianza de la validez de estado

Este teorema establece que el estado del sistema se mantiene válido sin importar qué comando se ejecute. Es decir, dado el conjunto de propiedades de validez de estado que se cumplen para el estado actual del sistema, si se satisfacen las condiciones para ejecutar los comandos, es decir sus pre y postcondiciones, entonces las propiedades de validez de estado también deben satisfacerse en el nuevo estado que resulte de dicha ejecución. La invarianza de la validez de estado es clave para garantizar otras propiedades relevantes de seguridad del sistema.

Esta propiedad se expresa formalmente de la siguiente manera:

Teorema 4.2.1
$state\_validity\_invariance: \forall\ (s\ s' \in State)\ (f \in Funcs),\ Valid\ s\ \wedge\ Pre\ s\ f\ \wedge\ Pos\ s\ s'\ ok\ f\ \rightarrow\ Valid\ s'$

A continuación, se demuestran varios lemas para ilustrar algunas propiedades que se mantienen invariantes tras la ejecución de ciertos comandos.

El primer lema afirma que, luego de ejecutarse el comando $AddUser$, se obtiene un estado s' en el que se satisface la propiedad $existsSessionOwner$:

Lema 4.2.1.1
$invariance\_existsSessionOwner\_AddUser$:
$\forall\ (s\ s' \in State)\ (u \in USERS),$
$existsSessionOwner\ s\ \wedge\ Pre\ s\ (AddUser\ u)\ \wedge\ Pos\ s\ s'\ ok\ (AddUser\ u)\ \rightarrow$
$existsSessionOwner\ s'$

Demostración:

Expandiendo la definición de $existsSessionOwner\ s'$, se debe probar lo siguiente:

$$\forall\ (sid \in SESSIONS),\ sid \in s'.sid_S\ \rightarrow\ \exists\ (u' \in USERS),\ u' \in s'.u_S\ \wedge\ u' \in sessionUser_S'\ (sid)$$

Tras agregar la sesión sid y el antecedente de la prueba a las hipótesis del lema, se aplica la hipótesis $(existsSessionOwner\ s\ sid)$ para obtener al usuario u' como testigo, y así probar las siguientes dos afirmaciones:

Prueba 1:

$$u' \in s'.u_S\ \wedge\ u' \in sessionUser_S'\ (sid)$$

Prueba 2:

$$sid \in s.sid_S$$

Para demostrar $u' \in s'.u_S$, se debe aplicar la postcondición de $(AddUser\ u')$ y luego distinguir entre dos casos: $u = u'$ y $u \neq u'$

Para el primer caso, se puede aplicar la hipótesis $s'.u_S = s.u_S \cup \{u'\}$, lo que se reduce a probar $u' \in s.u_S \cup \{u'\}$, que es trivialmente cierto.

En el segundo caso, no se estaría agregando un usuario nuevo al sistema, por lo que $s'.u_S = s.u_S$. Como consecuencia, $s.u_S$ se satisface aplicando la hipótesis $(existsSessionOwner\ s)$.

Por otro lado, $u' \in sessionUser_S'\ (sid)$ se mantiene válido, ya que no se producen cambios al aplicar el comando $AddUser$. Es decir,
$sessionUser_S'\ (sid) = sessionUser_S\ (sid)$ y también se satisface
$u' \in sessionUser_S\ (sid)$ al aplicar la hipótesis $(existsSessionOwner\ s)$.

De forma análoga, para la prueba 2, no se generan cambios en las sesiones, por lo que $s'.sid_S = s.sid_S$, lo cual satisface la propiedad al aplicar el antecedente de $(existsSessionOwner\ s')$ que ya fue añadido a las hipótesis del lema.

□

El siguiente lema establece que, tras la ejecución del comando $DeleteUser$ se obtiene un estado s' en el que se sigue cumpliendo $existsSessionOwner$.

Lema 4.2.1.2

$invariance\_existsSessionOwner\_DeleteUser$:

$\forall\ (s\ s' \in State)\ (u \in USERS),$
$existsSessionOwner\ s\ \wedge\ Pre\ s\ (DeleteUser\ u)\ \wedge\ Pos\ s\ s'\ ok\ (DeleteUser\ u) \rightarrow$
$existsSessionOwner\ s'$

Demostración:

Al tratarse de la misma propiedad que en el lema anterior, esta prueba también comienza agregando la sesión sid y la afirmación $sid \in s'.sid_S$ a las hipótesis, generando la siguiente obligación de prueba:

$$\exists\ (u' \in USERS),\ u' \in s'.u_S\ \wedge\ u' \in sessionUser_S'\ (sid)$$

Sin embargo, en este caso se observará que esta última hipótesis es falsa para el caso de $DeleteUser$, por lo que se aplicará la postcondición del comando para buscar alguna contradicción que permita llegar a un absurdo y concluir la demostración.

La postcondición establece que $s'.sid_S = s.sid_S \setminus \{sid\}$. Al sustituir en la hipótesis, se tiene que $sid \in s.sid_S \setminus \{sid\}$, lo cual es trivialmente falso. Dado que una de las hipótesis es falsa, la propiedad se cumple.

Cuando el comando $DeleteUser$ no afecta la sesión, se tiene que $s'.u_S = s.u_S$ y $sessionUser_S'(sid) = sessionUser_S(sid)$, por lo que la demostración sigue el mismo razonamiento que en el lema anterior.

□

A continuación, el siguiente lema establece que al ejecutarse el comando $AddUser$, se obtiene un estado s' en el que se sigue cumpliendo $uniqueSessionOwner$:

Lema 4.2.1.3

$invariance\_uniqueSessionOwner\_AddUser$:

$\forall\ (s\ s' \in State)\ (u \in USERS),$
$uniqueSessionOwner\ s\ \wedge\ Pre\ s\ (AddUser\ u)\ \wedge\ Pos\ s\ s'\ ok\ (AddUser\ u)\ \rightarrow$
$uniqueSessionOwner\ s'$

Demostración:

Expandiendo la definición de $uniqueSessionOwner\ s'$, se tiene:

$$\forall\ (sid \in SESSIONS)\ (u'\ u'' \in USERS),\ sid \in s'.sid_S\ \wedge\ u' \in sessionUser_S'(sid)\ \wedge\ u'' \in sessionUser_S'(sid)\ \rightarrow\ u' = u''$$

Dado que $AddUser$ no modifica ni $s'.sid_S$ ni $sessionUser_S'(sid)$, se obtienen como hipótesis los antecedentes de $uniqueSessionOwner\ s$. A partir de estas hipótesis, se puede demostrar directamente que $u' = u''$, concluyendo así la demostración.

□

Ahora se pone mayor énfasis en los roles, demostrando que al ejecutarse el comando $AddRole$ se obtiene un estado s' donde se sigue satisfaciendo la propiedad $activeSessionRoles$:

Lema 4.2.1.4

$invariance\_activeSessionRoles\_AddRole$:

$\forall\ (s\ s' \in State)\ (r \in ROLES),$
$activeSessionRoles\ s\ \wedge\ Pre\ s\ (AddRole\ r)\ \wedge\ Pos\ s\ s'\ ok\ (AddRole\ r)\ \rightarrow$
$activeSessionRoles\ s'$

Demostración:

En este caso, se debe probar lo siguiente:

$$\forall\ (sid \in Session)\ (u \in USERS)\ (r' \in ROLES),\ \exists\ (ar \in ACTIVE\_ROLES),\ sid \in s'.sid_S\ \wedge\ sessionUser_S'(sid)\ \wedge\ r' \in ar\ \wedge\ ar \in sessionRoles_S'(sid)\ \rightarrow\ authorized\_user\ s'\ u\ r'$$

Aplicando la sesión sid, el usuario u y el rol r' sobre la hipótesis $activeSessionRoles\ s$, se obtiene el rol activo ar como testigo y se agregan los antecedentes de la demostración como nuevas hipótesis. Luego, al aplicar nuevamente $activeSessionRoles\ s$, obtenemos su consecuente en las hipótesis, quedando por demostrar las siguientes dos pruebas.

Prueba 1:

$$authorized\_user\ s'\ u\ r'$$

Prueba 2:

$$sid \in s.sid_S \ \wedge \ sessionUser_S\ (sid) \ \wedge \ r' \in ar \ \wedge \ ar \in sessionRoles_S\ (sid)$$

Para la primera prueba, las definiciones de $authorized\_user\ s'$ y $role\_of\_user\ s'\ r''\ u$ pueden expresarse como:

$$\exists\ (r'' \in ROLES),\ ((u,\ r'') \in UA \ \wedge \ (u,\ r'') \in s.ua_S) \ \wedge \ (r' \preccurlyeq_S r'')$$

Dado que $AddRole$ no modifica ninguna de las afirmaciones necesarias de la prueba, se puede aplicar la hipótesis $authorized\_user\ s$ para demostrarla.

De forma análoga, en la prueba 2 se observa que el comando no altera ninguna de las afirmaciones, por lo que se puede probar directamente utilizando los antecedentes de $activeSessionRoles\ s'$.

□

A continuación, se demuestra que al ejecutarse el comando $DeleteRole$ se obtiene un estado s' donde se mantiene la propiedad $activeSessionRoles$.

Lema 4.2.1.5

$invariance\_activeSessionRoles\_DeleteRole$:

$\forall\ (s\ s' \in State)\ (r \in ROLES),$
$activeSessionRoles\ s \ \wedge \ Pre\ s\ (DeleteRole\ r) \ \wedge \ Pos\ s\ s'\ ok\ (DeleteRole\ r) \ \rightarrow$
$activeSessionRoles\ s'$

Demostración:

Dado que se trata de la misma propiedad que el lema anterior, la demostración sigue el mismo esquema inicial hasta el momento en que se debe probar $authorized\_user\ s'$.

En este caso, dado que $DeleteRole$ modifica algunos de los atributos, se aplicará directamente su postcondición, distinguiendo entre los casos $r = r'$ y $r \neq r'$.

Primer caso: Si $r = r'$, ocurre algo similar al lema 4.2.1.2, ya que el comando establece que $s'.sid_S = s.sid_S \setminus \{sid\}$. Esto se puede aplicar a la hipótesis obtenida de los antecedentes de $activeSessionRoles\ s'$, es decir $sid \in s'.sid_S$, lo que genera la hipótesis falsa $sid \in s.sid_S \setminus \{sid\}$. De esta manera, se concluye la prueba de este caso.

Segundo caso: Para $r \neq r'$, la prueba sigue de forma análoga al lema anterior, ya que no se realizan modificaciones en los atributos del sistema.

□

Finalizando los lemas heredados de RBAC, el siguiente resultado demuestra que al ejecutarse el comando $DeassignUser$, se obtiene un estado s' donde se satisface la propiedad $activeSessionRoles$:

Lema 4.2.1.6

$invariance\_activeSessionRoles\_DeassignUser$:

$\forall\ (s\ s' \in State)\ (u \in USERS)\ (r \in ROLES),$
$activeSessionRoles\ s\ \wedge\ Pre\ s\ (DeassignUser\ u\ r)\ \wedge\ Pos\ s\ s'\ ok\ (DeassignUser\ u\ r)\ \rightarrow$
$activeSessionRoles\ s'$

Demostración:

Esta prueba es análoga al lema anterior, ya que $DeassignUser$ vuelve a generar la hipótesis $s'.\ sid_{\mathrm{S}} = s.\ sid_{\mathrm{S}} \setminus \{sid\}$, lo que conduce al mismo absurdo:
$sid \in s.\ sid_{\mathrm{S}} \setminus \{sid\}$.

□

Los siguientes lemas se enfocan más en el modelo PCA-RBAC. En el primero de ellos se demuestra que al ejecutarse el comando $DeletePurpose$ se obtiene un estado s' donde se satisface la propiedad $existsPdtInConsentAndDM$:

Lema 4.2.1.7

$invariance\_existsPdtInConsentAndDM\_DeletePurpose$:

$\forall\ (s\ s' \in State)\ (prp \in PURPOSES),$
$existsPdtInConsentAndDM\ s\ \wedge$
$Pre\ s\ (DeletePurpose\ prp)\ \wedge$
$Pos\ s\ s'\ ok\ (DeletePurpose\ prp)\ \rightarrow$
$existsPdtInConsentAndDM\ s'$

Demostración:

Expandiendo la definición de $existsPdtInConsentAndDM\ s'$, se debe demostrar lo siguiente:

$\forall\ (prp' \in PURPOSES)\ (ob \in OBJS),$
$prp' \in s'.\ prp_{\mathrm{S}}\ \wedge\ ob \in s'.\ ob_{\mathrm{S}}\ \rightarrow$
$\exists\ (pdt \in PRIVACY\_DATATYPES),\ pdt \in s'.\ pdt_{\mathrm{S}}\ \wedge\ (prp',\ pdt) \in CONSENTS\ \wedge\ (ob,\ pdt) \in DM$

Aplicando el propósito prp' y el objeto ob sobre $existsPdtInConsentAndDM\ s$, se obtiene el tipo de privacidad pdt como testigo y se agrega el antecedente de la demostración como nueva hipótesis. A partir de esto, se generan dos pruebas:

Prueba 1:

$$pdt \in s'.\ pdt_{\mathrm{S}}\ \wedge\ (prp',\ pdt) \in CONSENTS\ \wedge\ (ob,\ pdt) \in DM$$

Prueba 2:

$$prp' \in s.\ prp_{\mathrm{S}}\ \wedge\ ob \in s.\ ob_{\mathrm{S}}$$

Dado que eliminar el propósito no modifica los tipos de privacidad, al aplicar el comando se cumple que $s'.\ pdt_{\mathrm{S}} = s.\ pdt_{\mathrm{S}}$, por lo que se puede probar $pdt \in s'.\ pdt_{\mathrm{S}}$

y el resto de las afirmaciones de la prueba 1 con el consecuente de $existsPdtInConsentAndDM\ s$.

Para la segunda prueba, se aplica $(DeletePurpose\ prp')$, distinguiendo los casos $prp = prp'$ y $prp \neq prp'$. En el primer caso, la precondición del comando establece que $prp \in s.prp_S$, por lo que al cumplirse $prp = prp'$ se puede utilizar esta afirmación para realizar la prueba.
Para el segundo caso, se cumple $s'.prp_S = s.prp_S$, ya que no existe modificación, y se demuestra directamente.
Finalmente, el objeto no se ve afectado por el comando, por lo que su afirmación también puede probarse de forma directa.

□

A continuación, se prueba que al ejecutarse el comando $AddOwner$ se obtiene un estado s' donde se satisface la propiedad $existsConsentOwner$:

Lema 4.2.1.8
$invariance\_existsConsentOwner\_AddOwner$:
$\forall\ (s\ s' \in State)\ (own \in OWNERS),$
$existsConsentOwner\ s\ \wedge\ Pre\ s\ (AddOwner\ own)\ \wedge\ Pos\ s\ s'\ ok\ (AddOwner\ own)\ \rightarrow$
$existsConsentOwner\ s'$

Demostración:

Expandiendo $existsConsentOwner\ s'$, se busca probar lo siguiente:

$$\forall\ (consent \in CONSENTS),\ consent \in s'.consent_S\ \rightarrow$$
$$\exists\ (own' \in OWNERS),\ own' \in s'.own_S\ \wedge\ (own',\ consent) \in OC$$

Al aplicar el consentimiento $consent$ sobre $existsConsentOwner\ s$, se obtiene el titular de datos own' como testigo y se agrega el antecedente de la demostración como nueva hipótesis. Luego, se derivan dos pruebas a realizar:

Prueba 1:

$$own' \in s'.own_S\ \wedge\ (own',\ consent) \in OC$$

Prueba 2:

$$consent \in s'.consent_S$$

Para la prueba 1, se aplica el comando $AddOwner$, obteniendo dos casos a analizar: $own = own'$ y $own \neq own'$.

En el primer caso, se tiene que $s'.own_S = s.own_S \cup \{own'\}$, lo que implica que $own' \in s.own_S \cup \{own'\}$, que se cumple trivialmente.

En el segundo caso, al no añadirse un nuevo titular de datos al sistema, se puede aplicar directamente $existsConsentOwner\ s$.

En ambos casos, la expresión $(own',\ consent) \in OC$ se satisface directamente, ya que el estado del sistema no se ve afectado por el comando.

Para la prueba 2, se observa que el consentimiento no se ve afectado al agregar un titular (own); luego, puede probarse directamente utilizando $existsConsentOwner\ s'$.

□

Para finalizar con los lemas vinculados a la invarianza de la validez de estado, se demostrará que al ejecutarse el comando $DropObjectPersonalInformation$ se obtiene un estado s' donde se satisface $Valid\_objectOwner$:

Lema 4.2.1.9

$invariance\_Valid\_objectOwner\_DropObjectPersonalInformation$:

$\forall\ (s\ s' \in State)\ (ob \in OBJS)\ (pdt \in PRIVACY\_DATATYPES)\ (own \in OWNERS),$
$Valid\_objectOwner\ s\ \wedge$
$Pre\ s\ (DropObjectPersonalInformation\ ob\ pdt\ own)\ \wedge$
$Pos\ s\ s'\ ok\ (DropObjectPersonalInformation\ ob\ pdt\ own)\ \rightarrow$
$Valid\_objectOwner\ s'$

Demostración:

En este caso, se busca demostrar lo siguiente:

$$\forall\ (ob' \in OBJS),\ \exists\ (ob\_pi \in OBJS\_PI)\ (own' \in OWNERS),$$
$$ob' \in ob\_pi\ \wedge\ own' \in s'.own_S\ \wedge\ own' \in objectOwner_S'\ (ob\_pi)$$

Para esta prueba, las primeras dos afirmaciones se resuelven directamente aplicando $Valid\_objectOwner\ s$, pues en la primera no se involucra el estado y en la segunda el comando no lo modifica.

Para demostrar $own' \in objectOwner_S'\ (ob\_pi)$, se aplica la postcondición del comando. En el caso de igualdad, se deduce que $\neg\ (ob' \in ob\_pi)$. Sin embargo, según la propiedad de validez para el estado s, se tiene que $ob' \in ob\_pi$, lo cual resulta en un absurdo, concluyendo así la prueba. En el caso de desigualdad, como es habitual, el comando no afecta las afirmaciones.

□

4.2.2. Intransferibilidad de las sesiones

Este lema, también heredado de RBAC, establece que a partir de un estado válido s, un usuario u y una sesión sid, si u es dueño de la sesión sid en dicho estado y se ejecuta exitosamente cualquiera de los comandos f, se obtiene un nuevo estado s' en el cual u sigue siendo el dueño de sid.

Lema 4.2.2

$non\_transferibility\_of\_sessions$:

$$\forall\ (s\ s' \in State)\ (u \in USERS)\ (sid \in SESSIONS)\ (f \in Funcs),$$
$$sid \in s.sid_S\ \wedge\ u \in sessionUser_S\ (sid)\ \wedge\ exec\ s\ s'\ ok\ f\ \wedge\ sid \in s'.sid_S\ \rightarrow$$
$$u \in sessionUser_S'\ (sid)$$

Demostración:

Para probar este lema, se realiza un análisis de casos respecto a los comandos (f).

Para todos los comandos administrativos y muchas de las funciones del sistema, la afirmación $u \in sessionUser_S'\ (sid)$ no se ve afectada, ya que se cumple que $sessionUser_S'\ (sid) = sessionUser_S\ (sid)$. Por lo tanto, dicha afirmación queda demostrada directamente al encontrarse $u \in sessionUser_S\ (sid)$ en las hipótesis.

Para los comandos donde no se cumple la igualdad mencionada, es decir, en $CreateSession$, $AddActiveRole$ y $DropActiveRole$, se aplica sus respectivas postcondiciones, distinguiendo los casos de igualdad y desigualdad para sus atributos más relevantes (en los tres casos, sid respecto a sid').

En el primer caso, se puede observar que para los tres comandos se satisface directamente la afirmación $u \in sessionUser_S'\ (sid')$, concluyendo así la prueba, dado que $sid = sid'$.

El segundo caso es análogo al de los comandos anteriores, al cumplirse: $sessionUser_S'\ (sid) = sessionUser_S\ (sid)$.

Finalmente, para el caso de $CheckAccess$ y las funciones de revisión, así como el caso que no se cumplen las precondiciones de f $(exec\_npre)$, se verifica que $s' = s$, y en consecuencia la demostración es trivial.

□

Desde el punto de vista de seguridad es un resultado importante, que se preserva luego de las extensiones propuestas en el presente trabajo, ya que la posible transferencia de las sesiones sería eventualmente un problema del modelo.

4.2.3. Comportamiento correcto de la jerarquía de roles

Para concluir con las demostraciones, se analiza una propiedad respecto a la corrección de la jerarquía de roles. Esta propiedad afirma que no puede ocurrir que un usuario asignado a roles de mayor jerarquía que otro, no esté autorizado para un rol, mientras que el segundo sí lo esté. Su especificación y prueba son las siguientes:

Lema 4.2.3

$correctness\_of\_role\_hierarchy$:

$$\forall\ (s \in State)\ (u\ u' \in USERS)\ (r\ r'' \in ROLES),$$
$$(\forall\ (r' \in ROLES),\ role\_of\_user\ s\ r'\ u\ \wedge\ r'\ \preccurlyeq_S\ r'') \wedge authorizedUser\ s\ u'\ r'' \rightarrow$$
$$\neg\ (authorizedUser\ s\ u\ r\ \wedge\ \neg\ authorizedUser\ s\ u'\ r)$$

Demostración:

Primero, se expande la definición de $(authorizedUser\ s\ u\ r)$ para simplificar:

$$\exists\ (r' \in ROLES),\ (role\_of\_user\ s\ r'\ u)\ \wedge\ (r\ \preccurlyeq_S\ r')$$

Dado que el consecuente del lema a demostrar es una negación, esto equivale a demostrar $False$ al agregar $(authorizedUser\ s\ u\ r\ \wedge\ \neg\ authorizedUser\ s\ u'\ r)$ a las hipótesis.

Aplicando estas dos afirmaciones —primero $(\neg\ authorizedUser\ s\ u'\ r)$ y luego $(authorizedUser\ s\ u\ r)$— y utilizando el rol testigo r' obtenido de la hipótesis $(authorizedUser\ s\ u'\ r'')$, queda como objetivo de prueba $authorizedUser\ s\ u'\ r$. Expandiéndolo, se obtiene:

$$(role\_of\_user\ s\ r'\ u')\ \wedge\ (r\ \preccurlyeq_S\ r')$$

Para probar $(role\_of\_user\ s\ r'\ u')$, se puede aplicar directamente la hipótesis $(role\_of\_user\ s\ r'\ u')$ que se encuentra en la definición de $authorizedUser\ s\ u'\ r''$.

Luego, para demostrar $(r\ \preccurlyeq_S\ r')$, se usarán como hipótesis: $(r'\ \preccurlyeq_S\ r'')$, obtenida de $(authorizedUser\ s\ u'\ r'')$, y $(r\ \preccurlyeq_S\ r'')$, obtenida de
$(\forall\ (r \in ROLES),\ role\_of\_user\ s\ r\ u\ \wedge\ r\ \preccurlyeq_S\ r'')$, donde se aplica el rol r en el $\forall$.

Teniendo esto, se aplica la propiedad transitiva sobre la relación $(r\ \preccurlyeq_S\ r')$, demostrando así $(r'\ \preccurlyeq_S\ r'')$ y $(r\ \preccurlyeq_S\ r'')$. Luego, el lema queda probado.

□

Capítulo 5: Conclusiones y Trabajo Futuro

Este proyecto de grado está fuertemente basado en la tesis de grado propuesta por Guillermo Rodríguez, donde después de analizar el concepto de privacidad y de relevar requerimientos a partir de la Ley 18.331 de Protección de Datos Personales, se presenta de manera semi-formal el modelo PCA-RBAC [1]. A su vez, toma inspiración de la tesina de grado escrita por Cristian Daniel Rosa, quien logró obtener un análisis detallado del modelo RBAC [13]. Aplicando la metodología utilizada en [13], en el presente trabajo se ha conseguido realizar una especificación formal de PCA-RBAC a partir del CCI y el asistente de pruebas Coq.

Tomando en cuenta la formalización de RBAC, se definió la estructura de PCA-RBAC considerando los elementos y relaciones que extienden al primero. Se definió además un estado del sistema con el fin de capturar un momento determinado y válido del modelo, y se añadieron comandos administrativos, así como funciones del sistema y de revisión, para poder estudiar el comportamiento del modelo al ejecutarse dichas acciones, conceptualizando formalmente así una máquina de estados.

Por otro lado, utilizando las propiedades de validez de estado del sistema, junto con las precondiciones y, sobre todo, las postcondiciones de los comandos definidos sobre el modelo, se consiguió probar algunas propiedades esenciales referentes a PCA-RBAC, que contribuyen a su estudio y comprensión. Se destaca entre ellas la verificación de que *se cumplan los requisitos necesarios respecto al propósito y al consentimiento para poder tener el permiso de acceso a los datos personales, siempre y cuando el estado del sistema sea válido*; con esto se demuestra que la política de privacidad se mantiene invariante. De forma complementaria, el análisis de la invarianza de determinadas propiedades de validez de estado al ejecutar algunos comandos, junto a la validez de los lemas de intransferibilidad de las sesiones y la corrección de la jerarquía de roles, son de gran utilidad para comprobar corrección del modelo, de manera rigurosa.

Este trabajo deja abiertas líneas y actividades como trabajo futuro.
En primer lugar, se podría realizar un análisis más exhaustivo de las regulaciones actuales para proteger datos personales, con el objetivo de complementar el trabajo ya realizado por Guillermo Rodríguez respecto a la Ley 18.331. Esto permitiría evaluar si es posible obtener más y mejores requisitos, de forma tal que se pueda potencialmente mejorar el modelo ya especificado, o proponer algún otro que resulte eventualmente más adecuado para la temática.
Por un lado, se sugiere estudiar la viabilidad de representar el principio de veracidad expresado en la Ley (Art. 7), donde se requiere que los datos personales sean verídicos y exactos, así como de un responsable del tratamiento (definido en el Art. 4 literal H), que debe suprimir estos datos en caso de no cumplirse lo anterior. Una posibilidad es incluir a este responsable en el modelo y vincularlo a los objetos que representan información personal mediante una función que chequee su exactitud y veracidad. Esto resulta desafiante, pues habría que establecer una lista de puntos que especifiquen qué hace de una información veraz y exacta.
A su vez, se podría evaluar la posibilidad de representar la información sensible (Art. 18), distinguiéndose de la información personal ya expresada en el modelo. Para

ello es fundamental definir consentimientos acordes a este tipo de datos, vinculados al titular de dichos datos. Estos consentimientos se componen de un nuevo tipo de privacidad específico para la información sensible y de un propósito legal, el cual se podría representar mediante un tipo de propósito distinto al ya definido o alternativamente se podría incorporar un atributo en los propósitos donde se distingan entre información personal y sensible.

Por otro lado, aunque fue posible realizar una representación formal de PCA-RBAC, ésta no contempla la expresión matemática de la política de privacidad. Si bien los consentimientos se representan a partir de un propósito y un tipo de privacidad, no se considera que éstos podrían ser obligatorios, opcionales, o directamente no utilizados. Para ello, podría definirse un tipo inductivo $clasify\_consent$ y una función parcial $policy$ de la siguiente forma:

$$\begin{aligned} clasify\_consent := \ & Not\_used \\ & | \ OptionalRejected \\ & | \ OptionalAccepted \\ & | \ Compulsory \end{aligned}$$

$$policy : \ PURPOSES \ \rightarrow PRIVACY\_DATATYPES \ \rightarrow \ option \, clasify\_consent$$

A partir de lo anterior, debería redefinirse el elemento $CONSENTS$ y evaluar si algún comando o propiedad puede verse afectada.
En particular, al definir las precondiciones del comando $GrantConsent$, se debe considerar el resultado de aplicar la función $policy$ sobre el propósito prp y el tipo de privacidad pdt. Para que se cumpla la precondición, la función debe devolver $Compulsory$ u $OptionalAccepted$.
A su vez, este chequeo debe incluirse en las propiedades de validez de estado donde aparezcan el consentimiento junto al propósito y tipo de privacidad, al igual que en el teorema de la invarianza de la política de privacidad.

A modo de conclusión final del proyecto, se puede destacar la dificultad que supone realizar una representación formal de un modelo con la rigurosidad que imponen CCI y Coq. La formalización Coq contempla con 5721 líneas en total. Asimismo, resulta un desafío tener en cuenta todos los aspectos de la realidad para obtener una mayor completitud en la representación y análisis de posibles propiedades de interés. No obstante, se ha logrado un estudio exhaustivo de PCA-RBAC, abriendo la posibilidad de continuar realizando mejoras y seguir aproximándose a un modelo ideal de relevancia, dada la importancia de la temática.

Referencias

[1] Guillermo Rodríguez. “Formalización y validación de consentimiento para el control de acceso a datos personales". Instituto de Computación, Facultad de Ingeniería, Universidad de la República, 2018.

[2] Alan F Westin. “Privacy and freedom". En: *Washington and Lee Law Review*, 25.1 (1968), pág. 166.

[3] Daniel J Solove. “A taxonomy of privacy". En: *University of Pennsylvania Law Review*, 154 (2005), pág. 477.

[4] Daniel J Solove. “Why privacy matters even if you have ‘nothing to hide'". En: *Chronicle of Higher Education*, 15 (2011).

[5] Eugene H Spafford y Annie I Antón. “The balance of privacy and security". En: *Controversies in Science and Technology - Volume 2: From Climate to Chromosomes* (2007), Cap. 8, págs. 152-168.

[6] Poder Legislativo. Ley No 18.331. *Protección de Datos Personales*. Montevideo, Uruguay, 2008.

[7] Ann Cavoukian. “Privacy By Design: The Seven Foundational Principles". En: *Information and Privacy Commissioner of Ontario*, Canada, 2010.

[8] Seda Gürses, Carmela Troncoso y Claudia Diaz. “Engineering privacy by design". En: *Conference on Computers, Privacy and Data Protection*, 2011.

[9] David Ferraiolo, Janet Cugini y D Richard Kuhn. “Role-based access control (RBAC): Features and motivations". En: *Proceedings of 11th annual computer security application conference* (1995), págs. 241-248.

[10] D. Ferraiolo y R. Kuhn. “Role-based access controls”. En: *Proceedings of 15th NIST-NCSC National Computer Security Conference* (1992), págs 554-563.

[11] R. Sandhu, D. Ferraiolo, y R. Kuhn. “The nist model for role-based access control: Towards a unified standard”. En: *Proceedings of the fifth ACM workshop on Role-based access control*. ACM Press, 2000, págs 47-63.

[12] David F Ferraiolo y col. “Proposed NIST standard for role-based access control". En: *ACM Transactions on Information and System Security (TISSEC)*, 4.3 (2001), págs. 224-274.

[13] Cristian Daniel Rosa. “Especificación Formal del Modelo RBAC en el Cálculo de Construcciones Inductivas". Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad Nacional de Rosario, 2008.

[14] T. Coquand y G. P. Huet. “The calculus of constructions”. *Inf. Comput.*, Vol 76 (Issues 2-3) (1988), págs 95-120.

[15] P. Martin-Löf y G. Mints, editores. *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988, Proceedings.* Volumen 417 de “Lecture Notes in Computer Science”. Springer. 1990.

[16] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project. 2022, versión 8.15.2.

[17] Y. Bertot and P. Casteran. “Interactive theorem proving and program development. coqart: The calculus of inductive constructions”, 2004.

[18] Amirreza Masoumzadeh y James BD Joshi. “PuRBAC: Purpose-aware role-based access control". En: *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*. Springer. 2008, págs. 1104-1121.

[19] Qun Ni y col. “Privacy-aware role-based access control". En: *ACM Transactions on Information and System Security (TISSEC)*, 13.3 (2010), pág. 24.

[20] Calvin S Powers, Paul Ashley y Matthias Schunter.”Privacy promises, access control, and privacy management. Enforcing privacy throughout an enterprise by extending access control". En: *Electronic Commerce, 2002. Proceedings. Third International Symposium on*, IEEE. 2002, págs. 13-21.

[21] Mohammad Jafari y col. “A framework for expressing and enforcing purpose-based privacy policies". En: *ACM Transactions on Information and System Security (TISSEC)*, 17.1 (2014), pág. 3.

[22] S. Z. Beguelin. “Especificación formal del modelo de seguridad de MIDP 2.0 en el Cálculo de Construcciones Inductivas”. Tesis de maestría, Universidad Nacional de Rosario, mayo de 2006.