Technical Report

UCAM-CL-TR-628
ISSN 1476-2986

Number 628

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

A formal security policy for an NHS electronic health record service

Moritz Y. Becker

March 2005

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

A formal security policy for an NHS electronic health record service

Moritz Y. Becker
Computer Laboratory, University of Cambridge
JJ Thomson Avenue, Cambridge, United Kingdom
moritz.becker@cl.cam.ac.uk

Abstract

The ongoing NHS project for the development of a UK-wide electronic health records service, also known as the 'Spine', raises many controversial issues and technical challenges concerning the security and confidentiality of patient-identifiable clinical data. As the system will need to be constantly adapted to comply with evolving legal requirements and guidelines, the Spine's authorisation policy should not be hard-coded into the system but rather be specified in a high-level, general-purpose, machine-enforceable policy language.

We describe a complete authorisation policy for the Spine and related services, written for the trust management system Cassandra, and comprising 375 formal rules. The policy is based on the NHS's Output-based Specification (OBS) document and deals with all requirements concerning access control of patient-identifiable data, including legitimate relationships, patients restricting access, authenticated express consent, third-party consent, and workgroup management.

1 Background

Electronic health record (EHR) schemes are now being developed in Europe, the United States, Canada and Australia to provide "cradle-to-grave" summaries of patients' records, linking clinical information across the entire health system [Cor02]. In the UK, the National Programme for IT (NPfIT) will bring IT technology into the NHS to modernise patient care and services. The NHS has been planning since 1998 to develop a nationwide Integrated Care Records Service (ICRS) providing health care providers and patients with 24 hour on-line access to EHRs on a central data-spine [Nat98, Dep01a]. The deployment of what is now the world's largest and one of the most complex IT projects in the public sector is scheduled for between 2005 and 2010. While the potential benefits of such a system are obviously huge, the government's plans have also been subject of a fiercely controversial public debate (see e.g. [Haw03, Rog03, CS03, Col03a, Pal03, Col03b, Cro03, Fou03, Arn03, Ley04, Col04]).

A main issue of concern is the confidentiality of patient health information: according to a recent survey conducted by the Consumers' Association, 72 percent of the respondents said security and confidentiality are a primary concern [Arn03]. At stake is not just the privacy of sensitive personal information but the success of the entire project. Patients will refuse to share their data if they do not trust the system or do not have sufficient control over the use of their data. Already, public confidence is eroded: with the ICRS project, the NHS won a "Most Heinous Government Organisation" Big Brother Award in 2000 and "Most Appalling Project" Big Brother Award in 2004 [Ley04]. It is equally important to gain clinician buy-in which will fail if the system is cumbersome to use, if the access restrictions are too strict or the response times too high. Indeed, the clinicians' confidence in the project is already eroded as well. At the annual meeting of the British Medical Association in June 2004, delegates voted for a motion which said, "given the uncertainties and lack of consultation on the Care Records Service [and] until GPs' legitimate concerns are answered, GPs should not engage with the Care Records Service" [Col04].

Any EHR access control policy will be a compromise between conflicting interests. The general framework for any such policy has to comply with relevant legislation and regulations

such as the Data Protection Act, Mental Health Act, Human Fertilisation and Embryology Act, the Abortion Regulations and the Venereal Diseases Regulations. Every health organisation is now required to have a privacy and data protection officer, also called "Caldicott Guardian" (named after the recommendation of the committee chaired by Dame Caldicott in 1997 [Nat97]), who is responsible for overseeing the organisation's security policy and investigating breaches of confidentiality. Anything beyond these requirements is still highly contentious. As an example, the Caldicott committee recommended that access to patient-identifiable information should be on a "strict need-to-know basis". In contrast, common medical code of ethics and professional practice goes further and requires the patient's consent for accessing personal information [SMW93]. Anderson [And96] stresses the same point in his security policy commissioned by the British Medical Association, and further demands that patients should automatically receive notifications when their data are accessed. It now seems to be current consensus that patient consent should be the basis for access decisions although it is not yet clear when explicit consent has to be sought and when implicit consent can be assumed.

It should be sufficiently clear from this that the Spine's access control policy will undergo frequent changes as the public debate evolves. Furthermore, health organisations are likely to have customised policies, compatible with but different from the national one. It is therefore necessary to be able to specify the policy independently of the implementation instead of having it hard-coded into the access control engine.

In the trust management approach [BFL96], an organisation's policy is specified explicitly in a high-level policy language. Access requests are submitted along with supporting digital credentials to prove that the request complies with the local policy. Strangers thus establish mutual trust by exchanging sets of suitable credentials. This idea is common in everyday life: for example, a passenger can check in and request a boarding pass on production of a passport and a flight ticket.

We have produced a complete EHR policy with 375 formal rules, based mainly on the Output Based ICRS Specification [Nat03]. It is written in Cassandra [BS04a, BS04b], a formally specified distributed trust management system with a policy language that is unique in that its expressiveness can be flexibly tuned according to need. §2 introduces the Cassandra trust management system and its policy language. A brief overview of the EHR architecture is given in §3. §4 describes a complex scenario illustrating some of the challenging security requirements and how patients and clinicians may use such a system in practice. §5 describes our NHS policy in detail. The formal policy rules can be found in Appendix A.

2 Cassandra overview

Cassandra is a trust management system enabling a potentially large network of entities to share their resources under well-defined restrictions, specified by local access control policies, even if they are mutual strangers. This section gives a brief and informal overview of the system and illustrates it with examples; see [BS04b, BS04a] for a more formal treatment. §2.1 describes the main components of Cassandra from a high-level point of view; §2.2 outlines the policy language and §2.3 the access control engine.

2.1 Architectural overview

Imagine a network of entities (human users, organisations, autonomous programs) who would like to collaborate with each other. Every entity runs its own copy of a Cassandra service, which acts as a protective layer around the resources. Figure 1 shows the internal components of a Cassandra service. Entities can interact with each other only via the interface. The design goal of the interface was simplicity, orthogonality and generality. Consequently, the interface defines only the most essential and basic requests relevant to role-based trust management: performing an action (i.e. accessing a resource), activating and deactivating a role, and requesting a credential that perhaps is needed to gain authorisation somewhere else.

The *access control engine* handles such a request by invoking the *evaluation engine*, which

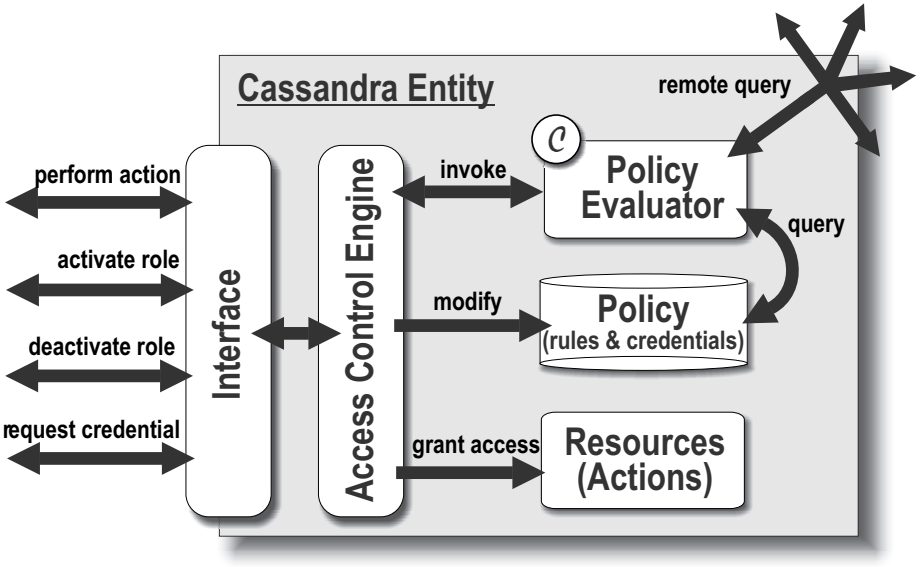

Figure 1: Cassandra components.

in turn queries the local Cassandra policy. The expressiveness of the policy specification language depends on the globally chosen constraint domain $\mathcal{C}$, an independent module that is plugged into the query evaluation engine. Cassandra is designed to support automated credential retrieval: a policy of one entity can refer to policies of other entities — query evaluation may thus trigger credential requests from some other entity's policy (possibly the requester's) over the network. The answer of the evaluation engine is used by the access control engine to decide whether the request is to be granted.

2.2 Policy specification overview

Entities protect their resources by specifying a policy. In Cassandra, a policy is a set of rules written in the Cassandra policy language. Rules govern the access control behaviour of the system; in particular, they specify role membership, permissions to perform actions, and the conditions for role activation, role deactivation and for disclosing credentials. This section outlines the policy language: it first introduces Datalog$^{\mathcal{C}}$ and then sketches how Datalog$^{\mathcal{C}}$ is extended with special predicates and trust management constructs.

2.2.1 Datalog$^{\mathcal{C}}$

Cassandra's policy specification language is based on Datalog$^{\mathcal{C}}$, a language known from Constraint Logic Programming (CLP). Datalog$^{\mathcal{C}}$ is a generic extension of negation-free Datalog [AHV95] (Prolog without function symbols) where the language expressiveness can be tuned by varying the constraint domain parameter $\mathcal{C}$. A Datalog$^{\mathcal{C}}$ rule is of the form

$$p_0(\vec{e}_0) \leftarrow p_1(\vec{e}_1), .., p_n(\vec{e}_n), c$$

where the p_i are predicate names and the $\vec{e}_i$ are (possibly empty) expression tuples (that may contain variables) matching the parameter types of the predicate. $p_0(\vec{e}_0)$ is the *head* of the rule, and the sequence of predicates on the right hand side of the arrow is the *body* of the rule; c is a *constraint* on the parameters occurring in the rest of the rule. Intuitively, to deduce the head of a rule, all body predicates must be deducible in such a way that the constraint is also satisfied. A set of Datalog$^{\mathcal{C}}$ rules can then be interpreted as the deductive closure of the set.

The constraint of a rule, c, is a formula from some fixed *constraint domain* $\mathcal{C}$, a language of first order formulae containing at least true, false and the identity predicate "=" between

$\mathcal{C}$-expressions (variables, entities and possibly other constructs). It must be closed under variable renaming, conjunction ($\wedge$) and disjunction ($\vee$). Furthermore, it must be equipped with an interpretation that defines when formulae are satisfied.

The expressiveness of Datalog$^{\mathcal{C}}$ depends on the chosen constraint domain $\mathcal{C}$. For example, the least expressive constraint domain is the one where the only atomic constraints are equalities between variables and constants. Choosing this trivial constraint domain reduces the expressiveness of the language to standard Datalog or Horn clauses without function symbols. More powerful constraint domains often include boolean, arithmetic and set constraints, and make use of more complex expressions such as tuples, set expressions and (side-effect free) function applications (e.g. to access the current time). The computational complexity of evaluating Datalog$^{\mathcal{C}}$ programs increases with increasing expressiveness: with set constraints it is already possible to encode the Hamiltonian cycle problem, and thus all NP-complete problems. Care must be taken not to choose a constraint domain that is too expressive as this can result in programs in which queries are undecidable. We will later introduce the notion of *constraint compactness* to restrict constraint domains to those that guarantee termination of queries. A more formal treatment of Datalog$^{\mathcal{C}}$ and CLP can be found in [JM94, JMMS98, Rev02]

2.2.2 Roles and actions

In Cassandra, access control is role-based, and our roles, as well as actions, are parameterised. Role-based access control (RBAC) was initially introduced to simplify security administration of large enterprises. The use of role parameters enhances language expressiveness, and furthermore can dramatically reduce the number of required roles in a policy. In the context of distributed trust management, roles can more generally be used as a representation of authenticated subject attributes in decentralised access control [LMW02].

Formally, a *role* is a typed role name applied to an expression (that may contain variables) of a matching type, e.g. `Manager`(`Sales-dept`). Similarly, an *action* is an action name applied to an expression, e.g. `Read-file`(*file*). For the remainder of the dissertation, variables will be written in small letters and italics (e.g. *file*), generic constants in italics but capitalised (e.g. some entity E), and concrete constants in typewriter font (e.g. `Sales-dept`).

2.2.3 Predicates and rules

Policies are specified by rules defining predicates that govern access control decisions. Recall that the interface defines requests for the four most basic tasks of performing an action, activating and deactivating a role, and requesting a credential. Cassandra's predicates were designed to express conditions for precisely these four requests. Consequently, we have a predicate permits that defines who can perform which action; canActivate specifies who can activate which roles (and thus implicitly defines the role membership relation); canDeactivate specifies who can revoke which role; isDeactivated is used to define automatically triggered role revocation; and canReqCred rules specify the conditions to be satisfied before the service is willing to issue and disclose a credential. We also have the predicate hasActivated that specifies who has activated which role. The reason why we need this predicate is that role activations are reflected in the policy; in other words, if an entity activates a role, a new hasActivated statement is added to the policy; and conversely, the statement is removed from the policy when the role is deactivated.

It may come as a surprise that requests modify the policy. Indeed, in a real implementation, hasActivated predicates would likely be stored separately from the policy for efficiency reasons. However, viewing the current role activations as a part of what the policy asserts simplifies our model, as the entire state of the system (i.e., which roles are activated) is then captured by the policies alone. Moreover, the conditions in a rule are often concerned with whether somebody *has* activated a role, and sometimes whether somebody *can* activate a role (i.e., is a member of the role). Similarly, as we shall see in the next section, credentials are just signed predicates, and are often used for asserting that somebody *has* activated a role,

or that somebody *can* activate a role. Therefore, expressing both role activation and role membership in form of predicates is a logical design decision that keeps the model uniform.

Finally, user-defined predicates are also allowed. These predicates are auxiliary and do not have any inherent access control meaning.

The following simple examples of policy rules illustrate the meanings of some of these special predicates. This rule specifies that human resource managers who have activated their role can register new employees in any department, apart from executive board members.

$$\begin{aligned}&\mathsf{permits}(e, \texttt{Register-employee}(name, dept)) \leftarrow \\ &\quad \mathsf{hasActivated}(e, \texttt{Manager}(\texttt{HR})), \\ &\quad dept \neq \texttt{executive-board}\end{aligned}$$

The next policy fragment consists of three rules. The first rule allows Alice and Bob to activate the administrator role if their user roles have been activated; the second one allows everyone to deactivate their own user role; and the third rule specifies that an administrator role is automatically deactivated if the corresponding user role is deactivated.

$$\begin{aligned}&\mathsf{canActivate}(e, \texttt{Admin}()) \leftarrow \\ &\quad \mathsf{hasActivated}(e, \texttt{User}()), \\ &\quad e \in \{\texttt{Alice}, \texttt{Bob}\}\end{aligned} \tag{1}$$

$$\mathsf{canDeactivate}(e_1, e_2, \texttt{User}()) \leftarrow e_1 = e_2$$

$$\begin{aligned}&\mathsf{isDeactivated}(e, \texttt{Admin}()) \leftarrow \\ &\quad \mathsf{isDeactivated}(e, \texttt{User}())\end{aligned}$$

Note that the second rule could have been written more concisely as

$$\mathsf{canDeactivate}(e, e, \texttt{User}())$$

The first argument of every special predicate is an explicit *subject* parameter; for example, in the case of canDeactivate, the first parameter specifies *who* can perform the deactivation. In the simple rules shown above, the subject in the head is always the same as in the body predicates. However, there are rules where this is not the case. These rules cannot be easily expressed in languages where the subjects of head and body are implicitly the same.

The Cassandra policy language also allows restricted aggregation operators for computing the set of all different values that satisfy a predicate. An *aggregation rule* is a rule whose head contains one of the two operators group and count. Intuitively, the group operator finds the set of all different ground values that satisfy a predicate, whereas count computes the size of that set. For example, consider the following rule:

$$\begin{aligned}&\mathsf{countMonkeys}(\mathsf{count}\langle x\rangle, age) \leftarrow \\ &\quad \mathsf{hasActivated}(x, \texttt{Monkey}(age))\end{aligned}$$

Querying the policy with the query $\mathsf{countMonkeys}(n, 5)$ would find the number n of active monkeys of age 5.

2.2.4 Credentials and distributed policies

In the trust management approach, access control decisions are based on credentials asserting properties about the holders. In Cassandra, the properties asserted by credentials are constrained predicates. Furthermore, a credential is signed and issued by an entity called *issuer*, and is held by an entity called *location*. We write

$$\begin{aligned}&\texttt{Alice@UCam}.\mathsf{canActivate}(\texttt{Alice}, \texttt{Student}(subj)) \leftarrow \\ &\quad subj \in \{\texttt{Maths}, \texttt{Psychology}\}\end{aligned}$$

to represent a credential held by Alice, issued by the University of Cambridge (UCam), and asserting that Alice is a Maths and Psychology student.

In Cassandra, every predicate in a rule has a location prefix and an issuer prefix. This enables rules to refer to predicates that are stored somewhere else on the network and are signed by someone else. In other words, rules can refer to credentials. As in the rule examples from the previous section, we usually omit those prefixes that refer to the location of the rule itself. The last example written out with all prefixes would have read (assuming that it is stored in E's policy):

$$E@E.\mathsf{isDeactivated}(e, \mathtt{Admin}()) \leftarrow \\ E@E.\mathsf{isDeactivated}(e, \mathtt{User}())$$

The following example illustrates how Cassandra deals with predicates that refer to remote locations. The first rule belongs to the policy of a bookshop called Heffers which offers discounts to Physics, Maths and Computer Science students from UCam and the Anglia Polytechnic University (APU). If Alice sends a request to Heffers to activate a discount role, the bookshop's Cassandra service will send a credential request back to Alice (since $e_1 = \mathtt{Alice}$) to ask her for an appropriate student credential issued by either UCam or APU.

$$\mathtt{Heffers}@\mathsf{canActivate}(e_1, \mathtt{Discount}()) \leftarrow \\ e_1@e_2.\mathsf{canActivate}(e_1, \mathtt{Student}(subj)), \\ subj \in \{\mathtt{Physics}, \mathtt{Maths}, \mathtt{CompSci}\}, \\ (e_2 = \mathtt{UCam} \vee e_2 = \mathtt{APU})$$

Alice's Cassandra service will return a copy of the requested credential to Heffers because Alice does indeed possess a suitable credential (first rule) and she has a rule stating that she is willing to reveal her student credential to any requester (second rule, with empty body and no constraint):

$$\mathtt{Alice}@\mathtt{UCam}.\mathsf{canActivate}(\mathtt{Alice}, \mathtt{Student}(subj)) \leftarrow \\ subj \in \{\mathtt{Maths}, \mathtt{Psychology}\}$$

$$\mathtt{Alice}@\mathsf{canReqCred}(e, \mathtt{UCam}.\mathsf{canActivate}(\mathtt{Student}(subj)))$$

2.3 Policy enforcement overview

Entities interact with each other by sending requests through the interface depicted in Figure 1. To describe the Cassandra system, it is not sufficient to describe the policy language; it is also necessary to define the access control engine and how it interacts with the policy.

Recall that the interface defines four types of requests: performing an action, activating a role, deactivating a role and requesting a credential. Upon a request, the access control engine queries the policy to check whether it should be granted.

The exact query depends on the type of request. If the request is to

- **perform an action**, the query will be a corresponding permits predicate;
- **activate a role**, the query will first be a hasActivated predicate to see whether the role has already been activated for the requester, and if this is not the case, there will be a second query, a corresponding canActivate predicate;
- **deactivating somebody's role**, the query will first be a hasActivated predicate to see whether the "victim" has activated that role at all, and then a corresponding canDeactivate predicate. If this succeeds the access control engine will send a isDeactivated query to find the set of all role deactivations triggered by the initial one;
- **request a credential**, the query will first be a corresponding canReqCred predicate, and then secondly, since a credential is nothing but a predicate, the credential itself.

All granted requests (apart from performing an action) have side-effects on the global state: activating a role adds a hasActivated fact (i.e., a rule without body predicates) to the service's policy; deactivating a role removes one or more hasActivated facts; and requesting a credential adds a fact representing the credential to the requester's policy.

For example, reconsider the three policy rules from example 1. Suppose the policy also contains a fourth rule

$$\mathsf{hasActivated}(\texttt{Alice}, \texttt{User}()) \leftarrow$$

Now suppose Alice wants to activate the `Admin`() role. The access control engine will first query the policy with the predicate $\mathsf{hasActivated}(\texttt{Alice}, \texttt{Admin}())$ to check whether Alice has already activated that role. This fails, as expected, so the second query is $\mathsf{canActivate}(\texttt{Alice}, \texttt{Admin}())$. This succeeds because of the first and the fourth rule, and a fifth rule is added to the policy:

$$\mathsf{hasActivated}(\texttt{Alice}, \texttt{Admin}()) \leftarrow$$

Now suppose Alice requests that her `User`() role be activated. The access control engine tries to deduce the predicate $\mathsf{hasActivated}(\texttt{Alice}, \texttt{User}())$; and indeed, Alice is currently active in the user role because of the fifth rule. Then, the policy is queried with $\mathsf{canDeactivate}(\texttt{Alice}, \texttt{Alice}, \texttt{User}())$ which also succeeds because of the second rule. Finally, the access control engine will attempt to find all values satisfying the predicate $\mathsf{isDeactivated}(e, r)$ under the assumption $\mathsf{isDeactivated}(\texttt{Alice}, \texttt{User}())$. By assumption, this will of course be the pair (`Alice`,`User`()) itself, and by the third rule, also the pair (`Alice`,`Admin`()). The found values match the hasActivated facts from the fourth and the fifth rules. Consequently, these two rules are removed from the policy.

3 EHR architecture

The *Spine* is the central service that holds the EHRs of all NHS patients. Including deceased users and users who have moved abroad, the total number of records is expected to be in the order of 10^8. It provides online read and write access to the records to authorised users; these will mainly be clinicians and personal users (patients and their agents).

The Spine is supported by the national *Patient Demographic Service* (PDS). It serves as a single, comprehensive and consistent source of up-to-date demographic patient data (e.g. NHS number, name, address, preferred language). This data is accessed by the Spine and other applications for identifying and authenticating personal users.

The national services are large and have to be able to cope with high loads. By 2010, when the ICRS project is expected to be completed, there will be an estimated number of 50 million patients, 300 million GP appointments, 70 million inpatient episodes and out-patient hospital attendances, and about 30 million other health episodes and encounters each year (§740, [Nat03]).

An integral goal of the NHS National Programme for IT in the NHS (NPfIT) is the deployment of an infrastructure for identification, registration and authentication of users in a secure, standardised and seamless manner across all national and local applications, based on digital credentials and public key technology. Professional users, i.e. clinical and administrative staff, access EHR data in the Spine based on role credentials, issued by NHS-approved *Registration Authorities* (RAs). RAs are also responsible for managing clinical workgroup membership. The size of an RA can vary considerably. Most RAs will be local to a single health organisation, but some may be "more nationally based" (§730.24.0, [Nat03]). It is conceivable that large RAs could be located on the NHS cluster level of which there are five in England (covering London; North East, Yorkshire and Humberside; South East and South West; East of England and East Midlands; West Midlands and North West). A typical cluster comprises of up to 2000 General Practices and 100 Acute Trusts and other health organisations. An RA policy should therefore be able to cope with up to 200,000 registered health professionals.

Local applications are expected to make use of and interoperate with the national services. In particular, local health organisations (e.g. hospitals, doctors' practices) will gradually move from the traditional paper-based records to electronic databases. The records kept on this level are called *Electronic Patient Records* (EPR); summaries of these are used to

populate the Spine. This process may take a long time, and the local procedures differ substantially, so the EHR service cannot be deployed on this level by connecting up all existing EPR systems. Health organisations can be as small as single GP practices but could also be entire NHS trusts with up to 500,000 registered patients.

4 Scenario

The following scenario illustrates some of the more challenging requirements of security policies for our EHR architecture. §5 discusses how these requirements are met by our Cassandra policy (we use the role and action names from the policy in the text below).

Anson Arkwright goes to see Dr Zoe Zimmer, his family's General Practitioner (GP), for an HIV test. Dr Zimmer records the visit in a local EPR item[1] by performing an `Add-record-item` action (with suitable parameters) but does not submit a summary to the Spine on Anson's request. Some time later, Bob Arkwright, Anson's father, visits Dr Zimmer because of heart problems. During the visit he also tells her that he believes his son Anson may be a hypochondriac. Dr Zimmer adds a record item to Bob's EPR about his heart condition and an item in Anson's EPR about his father's comments. The latter item is marked as containing third party information about his father, so as long as his father (or the Caldicott Guardian on his behalf) does not enter a `Third-party-consent` role for that item, Anson will not be able to read it. (Note that we use roles not just to model job positions within an organisation but also to indicate state changes, e.g. giving third-party consent.)

Dr Zimmer also attempts to submit a summary of Bob's new EPR item to his shared EHR: She first activates her `Spine-clinician` role on the Spine by submitting an RA-issued `NHS-clinician-cert` role credential along with the activation request. Subsequently, her `Add-spine-record-item` action succeeds because the Spine can deduce she is Bob's `Treating-clinician` (Bob has explicitly consented to treatment years ago and has not withdrawn his `Consent-to-treatment` role). Dr Zimmer also decides to refer Bob to a local hospital's cardiologist, Dr Hannah Hassan. As Bob's treating clinician, Dr Zimmer can enter a `Referrer` role on the Spine, thus enabling Dr Hassan to also become a treating clinician with a *legitimate relationship*. Bob's consent is not needed, but he has the power to cancel the referral by deactivating Dr Zimmer's `Referrer` role.

At the hospital, a `Receptionist` registers Bob as a patient by activating a `Register-patient` role. After his out-patient visit with Dr Hassan, the receptionist registers him with a surgical team in the same hospital for a heart bypass operation. For this purpose, the receptionist activates appropriate `Register-team-episode` and `Register-ward-episode` roles on the hospital's service, thereby establishing a legitimate treating clinician relationship between Bob and the surgical team and the ward nurses. During surgery, abnormal liver values are found, so the team attempts to search for potentially important information in Bob's EHR on the Spine. However, years ago, Bob activated a `Conceal-request` role on the Spine to conceal the contents of all items in his EHR concerning an alcohol-related liver problem from everybody except clinicians treating him as GP, and this request had been granted by Dr Zimmer, who activated a corresponding `Concealed-by-spine-patient` role for this purpose. The head of the surgical team, Dr Lily Littlewood, decides to "break the seal" to view Bob's restricted EHR item by performing the action `Force-read-spine-item`. This can be done by any clinician with a legitimate relationship but will be marked in the audit trail to be investigated by the hospital's Caldicott Guardian.

Unfortunately, the team encounters further complications during the operation and Bob needs to be kept in an artificial coma. Dr Zimmer agrees to appoint Bob's wife, Carol, to be his agent by activating a `Register-agent` role on the Spine. Carol then requests to activate the `Agent` role on the hospital's service. This succeeds after a cycle of trust negotiation between the hospital, the Spine, and the hospital's RA: the hospital's service reacts to Carol's request by requesting an agent registration credential from the Spine; the

[1] Recall that *EPRs* are the detailed patient records held in health organisations and are meant to replace the traditional paper-based records, whereas *EHRs* are the shared summary records held on the Spine.

Spine replies by requesting a health organisation credential; the hospital agrees by sending an health organisation credential, issued by some RA, to the Spine; the Spine requests an NHS-signed credential from the RA to check if it is an officially approved RA, and finally, the Spine sends the originally requested `Register-agent` credential to the hospital certifying that Carol is indeed Bob's agent.

When Bob is woken and released, he attempts to revoke the agent registration for his wife but fails because it was Dr Zimmer who registered Carol. However, on Bob's request, Dr Zimmer deactivates her `Register-agent` role for Carol. If Carol is active with an `Agent` role on the Spine at that moment, cascading revocation causes that role to be deactivated immediately as well.

5 EHR policy

5.1 Overview

We have drafted a complete Cassandra policy for the NHS Spine and related services, based mainly on the Output Based ICRS Specification Version 2.0 (OBS) [Nat03], reports from NHS pilot projects of the Electronic Records Development and Implementation Programme (ERDIP) [Nat02, Gau03], and various Department of Health documents [Dep01b, Dep02].

The OBS is a 900-page document given to potential suppliers during the procurement process in August 2003. According to the OBS, the ICRS modules are to be delivered by contractors in three phases. By December 2004 (Phase 1), a basic system for accessing EHRs on the Spine should have been delivered, and the Spine populated with patient data. At this stage, the required confidentiality requirements are dangerously low and have prompted harsh criticism from doctors, patient interest groups and the media: A one-off general consent given by a patient would make his personal data available to all clinical users of the Spine. Stricter confidentiality measures are introduced with the later phases. In Phase 2 (December 2006), patients will directly access their health data, they can request to conceal parts of their records, and can identify people (agents) to act on their behalf concerning access to records. Furthermore, access control based on clinical workgroups will be supported. In Phase 3 (December 2010), clinicians will also be able to conceal parts of their patients' records, access will be based on legitimate relationships between patients and clinicians, and systems must separately deal with data containing confidential third-party information. Our proposed policy covers all requirements regarding the access of patient-identifiable data up to and including Phase 3.

The most relevant section for our case study is section §730 on information governance, a list of security and confidentiality requirements for ICRS systems handling patient data. It was specifically written to comply with relevant legislation and guidelines, in particular with the Data Protection Act 1998 and the Caldicott guidelines, and, as far as patient registration is concerned, with the "Registration and Authentication e-Government Strategy Framework Policy and Guidelines". The document acknowledges that the requirements of this section are likely to evolve due to changes in national guidelines and legal requirements, but also to new positions emerging from still ongoing NHS consultation processes. Some of the points will be affected by design work yet to be done, and some are "subject to further guidance".

Consequently, the requirements are sketchy in places. For example, section §730.9 states the requirements of role-based access control without specifying which roles will be used and their associated privileges. Similarly, it is a Phase 3 requirement that only clinicians with a legitimate relationship have access to patient data (§730.17). However, the section does not explain the rules for establishing legitimate relationships. In writing a formal policy for the system, many such missing details had to be filled in. Many of our rules are based simply on common sense, even though they are not explicitly required in the OBS: for example, that a legitimate relationship is automatically revoked if the respective patient is no longer registered in the system.

In some cases we had to choose from a number of conceivable options and, in general, favoured the more demanding solutions. For example, patients may seal off groups of related clinical data (e.g. all the data about a particular event) (§730.48.2), but the OBS defers the

specification of the granularity of such groupings until further guidance has been produced. Our solution gives patients an extremely high flexibility in specifying sealing-off criteria and may well be more flexible than what is actually needed, but it shows that less demanding solutions could also be implemented in Cassandra.

In our case study, the *entities* are the individual users (patients, clinicians, staff) as well as the national and local services. We have written policies for the Spine, the PDS, Addenbrooke's Hospital (an exemplary hospital), and Addenbrooke's RA. The policies use the constraint domain $\mathcal{C}_0$ [BS04b] and comprise 375 rules, 71 roles and 12 actions. Of the 375 rules, there are 118 canActivate, 97 canDeactivate, 51 isDeactivated, 29 permits and 27 canReqCred rules. The remaining 53 are user-defined rules. The case study suggests that common policy idioms such as appointment hardly occur in their pure forms in practice. It is therefore not sufficient to equip a policy language with standard policy constructs (e.g. appointment in OASIS [HBM98, YMB02, BMY02]); rather, it is necessary to be able to express different variants of them. The rules can be roughly divided into the following categories:

5.1.1 Permissions assignment

Many of the permits rules are straightforward parameterised role-action assignments, e.g. "patients can annotate their own record items". Others require more than one role-related prerequisite condition, e.g. "clinicians can force-read record items concealed by a patient if they have activated their clinician role and if they are member of a workgroup (clinical team or ward) currently treating the patient". The last condition is also an example of an auxiliary or derived role: the `Group-treating-clinician` role need not be activated when using the rule; it is sufficient that it *can* be activated.

The permits rules concerning reading record items are typically also conditioned on patient and third-party consent and (absence of) access restrictions. All these conditions correspond to role activations of users other than the requester. Such rules cannot be easily expressed in languages in which the subject parameters of the head and the body are implicitly the same, e.g. SPKI/SDSI [Ell99, EFL+99], RT [LMW02] or OASIS [HBM98, YMB02, BMY02].

5.1.2 Consent

Access to health records is primarily based on explicit patient consent. Consent may be required for initial treatment, for referrals and for disclosure of third-party information. We implement consent as a form of appointment: by activating a consent role, a patient "appoints" a clinician to be e.g. a `Treating-clinician` with a legitimate relationship. To prevent frivolous users from unsolicitedly activating myriads of consent roles, the user must first have been requested to activate the consent role. These consent requests are again implemented as a form of appointment, but now the other way round: by activating a consent request role, the clinician enables the patient to activate a consent role. Consent is thus implemented as a two-stage appointment mechanism.

5.1.3 Registration

Registration is an administrative task that takes on many forms in our case study: for example, PDS managers enter newly born patients into the PDS, receptionists register new patients, human resource managers employ clinicians and other staff, head nurses assign nurses to wards, and heads of clinical teams assign clinicians to their respective teams. Registration can again be implemented using variants of the appointment encoding given in [BS04b]. Variants include combinations with cardinality restrictions ("patients can register at most three distinct agents acting on their behalf") and uniqueness constraints ("a patient can only be registered if no one has already activated the registration role for that patient"). The two mentioned variants make use of Cassandra's aggregation operators.

5.1.4 Referral

Referral is implemented as a form of delegation. Our case study exhibits two kinds of patient referral. On the Spine, no patient consent is required, and referral chains are of unbounded length. On the level of the local health organisation, we decided to implement a stricter alternative: a local treating clinician can refer the patient to an external clinician (who will then have restricted rights to read the local EPR record items) only with explicit patient consent, and the delegation chain can only be of length one.

5.1.5 Sealing off data

This is a policy idiom motivated by the requirement that patients may specify access restrictions on their data. Patients can fine-tune the access rights to their records by activating an appropriate concealment request role, if the request is subsequently approved by a clinician. The permits rules governing read access need to check that no such concealment role has been activated and approved; this requires universally quantified negation, expressed with the help of aggregation operators.

5.1.6 Deactivation

canDeactivate rules specify who can deactivate which roles. Although these rules are rather straightforward, it is important that deactivation can be specified flexibly. For example, revocation of agent registrations is asymmetric in the sense that patients can only revoke the agents they have appointed themselves (grant-dependent revocation), whereas Caldicott Guardians can revoke not only the agents they have appointed for a patient but also those appointed by the patient (variant of grant-independent revocation). Furthermore, agent role activations are revoked only if all their registrations have been revoked. This is yet another example of universally quantified negation requiring aggregation operators.

Cascading deactivation, specified by isDeactivated rules, is used to automatically deactivate a role if some other role is deactivated. For example, the revocation of a patient's registration in the hospital triggers the deactivation of all roles that have something to do with that patient, including agent registrations, inpatient episode registrations, legitimate relationships with clinicians, access restriction roles, and consent roles.

5.1.7 Credential management

Credential-based trust negotiation and credential protection are governed by the interaction between canReqCred rules and rules with remote body predicates. The scenario in §4 gives an example of multi-phase automatic trust negotiation. canReqCred rules are also used for regulating direct credential requests from entities. For example, agent credentials from the Spine can be requested by certified health organisations, and also by the agent himself. The location parameter of Cassandra predicates facilitates very flexible forms of automated credential retrieval: unlike other policy systems, credential locations are not restricted to the issuer or the credential subject. For example, a credential of the form

$$\texttt{RA}.\mathsf{hasActivated}(\texttt{RA-adm}, \texttt{NHS-health-org-cert}(\texttt{Org}, \texttt{Start}, \texttt{End}))$$

may be found at the location `Org` which is neither the issuer (`RA`) nor the subject (`RA-adm`).

In the following we describe in detail the policies for the EHR architecture outlined in §3 and illustrated in the scenario (§4). All rules are also listed in the Appendix A. The section numbers starting with 730 relate to the OBS [Nat03].

5.2 The Spine

The Spine is the heart of the ICRS, containing EHRs for all patients and providing online access to the records to both to both patients and professional users. Our policy for the

Spine defines the access roles and privileges, and manages consent, legitimate relationships, and concealment of record items. The Spine policy comprises 137 rules.

5.2.1 EHR structure

Each patient is associated with exactly one EHR consisting of a set of record items, indexed by an ID. We thus assume that each record item is uniquely identified by a pair (*pat*, *id*): the item with ID *id* in the EHR of patient *pat*. The policy accesses relevant fields of a record item via the following system functions, each of which take such a pair as argument:

- `Get-spine-record-author` returns the author of the item. This is always a clinician.
- `Get-spine-record-org` returns the health organisation of the author.
- `Get-spine-record-time` returns the time and date of the item's creation.
- `Get-spine-record-subjects` returns a set of subject matters (chosen from a predefined list of valid subjects such as 'allergy', 'abortion', 'cancer') the item relates to.
- `Get-spine-record-third-parties` returns a set of all third parties whose consent must be sought prior to revealing the item to the patient.

As is the case throughout this case study, fields and parameters that are irrelevant to the policy are omitted. For example, there is no 'content' field for record items, as the policy does not perform any computations on the actual content of the item. Similarly, alert-raising actions such as `Force-read-spine-record-item`, i.e. the forced reading of an item against the patient's wishes, would in reality contain a 'reason' parameter explaining why this action is performed, but it is omitted since it is irrelevant for the policy.

5.2.2 Main roles

The Spine policy is role-based (§730.9), and designed in such a way that users need to have activated a single main role, `Spine-clinician`, `Spine-admin`, `Patient`, `Agent` and `Third-party`, before performing any action or activating a registration, consent or concealment roles. The requirement that *exactly one* role must be active (§730.12.10) is an example of dynamic n-wise separation of duties [BS04b]. The activation rules for all these roles contain a user-defined predicate no-main-role-active(*user*) (S1.5.3) that is satisfied only if the user has not already activated any of these roles:

(S1.5.3)
no-main-role-active(*user*) ←
    count-agent-activations(n, *user*),
    count-spine-clinician-activations(n, *user*),
    count-spine-admin-activations(n, *user*),
    count-patient-activations(n, *user*),
    count-third-party-activations(n, *user*),
    $n = 0$

As in [BS04b], this is achieved by the use of aggregation predicates (S1.1.4, S1.2.4, S1.3.4, S1.4.5, S2.2.13).

(S1.1.4)
count-spine-clinician-activations(count⟨u⟩, *user*) ←
    hasActivated(*user*, `Spine-clinician`(*ra*, *org*, *spcty*))

(S1.2.4)
count-spine-admin-activations(count⟨u⟩, *user*) ←
    hasActivated(*user*, `Spine-admin`())

(S1.3.4)
count-patient-activations(count⟨u⟩, *user*) ←
    hasActivated(*user*, `Patient`())

(S1.4.5)
count-agent-activations(count⟨u⟩, $user$) ←
    hasActivated($user$, Agent(pat))

(S2.2.13)
count-third-party-activations(count⟨u⟩, $user$) ←
    hasActivated($user$, Third-party())

The following describes the rules concerning the main access roles apart from Third-party which is discussed in §5.2.3.

Clinicians A clinician certified by an NHS-approved RA ra working for health organisation org with specialty $spcty$ can activate the role Spine-clinician(ra, org, $spcty$). The credential can be either submitted locally (S1.1.1) or retrieved from the RA (S1.1.2):

(S1.1.1)
canActivate(cli, Spine-clinician(ra, org, $spcty$)) ←
    ra.hasActivated(x, NHS-clinician-cert(org, cli, $spcty$, $start$, end)),
    canActivate(ra, Registration-authority()),
    no-main-role-active(cli),
    Current-time() ∈ [$start$, end]

(S1.1.2)
canActivate(cli, Spine-clinician(ra, org, $spcty$)) ←
    ra@ra.hasActivated(x, NHS-clinician-cert(org, cli, $spcty$, $start$, end)),
    canActivate(ra, Registration-authority()),
    no-main-role-active(cli),
    Current-time() ∈ [$start$, end]

In both cases, the credential must be still valid, and it is checked if ra is an NHS-approved RA (S1.5.1), if necessary, by contacting the RA itself (S1.5.2):

(S1.5.1)
canActivate(ra, Registration-authority()) ←
    NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

(S1.5.2)
canActivate(ra, Registration-authority()) ←
    ra@NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

A clinician can deactivate her own role (S1.1.3):

(S1.1.3)
canDeactivate(cli, cli, Spine-clinician(ra, org, $spcty$)) ←

Administrators Administrators are responsible for registering new patients and unregistering deceased patients or those who have moved away. The administrator role Spine-admin() can be activated if the person has been registered as such (S1.2.1):

(S1.2.1)
canActivate(adm, Spine-admin()) ←
    hasActivated(x, Register-spine-admin(adm)),
    no-main-role-active(adm)

Administrators can be registered if they have not already been registered (S1.2.5, S1.2.7) and unregistered (S1.2.6) by other administrators:

(S1.2.5)
canActivate(adm, Register-spine-admin($adm2$)) ←
    hasActivated(adm, Spine-admin()),
    spine-admin-regs(0, $adm2$)

(S1.2.7)
spine-admin-regs(count$\langle x \rangle$, adm) $\leftarrow$
    hasActivated(x, Register-spine-admin(adm))

(S1.2.6)
canDeactivate(adm, x, Register-spine-admin($adm2$)) $\leftarrow$
    hasActivated(adm, Spine-admin())

The role can be deactivated by the administrator herself (S1.2.2) and is automatically deactivated when the registration is cancelled (S1.2.3):

(S1.2.2)
canDeactivate(adm, adm, Spine-admin()) $\leftarrow$

(S1.2.3)
isDeactivated(adm, Spine-admin()) $\leftarrow$
    isDeactivated(x, Register-spine-admin(adm))

Patients A patient can activate (S1.3.1) the role Patient() if he has been registered on the Spine and also at the PDS. The latter condition is checked by contacting the PDS directly, as required by §730.24.0b:

(S1.3.1)
canActivate(pat, Patient()) $\leftarrow$
    hasActivated(x, Register-patient(pat)),
    no-main-role-active(pat),
    PDS@PDS.hasActivated(y, Register-patient(pat))

Patients can be registered if they have not already been registered (S1.3.5, S1.3.7) and unregistered (S1.3.6) by administrators:

(S1.3.5)
canActivate(adm, Register-patient(pat)) $\leftarrow$
    hasActivated(adm, Spine-admin()),
    patient-regs(0, pat)

(S1.3.7)
patient-regs(count$\langle x \rangle$, pat) $\leftarrow$
    hasActivated(x, Register-patient(pat))

In our policy, the removal of a patient's registration should only be performed if his data is permanently removed from the Spine, for example, in the case of the patient's death, after the legal minimal retention period.

The patient role can be deactivated by the patient himself (S1.3.2) and is automatically deactivated when the registration is cancelled (S1.3.3):

(S1.3.2)
canDeactivate(pat, pat, Patient()) $\leftarrow$

(S1.3.3)
isDeactivated(pat, Patient()) $\leftarrow$
    isDeactivated(x, Register-patient(pat))

Agents A patient can identify agents (for example carers, family members, guardians of a child) who have a special kind of legitimate relationship that allow them to act on the patient's behalf and to access his record (§730.20.10, §730.52). A person can activate (S1.4.1) the role Agent(pat) for a patient pat if he has been appointed as an agent and if

he is registered at the PDS. As in the case of patient role activation, the latter condition is checked by contacting the PDS directly (P2.2.5):

(S1.4.1)
$\mathsf{canActivate}(ag, \mathtt{Agent}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \mathtt{Register\text{-}agent}(ag, pat)),$
$\quad \mathtt{PDS@PDS}.\mathsf{hasActivated}(y, \mathtt{Register\text{-}patient}(ag)),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(ag)$

(S2.2.5)
$\mathsf{canDeactivate}(ag, y, \mathtt{Request\text{-}third\text{-}party\text{-}consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Agent}(pat))$

The role can be deactivated by the agent himself (S1.4.2) and is deactivated automatically if all his agent registrations for the patient have been cancelled (S1.4.3, S1.4.4):

(S1.4.2)
$\mathsf{canDeactivate}(ag, ag, \mathtt{Agent}(pat)) \leftarrow$

(S1.4.3)
$\mathsf{isDeactivated}(ag, \mathtt{Agent}(pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \mathtt{Register\text{-}agent}(ag, pat)),$
$\quad \mathsf{other\text{-}agent\text{-}regs}(0, x, ag, pat)$

(S1.4.4)
$\mathsf{other\text{-}agent\text{-}regs}(\mathsf{count}\langle y\rangle, x, ag, pat) \leftarrow$
$\quad \mathsf{hasActivated}(y, \mathtt{Register\text{-}agent}(ag, pat)),$
$\quad x \neq y$

Patients can appoint up to three personal agents (S1.4.9, S1.4.14). The rationale behind this cardinality restriction is to prevent users from frivolously or maliciously clogging up the policy.

(S1.4.9)
$\mathsf{canActivate}(pat, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Patient}()),$
$\quad \mathsf{agent\text{-}regs}(n, pat),$
$\quad n < 3$

(S1.4.14)
$\mathsf{agent\text{-}regs}(\mathsf{count}\langle x\rangle, pat) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Register\text{-}agent}(x, pat))$

Agents can also be appointed by the patient's GP (S1.4.10). This can be done without the patient's consent if he lacks competence (§730.55.6), for example if the patient is a child and not Gillick-competent[2] but objects to his parents acting on his behalf:

(S1.4.10)
$\mathsf{canActivate}(cli, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \mathtt{Spine\text{-}clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \mathtt{General\text{-}practitioner}(pat))$

The patient can only cancel the agent appointments he has made himself but not those made by his GP (S1.4.11) (grant-dependent revocation). A patient's GP, on the other hand, can cancel any agent appointments (S1.4.12) (grant-independent revocation):

(S1.4.11)
$\mathsf{canDeactivate}(pat, pat, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Patient}())$

[2]A child is *Gillick-competent* if it is deemed mature enough to give or refuse consent to a medical procedure by him or herself. Parental consent is not legally required, only recommended.

(S1.4.12)
$\mathsf{canDeactivate}(cli, x, \texttt{Register-agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{General-practitioner}(pat))$

All agent appointments are automatically cancelled if the patient's registration is cancelled (S1.4.13):

(S1.4.13)
$\mathsf{isDeactivated}(x, \texttt{Register-agent}(agent, pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat))$

An agent can request a credential certifying his role as agent (S1.4.6):

(S1.4.6)
$\mathsf{canReqCred}(ag, \texttt{Spine.canActivate}(ag, \texttt{Agent}(pat))) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat))$

Such a credential could for example be used for authorisation in the EPR systems of local health organisations, as in the following rules located at Addenbrooke's Hospital's policy (A1.6.2, A1.6.3):

(A1.6.2)
$\mathsf{canActivate}(agent, \texttt{Agent}(pat)) \leftarrow$
$\quad \mathsf{canActivate}(pat, \texttt{Patient}()),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(agent),$
$\quad \texttt{PDS@PDS.hasActivated}(x, \texttt{Register-patient}(agent)),$
$\quad \texttt{Spine@Spine.canActivate}(agent, \texttt{Agent}(pat))$

(A1.6.3)
$\mathsf{isDeactivated}(ag, \texttt{Agent}(pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Register-agent}(ag, pat)),$
$\quad \mathsf{other\text{-}agent\text{-}regs}(0, x, ag, pat)$

Health organisations can also directly ask for such a credential. For this purpose they first have to authenticate themselves with a currently valid RA-signed health organisation credential (S1.4.7). If no such credential is submitted, the Spine tries to retrieve it from the health organisation (S1.4.8). In both cases it is also checked whether the RA is approved by the NHS:

(S1.4.7)
$\mathsf{canReqCred}(org, \texttt{Spine.canActivate}(ag, \texttt{Agent}(pat))) \leftarrow$
$\quad ra.\mathsf{hasActivated}(x, \texttt{NHS-health-org-cert}(org, start, end)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}()),$
$\quad \texttt{Current-time}() \in [start, end]$

(S1.4.8)
$\mathsf{canReqCred}(org, \texttt{Spine.canActivate}(ag, \texttt{Agent}(pat))) \leftarrow$
$\quad org@ra.\mathsf{hasActivated}(x, \texttt{NHS-health-org-cert}(org, start, end)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}()),$
$\quad \texttt{Current-time}() \in [start, end]$

5.2.3 Express consent

It is generally agreed that disclosure of patient data must be preceded by the patient's express consent. There is of course much controversy about the details: How specific should the statement of consent be? How often should consent be sought? When can (implied) consent be derived from express consent? Very strict consent requirement may be good in terms of medical ethics but may be too impractical for users.

The following describes the rules concerning patient consent based on our interpretation of the OBS. This part of the policy is very likely change as the debate on consent evolves and more guidelines are produced.

One-off consent According to §730.4.2, patients will not be allowed to refuse having their medical data stored on the Spine, but the patient's "one-off consent" is required to release the data for clinical care[3].

In the Spine policy, a patient *pat* can give a one-off consent to have his data made available on the Spine by activating the role `One-off-consent`(*pat*) (S2.1.1):

(S2.1.1)
canActivate(*pat*, `One-off-consent`(*pat*)) ←
    hasActivated(*pat*, `Patient`())

Furthermore, his agent can also do this on his behalf (S2.1.2), and so can any treating clinician (i.e. a clinician with a legitimate relationship) (S2.1.3).

(S2.1.2)
canActivate(*ag*, `One-off-consent`(*pat*)) ←
    hasActivated(*ag*, `Agent`(*pat*))

(S2.1.3)
canActivate(*cli*, `One-off-consent`(*pat*)) ←
    hasActivated(*cli*, `Spine-clinician`(*ra*, *org*, *spcty*)),
    canActivate(*cli*, `Treating-clinician`(*pat*, *org*, *spcty*))

The patient (S2.1.4), or his agent (S2.1.5) or a treating clinician (S2.1.6) on his behalf, can withdraw the consent by deactivating the role:

(S2.1.4)
canDeactivate(*pat*, *x*, `One-off-consent`(*pat*)) ←
    hasActivated(*pat*, `Patient`())

(S2.1.5)
canDeactivate(*ag*, *x*, `One-off-consent`(*pat*)) ←
    hasActivated(*ag*, `Agent`(*pat*))

(S2.1.6)
canDeactivate(*cli*, *x*, `One-off-consent`(*pat*)) ←
    hasActivated(*cli*, `Spine-clinician`(*ra*, *org*, *spcty*)),
    canActivate(*cli*, `Treating-clinician`(*pat*, *org*, *spcty*))

The consent role is automatically deactivated if the patient's registration is cancelled (S2.1.7):

(S2.1.7)
isDeactivated(*x*, `One-off-consent`(*pat*)) ←
    isDeactivated(*y*, `Register-patient`(*pat*))

Third-party consent Health record items may sometimes contain information about someone other than the patient. A GP may for example include information about certain diseases of the patient's blood relatives [SMW93]. According to the UK Data Protection Act 1998, patients may not view record items that may reveal confidential information about third parties without the third parties' consent. The OBS only requires that any third-party information be withheld from the patient. Our policy also allows patients to request a third party to give their consent.

Third-party consent from a third party *x* for a particular record item *id* of a patient *pat* can be requested by the patient himself (S2.2.1), by his agent (S2.2.2) or by any clinician currently treating him (S2.2.3) by the activation of `Request-third-party-consent`(*x*, *pat*, *id*).

[3]This is one of the more controversial points as the specifications also allow clinicians to access a patient's records without his consent in "exceptional circumstances" (§730.4.11)

A further condition for this request is that x is actually recorded as a third party in the record item (*pat, id*):

(S2.2.1)
canActivate(pat, Request-third-party-consent(x, pat, id)) ←
    hasActivated(pat, Patient()),
    $x \in$ Get-spine-record-third-parties(pat, id)

(S2.2.2)
canActivate(ag, Request-third-party-consent(x, pat, id)) ←
    hasActivated(ag, Agent(pat)),
    $x \in$ Get-spine-record-third-parties(pat, id)

(S2.2.3)
canActivate(cli, Request-third-party-consent(x, pat, id)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, Treating-clinician(pat, org, $spcty$)),
    $x \in$ Get-spine-record-third-parties(pat, id)

The request can be cancelled by the same users who can activate it (S2.2.4-6). Additionally, the respective third party may also deactivate the request role (S2.2.7), thereby withholding (or later withdrawing) consent to disclosure:

(S2.2.4)
canDeactivate(pat, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(pat, Patient())

(S2.2.5)
canDeactivate(ag, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(pat, Agent(pat))

(S2.2.6)
canDeactivate(cli, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$))

(S2.2.7)
canDeactivate(x, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party())

All requests are automatically deactivated when the patient's registration is cancelled (S2.2.8):

(S2.2.8)
isDeactivated(x, Request-third-party-consent(y, pat, id)) ←
    isDeactivated(z, Register-patient(pat))

In order to give consent to third-party disclosure, the third party first has to activate the Third-party() role (S2.2.10). This is allowed if his consent has been requested and he is a registered user at the PDS:

(S2.2.10)
canActivate(x, Third-party()) ←
    hasActivated(y, Request-third-party-consent(x, pat, id)),
    no-main-role-active(x),
    PDS@PDS.hasActivated(z, Register-patient(x))

The role can be deactivated by the third party (S2.2.11) and is automatically deactivated when all relevant requests have been withdrawn (S2.2.12, S2.2.9):

(S2.2.11)
canDeactivate(x, x, Third-party()) ←

(S2.2.12)
isDeactivated(x, Third-party()) ←
    isDeactivated(y, Request-third-party-consent(x, pat, id)),
    other-third-party-consent-requests(0, y, x)

(S2.2.9)
other-third-party-consent-requests(count⟨x⟩, y, z) ←
    hasActivated(x, Request-third-party-consent(z, pat, id)),
    $x \neq y$

Once a user has activated the Third-party() role, he can grant existing third-party consent requests by activating the corresponding Third-party-consent(x, pat, id) role (S2.2.14):

(S2.2.14)
canActivate(x, Third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party()),
    hasActivated(y, Request-third-party-consent(x, pat, id))

Alternatively, consent can also be given by a clinician currently treating the patient to whom the record item belongs (S2.2.15), as in many cases the third party will not be able to give consent in this way, or the treating clinician can deduce the third party's implied consent:

(S2.2.15)
canActivate(cli, Third-party-consent(x, pat, id)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, Treating-clinician(pat, org, $spcty$)),
    hasActivated(y, Request-third-party-consent(x, pat, id))

Third-party consent is automatically cancelled if all relevant requests have been cancelled (S2.2.16) (hence the consent role never needs to be deactivated directly):

(S2.2.16)
isDeactivated(x, Third-party-consent(x, pat, id)) ←
    isDeactivated(y, Request-third-party-consent(x, pat, id)),
    other-third-party-consent-requests(0, y, x)

Consent to treatment The OBS does not clearly say how legitimate relationships are formally established. Our policy requires the patient to give express consent for a clinician to have the role of a *treating clinician* that represents the legitimate relationship between the clinician and the patient.

As is the case with third-party consent, the consent to treatment of patient *pat* by clinician *cli* must first be requested by a clinician (this may be *cli* herself), in order to prevent non-professional users from activating a large number of unrequested consent roles. The request is represented by the activation of Request-consent-to-treatment(*pat*, *org*, *cli*, *spcty*), where *org* is *cli*'s health organisation and *spcty* her specialty (S2.3.1):

(S2.3.1)
canActivate($cli1$, Request-consent-to-treatment(pat, $org2$, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, Spine-clinician($ra1$, $org1$, $spcty1$)),
    canActivate($cli2$, Spine-clinician($ra2$, $org2$, $spcty2$)),
    canActivate(pat, Patient())

The request can be cancelled by the requester herself (S2.3.2), by the clinician *cli* (S2.3.3), by the patient (S2.3.4), or his agent (S2.3.5) or his GP (S2.3.6) on the patient's behalf:

(S2.3.2)
canDeactivate($cli1$, $cli1$,
        Request-consent-to-treatment(pat, $org2$, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, Spine-clinician($ra1$, $org1$, $spcty1$))

(S2.3.3)
canDeactivate($cli2$, $cli1$,
Request-consent-to-treatment(pat, $org2$, $cli2$, $spcty2$)) ←
hasActivated($cli2$, Spine-clinician($ra2$, $org2$, $spcty2$))

(S2.3.4)
canDeactivate(pat, x, Request-consent-to-treatment(pat, org, cli, $spcty$)) ←
hasActivated(pat, Patient())

(S2.3.5)
canDeactivate(ag, x, Request-consent-to-treatment(pat, org, cli, $spcty$)) ←
hasActivated(ag, Agent(pat))

(S2.3.6)
canDeactivate(cli, x, Request-consent-to-treatment(pat, org, $cli2$, $spcty$)) ←
hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
canActivate(cli, General-practitioner(pat))

A request is automatically deactivated when the patient's registration is cancelled (S2.3.7):

(S2.3.7)
isDeactivated(x, Request-consent-to-treatment(pat, org, cli, $spcty$)) ←
isDeactivated(y, Register-patient(pat))

An activated request can be granted by the patient directly (S2.3.9), or by his agent (S2.3.10) or any treating clinician (S2.3.11) on his behalf, by activating Consent-to-treatment(pat, org, cli, $spcty$):

(S2.3.9)
canActivate(pat, Consent-to-treatment(pat, org, cli, $spcty$)) ←
hasActivated(pat, Patient()),
hasActivated(x, Request-consent-to-treatment(pat, org, cli, $spcty$))

(S2.3.10)
canActivate(ag, Consent-to-treatment(pat, org, cli, $spcty$)) ←
hasActivated(ag, Agent(pat)),
hasActivated(x, Request-consent-to-treatment(pat, org, cli, $spcty$))

(S2.3.11)
canActivate($cli1$, Consent-to-treatment(pat, org, $cli2$, $spcty$)) ←
hasActivated($cli1$, Spine-clinician(ra, org, $spcty$)),
canActivate($cli1$, Treating-clinician(pat, org, $spcty$)),
hasActivated(x, Request-consent-to-treatment(pat, org, $cli2$, $spcty$))

Consent is automatically cancelled if all relevant requests have been withdrawn (S2.3.12, S2.3.8):

(S2.3.12)
isDeactivated(x, Consent-to-treatment(pat, org, cli, $spcty$)) ←
isDeactivated(y, Request-consent-to-treatment(pat, org, cli, $spcty$)),
other-consent-to-treatment-requests(0, y, pat, org, cli, $spcty$)

(S2.3.8)
other-consent-to-treatment-requests(count⟨y⟩, x, pat, org, cli, $spcty$) ←
hasActivated(y, Request-consent-to-treatment(pat, org, cli, $spcty$)),
$x \neq y$

Often, it is a workgroup or team consisting of several clinicians providing care to the patient, each requiring access to the patient's record (§730.20.2). To support workgroup-based access control, patients can activate the Consent-to-group-treatment(pat, org, $group$) role if the corresponding Request-consent-to-group-treatment role has been activated. The

relevant rules (S2.4.1–12) are much the same as those for standard consent to treatment except that the request can also be deactivated by workgroup members (S2.4.6). Workgroup membership is checked by requesting an RA-issued membership credential from the RA:

(S2.4.6)
canDeactivate(cli, x, `Request-consent-to-group-treatment`(pat, org, $group$)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    ra@ra.canActivate(cli, `Workgroup-member`(org, $group$, $spcty$))

5.2.4 Legitimate Relationship

A clinician has a legitimate relationship with a patient if she is currently involved in providing care to the patient. In Phase 3, only clinicians with legitimate relationships have access to a patient's data (§730.17). We have seen above how such a relationship is formed by the clinician requesting the patient's consent and the patient giving consent to treatment. Here we discuss some more ways for establishing legitimate relationships, and the rules concerning the auxiliary roles `Treating-clinician` and `Group-treating-clinician`.

Referrals Clinicians can "delegate" their legitimate relationship to a patient to another clinician by the act of referral (§730.20.8). No express patient consent is needed in this case.

A clinician currently treating a patient *pat* can refer the patient to another clinician *cli* from *org* in specialty *spcty* by activating `Referrer`(*pat*, *org*, *cli*, *spcty*) (S3.1.1):

(S3.1.1)
canActivate($cli1$, `Referrer`(pat, org, $cli2$, $spcty1$)) ←
    hasActivated($cli1$, `Spine-clinician`(ra, org, $spcty2$)),
    canActivate($cli1$, `Treating-clinician`(pat, org, $spcty2$))

Both the referring clinician (S3.1.2) and the patient (S3.1.3) can cancel the referral:

(S3.1.2)
canDeactivate($cli1$, $cli1$, `Referrer`(pat, org, $cli2$, $spcty1$)) ←

(S3.1.3)
canDeactivate(pat, $cli1$, `Referrer`(pat, org, $cli2$, $spcty1$)) ←

The referral role is also deactivated if the patient's registration is cancelled (S3.1.4):

(S3.1.4)
isDeactivated($cli1$, `Referrer`(pat, org, $cli2$, $spcty1$)) ←
    isDeactivated(x, `Register-patient`(pat))

Accident and emergency In the case of accident or emergency, it may be necessary to get access to a patient's records without his explicit consent. A clinician can activate the `Spine-emergency-clinician`(*org*, *pat*) role for any registered patient *pat* (S3.2.1):

(S3.2.1)
canActivate(cli, `Spine-emergency-clinician`(org, pat)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    canActivate(pat, `Patient`())

This should trigger an alert and be highlighted in the audit trail (notifications and audit are not modelled by our policies). The `Spine-emergency-clinician` role can be deactivated by the clinician herself (S3.2.2):

(S3.2.2)
canDeactivate(cli, cli, `Spine-emergency-clinician`(org, pat)) ←

The role is automatically deactivated if the user's clinician role is deactivated (S3.2.3) and also if the patient's registration is cancelled (S3.2.4):

(S3.2.3)
$\mathsf{isDeactivated}(x, \texttt{Spine-emergency-clinician}(org, pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Spine-clinician}(ra, org, spcty))$

(S3.2.4)
$\mathsf{isDeactivated}(x, \texttt{Spine-emergency-clinician}(org, pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat))$

Treating clinicians A legitimate relationship between a clinician or a workgroup and a patient is manifested in the roles `Treating-clinician` or `Group-treating-clinician`, respectively. These are auxiliary roles that never need to be actually activated as it is only ever checked whether a user *can* activate them, but not whether they *have* been activated.

A clinician *cli* is associated with the role `Treating-clinician`(*pat*,*org*,*spcty*) if express consent to treatment has been given, i.e. the matching role `Consent-to-treatment`(*pat*, *org*, *cli*, *spcty*) has been activated (S3.3.1):

(S3.3.1)
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Consent-to-treatment}(pat, org, cli, spcty))$

Alternatively, no consent is required if the clinician is active as `Spine-emergency-clinician`(*org*, *pat*), but in this case, *spcty* must be set to '`A-and-E`' (S3.3.2):

(S3.3.2)
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-emergency-clinician}(org, pat)),$
$\quad spcty = \texttt{A-and-E}$

Treating clinicians are also those with a matching referral (S3.3.3):

(S3.3.3)
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)) \leftarrow$
$\quad \mathsf{canActivate}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Referrer}(pat, org, cli, spcty))$

The GP role `General-practitioner`(pat) is derived from the `Treating-clinician` role and represents a clinician who treats that patient in the specialty '`GP`' (S3.3.5):

(S3.3.5)
$\mathsf{canActivate}(cli, \texttt{General-practitioner}(pat)) \leftarrow$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)),$
$\quad spcty = \texttt{GP}$

Similarly, the `Group-treating-clinician` role can be activated by members of the workgroup, if consent to group treatment has been given. An RA-issued credential is required for proving workgroup membership, either to be submitted directly (S3.4.1) or automatically fetched from the clinician's RA (S3.4.2). The rules also check whether the RA is approved by the NHS:

(S3.4.1)
$\mathsf{canActivate}(cli, \texttt{Group-treating-clinician}(pat, ra, org, group, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Consent-to-group-treatment}(pat, org, group)),$
$\quad ra.\mathsf{canActivate}(cli, \texttt{Workgroup-member}(org, group, spcty)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}())$

(S3.4.2)
$\mathsf{canActivate}(cli, \texttt{Group-treating-clinician}(pat, ra, org, group, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Consent-to-group-treatment}(pat, org, group)),$
$\quad ra@ra.\mathsf{canActivate}(cli, \texttt{Workgroup-member}(org, group, spcty)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}())$

Any `Group-treating-clinician` can also be a `Treating-clinician` (S3.3.4); this is a simple example of role hierarchy:

(S3.3.4)
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)) \leftarrow$
$\quad \mathsf{canActivate}(cli, \texttt{Group-treating-clinician}(pat, ra, org, group, spcty))$

5.2.5 Sealing off data

In the past, patients were often refused access to their own records. The recent decades have brought much more openness between doctors and patients [SMW93]. The change is also reflected in law: The Medical Reports Act 1988 and the Access to Health Records Act 1990 give patients the right to access records created after November 1991. There still are, however, exceptional situations in which doctors may withhold parts of the record. The legislation states two legitimate reasons: firstly, if the information relates to third parties, and secondly, when the information must be regarded as harmful to the patient.

Clinicians restricting access The Spine will have provisions for clinicians to seal off data from the patient in exceptional circumstances (§730.49). We have already showed how our policy deals with record items relating to third parties. Treating clinicians can seal off a set of items *ids* from a patient *pat*'s EHR by activating the role `Concealed-by-spine-clinician`(*pat*, *ids*, *start*, *end*) (S4.1.1):

(S4.1.1)
$\mathsf{canActivate}(cli, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

The access restriction is valid only within the time interval [*start*, *end*] (cf. §730.51.8). Any such access restriction can be lifted by the clinician who imposed it (S4.1.2) by the patient's GP (S4.1.3), and by any clinician working in the same team (S4.1.4) (cf. §730.51.10):

(S4.1.2)
$\mathsf{canDeactivate}(cli, cli, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$

(S4.1.3)
$\mathsf{canDeactivate}(cli, cli2, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{General-practitioner}(pat))$

(S4.1.4)
$\mathsf{canDeactivate}(cli1, cli2, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli1, \texttt{Spine-clinician}(ra, org, spcty1)),$
$\quad \mathsf{canActivate}(cli1, \texttt{Group-treating-clinician}(pat, ra, org, group, spcty1)),$
$\quad \mathsf{canActivate}(cli2, \texttt{Group-treating-clinician}(pat, ra, org, group, spcty2)),$
$\quad \mathsf{hasActivated}(x, \texttt{Consent-to-group-treatment}(pat, org, group))$

The role is automatically deactivated as soon as the patient's registration is cancelled (S4.1.5):

(S4.1.5)
$\mathsf{isDeactivated}(x, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat))$

Patients restricting access The OBS also allows patients to seal off selected clinical parts of their record (§730.45). The specification suggests that the patient should file a sealing-off request which is subsequently dealt with by a clinician (§730.48.2).

Our solution gives patients much flexibility for specifying record items to seal off. A patient (S4.2.1) or his agent (S4.2.2) can file a sealing-off request by activating the role `Conceal-request`(*which, who, start, end*):

(S4.2.1)
canActivate(pat, `Conceal-request`($what, who, start, end$)) ←
  hasActivated(pat, `Patient`()),
  count-conceal-requests(n, pat),
  $what = (pat, ids, orgs, authors, subjects, \textit{from-time}, \textit{to-time})$,
  $who = (\textit{orgs1}, \textit{readers1}, \textit{spctys1})$,
  $n < 100$

(S4.2.2)
canActivate(ag, `Conceal-request`($what, who, start, end$)) ←
  hasActivated(ag, `Agent`(pat)),
  count-conceal-requests(n, pat),
  $what = (pat, ids, orgs, authors, subjects, \textit{from-time}, \textit{to-time})$,
  $who = (\textit{orgs1}, \textit{readers1}, \textit{spctys1})$,
  $n < 100$

For each patient, a maximum of 100 such requests can be activated (S4.2.7):

(S4.2.7)
count-conceal-requests(count$\langle y \rangle$, pat) ←
  hasActivated(x, `Conceal-request`(y)),
  $what = (pat, ids, orgs, authors, subjects, \textit{from-time}, \textit{to-time})$,
  $who = (\textit{orgs1}, \textit{readers1}, \textit{spctys1})$,
  $y = (what, who, start, end)$

Again, we invented this common-sense rule to prevent non-professional users from clogging up the policy.

Apart from a validity time interval [*start*, *end*] (cf. 730.48.12), the `Conceal-request` role specifies which items to seal off and whom the restriction applies to. The specification for *which* items to seal off is a 7-tuple

$$(pat, ids, orgs, authors, subjects, \textit{from-time}, \textit{to-time}).$$

A record item from a patient *pat*'s EHR is sealed off if its ID is in the set *ids*, its author is in the set *authors* and working for a health organisation in *orgs*, if its subject matter is in *subjects*, and its creation date is between *from-time* and *to-time*.

The specification for *whom* the restriction applies to is a triple

$$(orgs, readers, spctys).$$

A user is prevented from accessing the selected items (even if she has a legitimate relationship) if she is in the set *readers*, working for a health organisation (if applicable) in *orgs* in a specialty (if applicable) in the set *spctys*.

With $\mathcal{C}_0$'s universal set expression Ω and the set difference construct, patients can express explicit access permissions (e.g. "only doctors from Addenbrooke's may access items concerning cancer") as well as explicit access denials (e.g. "Dr Littlewood may not access items created after 2005"). Access to items that have not yet been created can also be restricted by setting *ids* to the universal set expression for IDs, and setting *to-time* to the future.

The `Conceal-request` role can also be used to request to withhold items from non-clinicians, including agents and the patient himself[4].

[4]Some patients wish not to be informed about certain particularly distressing subject matters. For example, a patient may specify to make all record items regarding cancer, including those created in the future, inaccessible to himself.

The patient (S4.2.3), his agents (S4.2.4) and his GP (S4.2.5) can all deactivate requests:

(S4.2.3)
$\mathsf{canDeactivate}(pat, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \pi_1^7(what) = pat$

(S4.2.4)
$\mathsf{canDeactivate}(ag, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat)),$
$\quad \pi_1^7(what) = pat$

(S4.2.5)
$\mathsf{canDeactivate}(cli, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{General-practitioner}(pat)),$
$\quad \pi_1^7(what) = pat$

A request is automatically deactivated if the patient's registration is cancelled (S4.2.6):

(S4.2.6)
$\mathsf{isDeactivated}(x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat)),$
$\quad \pi_1^7(what) = pat$

A treating clinician can apply the request by activating a matching `Concealed-by-spine-patient` role (S4.2.8):

(S4.2.8)
$\mathsf{canActivate}(cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Conceal-request}(what, who, start, end))$

This role can be revoked by the activator herself (S4.2.9) or any other clinician working in the same workgroup (S4.2.10):

(S4.2.9)
$\mathsf{canDeactivate}(cli, cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$

(S4.2.10)
$\mathsf{canDeactivate}(cli1, cli2, \texttt{Concealed-by-spine-patient}(what, who, start1, end1)) \leftarrow$
$\quad \mathsf{hasActivated}(cli1, \texttt{Spine-clinician}(ra, org, spcty1)),$
$\quad ra@ra.\mathsf{canActivate}(cli1,$
$\quad\quad \texttt{Group-treating-clinician}(pat, ra, org, group, spcty1)),$
$\quad ra@ra.\mathsf{canActivate}(cli2,$
$\quad\quad \texttt{Group-treating-clinician}(pat, ra, org, group, spcty2))$

The role is automatically revoked if the request is cancelled (S4.2.11):

(S4.2.11)
$\mathsf{isDeactivated}(cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Conceal-request}(what, who, start, end))$

This last rule in combination with S4.2.5 means that the patient's GPs can always remove an access restriction.

A clinician who is granted *authenticated express consent* by a patient may access any sealed-off data without raising an alert if she would also be entitled to access that data, had it not bean sealed off (§730.48.4, §730.48.17). A patient *pat* (S4.3.1), his agent (S4.3.2), or his

GP (S4.3.3) can activate the role `Authenticated-express-consent`(*pat, cli*) for a clinician *cli*:

(S4.3.1)
canActivate(*pat*, `Authenticated-express-consent`(*pat*, *cli*)) ←
    hasActivated(*pat*, `Patient`()),
    count-authenticated-express-consent(*n*, *pat*),
    $n < 100$

(S4.3.2)
canActivate(*ag*, `Authenticated-express-consent`(*pat*, *cli*)) ←
    hasActivated(*ag*, `Agent`(*pat*)),
    count-authenticated-express-consent(*n*, *pat*),
    $n < 100$

(S4.3.3)
canActivate(*cli1*, `Authenticated-express-consent`(*pat*, *cli2*)) ←
    hasActivated(*cli1*, `Spine-clinician`(*ra*, *org*, *spcty*)),
    canActivate(*cli1*, `General-practitioner`(*pat*))

The following aggregation rule restricts the number of such consent grants to 100 for the former two cases of activations (S4.3.8):

(S4.3.8)
count-authenticated-express-consent(count⟨*cli*⟩, *pat*) ←
    hasActivated(*x*, `Authenticated-express-consent`(*pat*, *cli*))

Similarly, consent can be withdrawn by the patient (S4.3.4), his agent(S4.3.5), or his GP (S4.3.6):

(S4.3.4)
canDeactivate(*pat*, *x*, `Authenticated-express-consent`(*pat*, *cli*)) ←
    hasActivated(*pat*, `Patient`())

(S4.3.5)
canDeactivate(*ag*, *x*, `Authenticated-express-consent`(*pat*, *cli*)) ←
    hasActivated(*ag*, `Agent`(*pat*))

(S4.3.6)
canDeactivate(*cli1*, *x*, `Authenticated-express-consent`(*pat*, *cli2*)) ←
    hasActivated(*cli1*, `Spine-clinician`(*ra*, *org*, *spcty*)),
    canActivate(*cli1*, `General-practitioner`(*pat*))

It is automatically withdrawn when the patient's registration is cancelled (S4.3.7):

(S4.3.7)
isDeactivated(*x*, `Authenticated-express-consent`(*pat*, *cli*)) ←
    isDeactivated(*y*, `Register-patient`(*pat*))

5.2.6 Access permissions

A new record item for a patient can be created by treating clinicians performing the action `Add-spine-record-item`(*pat*) (S5.1.1):

(S5.1.1)
permits(*cli*, `Add-spine-record-item`(*pat*)) ←
    hasActivated(*cli*, `Spine-clinician`(*ra*, *org*, *spcty*)),
    canActivate(*cli*, `Treating-clinician`(*pat*, *org*, *spcty*))

Patients may not add items to their records themselves, but they can add comments to existing items by performing `Annotate-spine-record-item`(*pat, id*) (S5.1.2):

(S5.1.2)
$\mathsf{permits}(pat, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}())$

Comments can also be added by a patient's agent (S5.1.3) or a treating clinician (S5.1.4) on his behalf (§730.59.6):

(S5.1.3)
$\mathsf{permits}(ag, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat))$

(S5.1.4)
$\mathsf{permits}(pat, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

Patients (S5.2.1), their agents (S5.2.2) and treating clinicians (S5.2.3) can get a list of all record item IDs of the patient by performing the action `Get-spine-record-item-ids`(*pat*):

(S5.2.1)
$\mathsf{permits}(pat, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}())$

(S5.2.2)
$\mathsf{permits}(ag, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat))$

(S5.2.3)
$\mathsf{permits}(cli, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

Depending on system implementation, each ID may be annotated with additional, non-confidential information about that item, such as date etc.

A patient (S5.3.1) or his agent (S5.3.2) can read a record item from the patient's record by performing the action `Read-spine-record-item`(*pat, id*) if the patient has given one-off consent to make his data available:

(S5.3.1)
$\mathsf{permits}(pat, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \mathsf{hasActivated}(x, \texttt{One-off-consent}(pat)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}clinician}(m, pat, id),$
$\quad \mathsf{third\text{-}party\text{-}consent}(consenters, pat, id),$
$\quad n = 0,$
$\quad m = 0,$
$\quad a = (pat, id),$
$\quad b = (\texttt{No-org}, pat, \texttt{No-spcty}),$
$\quad \texttt{Get-spine-record-third-parties}(pat, id) \subseteq consenters$

(S5.3.2)
$\mathsf{permits}(ag, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(ag, \texttt{Agent}(pat))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b)$,
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}clinician}(m, pat, id)$,
$\mathsf{third\text{-}party\text{-}consent}(consenters, pat, id)$,
$n = 0$,
$m = 0$,
$a = (pat, id)$,
$b = (\texttt{No-org}, ag, \texttt{No-spcty})$,
$\texttt{Get-spine-record-third-parties}(pat, id) \subseteq consenters$

The former rules authorise read access only if the items have not been sealed off. The check is implemented by the use of aggregation rules (S4.1.6, S4.2.12):

(S4.1.6)
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}clinician}(\mathsf{count}\langle x\rangle, pat, id) \leftarrow$
$\mathsf{hasActivated}(x, \texttt{Concealed-by-spine-clinician}(pat, ids, start, end))$,
$id \in ids$,
$\texttt{Current-time}() \in [start, end]$

(S4.2.12)
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(\mathsf{count}\langle x\rangle, a, b) \leftarrow$
$\mathsf{hasActivated}(x, \texttt{Concealed-by-spine-patient}(what, who, start, end))$,
$a = (pat, id)$,
$b = (org, reader, spcty)$,
$what = (pat, ids, orgs, authors, subjects, from\text{-}time, to\text{-}time)$,
$whom = (orgs1, readers1, spctys1)$,
$\texttt{Get-spine-record-org}(pat, id) \in orgs$,
$\texttt{Get-spine-record-author}(pat, id) \in authors$,
$sub \in \texttt{Get-spine-record-subjects}(pat, id)$,
$sub \in subjects$,
$\texttt{Get-spine-record-time}(pat, id) \in [from\text{-}time, to\text{-}time]$,
$id \in ids$,
$org \in orgs1$,
$reader \in readers1$,
$spcty \in spctys1$,
$\texttt{Current-time}() \in [start, end]$,
$\texttt{Get-spine-record-third-parties}(pat, id) = \{\}$,
$\texttt{non-clinical} \in \Omega - \texttt{Get-spine-record-subjects}(pat, id)$

It is also checked that all relevant third parties have given consent to disclosure (S2.2.17) (§730.20.9, §730.56):

(S2.2.17)
$\mathsf{third\text{-}party\text{-}consent}(\mathsf{group}\langle consenter\rangle, pat, id) \leftarrow$
$\mathsf{hasActivated}(x, \texttt{Third-party-consent}(consenter, pat, id))$

The author of a record item can always read it herself as long the patient has given his one-off consent, even if it has been sealed off by the patient (S5.3.3):

(S5.3.3)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\texttt{Get-spine-record-org}(pat, id) = org$,
$\texttt{Get-spine-record-author}(pat, id) = cli$

A treating clinician may view the item if the patient has given his one-off consent, if it has not been sealed off by the patient and if her specialty allows her[5] to read items regarding the subject-matters of the item (S5.3.4):

(S5.3.4)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{One-off-consent}(pat)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b),$
$\quad n = 0,$
$\quad a = (pat, id),$
$\quad b = (org, cli, spcty),$
$\quad \texttt{Get-spine-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

If the item is sealed off by the patient, she is only permitted to read it with authenticated express consent (S5.3.5) (§730.48.21).

(S5.3.5)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{One-off-consent}(pat)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)),$
$\quad \mathsf{hasActivated}(y, \texttt{Authenticated-express-consent}(pat, cli)),$
$\quad \texttt{Get-spine-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

The OBS (§730.48.17, §730.4.11) allows clinicians with a legitimate relationship to access a patient's item even if it has been sealed off by the patient, and even if the patient has not given any one-off consent (S5.3.6):

(S5.3.6)
$\mathsf{permits}(cli, \texttt{Force-read-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

Of course, clinicians are meant to 'break the seal' and access the item only in exceptional circumstances. The action `Force-read-spine-record-item`(*pat, id*) should trigger an alert in a real implementation and be marked in the audit trail. There is no override facility for patients to access data that has been sealed off by a clinician (§730.51.14).

5.3 Patient Demographic Service

The PDS will provide users with various search functions on patient demographic data. Our PDS policy only implements the minimal functionality required to interoperate with the Spine.

5.3.1 Main roles and patient registration

The main roles on the PDS are `PDS-manager`, `Patient`, `Agent`, and `Professional-user`. As before, a user may be active in only one such role at a time. This separation-of-duties

[5] We associate a set of permitted subjects for each specialty with the built-in function `Permitted-subjects`(*spcty*). For example, the specialty "dentistry" may allow the subjects "general" and "dental", but not "venereal disease".

constraint is enforced by the use of aggregation rules (P1.5.1, P1.1.4, P1.2.4, P1.3.5, P1.4.6):

(P1.5.1)
no-main-role-active($user$) ←
    count-agent-activations(n, $user$),
    count-patient-activations(n, $user$),
    count-PDS-manager-activations(n, $user$),
    count-preofessional-user-activations(n, $user$),
    $n = 0$

(P1.1.4)
count-PDS-manager-activations(count$\langle u \rangle$, $user$) ←
    hasActivated($user$, PDS-manager())

(P1.2.4)
count-patient-activations(count$\langle u \rangle$, $user$) ←
    hasActivated($user$, Patient())

(P1.3.5)
count-agent-activations(count$\langle u \rangle$, $user$) ←
    hasActivated($user$, Agent(pat))

(P1.4.6)
count-professional-user-activations(count$\langle u \rangle$, $user$) ←
    hasActivated($user$, Professional-user(ra, org))

Any user active in a role can deactivate their own role (P1.1.2, P1.2.2, P1.3.2, P1.4.5):

(P1.1.2)
canDeactivate(adm, adm, PDS-manager()) ←

(P1.2.2)
canDeactivate(pat, pat, Patient()) ←

(P1.3.2)
canDeactivate(ag, ag, Agent(pat)) ←

(P1.4.5)
canDeactivate(x, x, Professional-user(ra, org)) ←

A user can activate the PDS-manager() role if she is registered as a manager (P1.1.1):

(P1.1.1)
canActivate(adm, PDS-manager()) ←
    hasActivated(x, Register-PDS-manager(adm)),
    no-main-role-active(adm)

A manager can delegate the PDS-manager role to another user usr by activating Register-PDS-manager(usr) if usr has not already been registered by another manager (P1.1.5, P1.1.7):

(P1.1.5)
canActivate($adm1$, Register-PDS-manager($adm2$)) ←
    hasActivated($adm1$, PDS-manager()),
    pds-admin-regs(0, $adm2$)

(P1.1.7)
pds-admin-regs(count$\langle x \rangle$, adm) ←
    hasActivated(x, Register-PDS-manager(adm))

Managers can also cancel manager registrations (P1.1.6), thereby potentially triggering the deactivation of any matching active manager role (P1.1.3):

(P1.1.6)
$\mathsf{canDeactivate}(adm1, x, \texttt{Register-PDS-manager}(adm2)) \leftarrow$
    $\mathsf{hasActivated}(adm1, \texttt{PDS-manager}())$

(P1.1.3)
$\mathsf{isDeactivated}(adm, \texttt{PDS-manager}()) \leftarrow$
    $\mathsf{isDeactivated}(x, \texttt{Register-PDS-manager}(adm))$

In our policy, the manager role is designated for registering patients and storing their demographic data. A manager can register a patient *pat* who has not yet been registered so far by activating $\texttt{Register-patient}(pat)$ (P2.1.1, P2.1.3):

(P2.1.1)
$\mathsf{canActivate}(adm, \texttt{Register-patient}(pat)) \leftarrow$
    $\mathsf{hasActivated}(adm, \texttt{PDS-manager}())$,
    $\mathsf{patient\text{-}regs}(0, pat)$

(P2.1.3)
$\mathsf{patient\text{-}regs}(\mathsf{count}\langle x\rangle, pat) \leftarrow$
    $\mathsf{hasActivated}(x, \texttt{Register-patient}(pat))$

Every patient in the country will be associated with such a registration role activation. Managers can also deactivate patient registrations (P2.1.2):

(P2.1.2)
$\mathsf{canDeactivate}(adm, x, \texttt{Register-patient}(pat)) \leftarrow$
    $\mathsf{hasActivated}(adm, \texttt{PDS-manager}())$

Patients can activate the $\texttt{Patient}()$ role on the PDS if they are registered. An agent can activate the $\texttt{Agent}(pat)$ role if he himself is a registered patient, and the Spine confirms that he is an agent (P1.3.1):

(P1.3.1)
$\mathsf{canActivate}(ag, \texttt{Agent}(pat)) \leftarrow$
    $\mathsf{hasActivated}(x, \texttt{Register-patient}(ag))$,
    $\mathsf{no\text{-}main\text{-}role\text{-}active}(ag)$,
    $\mathsf{Spine@Spine.canActivate}(ag, \texttt{Agent}(pat))$

Patient and their agent roles are deactivated if their registrations are cancelled (P1.2.3, P1.3.3, P1.3.4):

(P1.2.3)
$\mathsf{isDeactivated}(pat, \texttt{Patient}()) \leftarrow$
    $\mathsf{isDeactivated}(x, \texttt{Register-patient}(pat))$

(P1.3.3)
$\mathsf{isDeactivated}(ag, \texttt{Agent}(pat)) \leftarrow$
    $\mathsf{isDeactivated}(x, \texttt{Register-patient}(ag))$

(P1.3.4)
$\mathsf{isDeactivated}(ag, \texttt{Agent}(pat)) \leftarrow$
    $\mathsf{isDeactivated}(x, \texttt{Register-patient}(pat))$

Our policy only exemplifies two kinds of professional users, clinicians and Caldicott Guardians, but other professional roles could easily be implemented in a similar fashion.

A user submitting a currently valid RA-issued clinician or Caldicott Guardian credential can activate the `Professional-user`(*ra, org*) role (P1.4.1, P1.4.3):

(P1.4.1)
canActivate(x, `Professional-user`(ra, org)) ←
    no-main-role-active(cli),
    ra.hasActivated(x, `NHS-clinician-cert`(org, cli, $spcty$, $start$, end)),
    canActivate(ra, `Registration-authority`()),
    `Current-time`() ∈ [$start$, end]

(P1.4.3)
canActivate(x, `Professional-user`(ra, org)) ←
    no-main-role-active(cg),
    ra.hasActivated(x, `NHS-Caldicott-guardian-cert`(org, cg, $start$, end)),
    canActivate(ra, `Registration-authority`()),
    `Current-time`() ∈ [$start$, end]

If no valid credential is submitted, it is requested directly from the RA (P1.4.2, P1.4.4).

(P1.4.2)
canActivate(x, `Professional-user`(ra, org)) ←
    no-main-role-active(cli),
    ra@ra.hasActivated(x, `NHS-clinician-cert`(org, cli, $spcty$, $start$, end)),
    canActivate(ra, `Registration-authority`()),
    `Current-time`() ∈ [$start$, end]

(P1.4.4)
canActivate(x, `Professional-user`(ra, org)) ←
    no-main-role-active(cg),
    ra@ra.hasActivated(x, `NHS-Caldicott-guardian-cert`(org, cg, $start$, end)),
    canActivate(ra, `Registration-authority`()),
    `Current-time`() ∈ [$start$, end]

In all cases, it is checked whether the RA is approved by the NHS (P1.5.2, P1.5.3):

(P1.5.2)
canActivate(ra, `Registration-authority`()) ←
    `NHS`.hasActivated(x, `NHS-registration-authority`(ra, $start$, end)),
    `Current-time`() ∈ [$start$, end]

(P1.5.3)
canActivate(ra, `Registration-authority`()) ←
    ra@`NHS`.hasActivated(x, `NHS-registration-authority`(ra, $start$, end)),
    `Current-time`() ∈ [$start$, end]

5.3.2 Patient-registration credentials

Patients are authenticated on the Spine and other applications after the PDS is contacted for confirmation of the patient's registration status. A request sent to the PDS for this purpose asks for the credential of the form `PDS`.hasActivated(x, `Register-patient`(pat)).

The PDS policy allows patient-registration-credential requests from patients (P2.2.1), agents (P2.2.2) and professional users (P2.2.3) who have activated their respective roles:

(P2.2.1)
canReqCred(pat, `PDS`.hasActivated(x, `Register-patient`(pat))) ←
    hasActivated(pat, `Patient`())

(P2.2.2)
canReqCred(ag, `PDS`.hasActivated(x, `Register-patient`(pat))) ←
    hasActivated(ag, `Agent`(pat))

(P2.2.3)
canReqCred(usr, PDS.hasActivated(x, Register-patient(pat))) ←
    hasActivated(usr, Professional-user(ra, org))

Credential requests are further granted to health organisations certified by an RA (P2.2.4). If no RA-issued health organisation credential is submitted, the PDS will try to retrieve it from the health organisation directly (P2.5.5):

(P2.2.4)
canReqCred(org, PDS.hasActivated(x, Register-patient(pat))) ←
    ra.hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    canActivate(ra, Registration-authority())

(P2.2.5)
canReqCred(org, PDS.hasActivated(x, Register-patient(pat))) ←
    org@ra.hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    canActivate(ra, Registration-authority())

Lastly, patient-registration credentials can also be revealed to RAs (P2.2.6) and the Spine (P2.2.7):

(P2.2.6)
canReqCred(ra, PDS.hasActivated(x, Register-patient(pat))) ←
    canActivate(ra, Registration-authority())

(P2.2.7)
canReqCred(Spine, PDS.hasActivated(x, Register-patient(pat))) ←

5.4 Local health organisations

The policy of our exemplary health organisation, Addenbrooke's Hospital (ADB), illustrates the access control principles of an EPR system. It also shows how a local application can collaborate with the Spine and other national services. For example, clinical workgroups can be managed locally, and local workgroup membership can be used to gain access to EHR items on the Spine. Or conversely, to be authenticated as a patient's agent on the hospital's system, the local policy can make use of the Spine's agent registration facilities.

As large parts of the ABD's policy are very similar to the Spine's, the following description will go into less detail and focus on the main differences. The full set of rules can be found in Appendix A.3.

5.4.1 Main roles

ADB's policy defines seven main roles, Clinician, Caldicott-guardian, HR-manager, Receptionist, Patient, Agent, Ext-treating-clinician and Third-party. As in the policies for the other services, aggregation rules ensure that only one main role can be activated at a time (A1.7.1, A1.1.7, A1.2.7, A1.3.7, A1.4.7, A1.5.7, A1.6.4, A2.2.5, A2.3.11).

The staff roles

- Clinician($spcty$) (A1.1.4–7),
- Caldicott-guardian() (A1.2.4–7),
- HR-manager() (A1.3.4–7), and
- Receptionist() (A1.4.4–7)

can be activated by a user if they have been registered (or appointed) by a human-resource manager. The corresponding registration roles are

- Register-clinician(usr, $spcty$) (A1.1.1–3),
- Register-Caldicott-guardian(usr) (A1.2.1–3),
- Register-HR-manager(usr) (A1.3.1–3), and

- `Register-receptionist`(usr) (A1.4.1–3).

Users in these staff roles can deactivate their own roles. Their roles are automatically deactivated when their corresponding registration role is deactivated, which can only be done by a human-resource manager.

Similarly, patients are registered by receptionists via the `Register-patient`(pat) role (A1.5.1–3) upon which they can activate their `Patient`() role (A1.5.4–7). The activation rule also checks if the patient is registered on the PDS.

Agents can be registered via the `Register-agent`(usr, pat) by both patients and Caldicott Guardians (A1.6.5–10), upon which the `Agent`(pat) role can be activated (A1.6.1–4). A user can also become an agent at ADB without registration if he is registered as an agent on the Spine (A1.6.2).

5.4.2 Caldicott Guardians

Caldicott Guardians are responsible for safeguarding the confidentiality of patient information in a health organisation. They are expected to check the audit trails for possible misconduct and to investigate events that trigger an alarm, e.g. a clinician reading a restricted item or assuming the role of an emergency clinician. They can also give consent on behalf of a patient, or, in exceptional circumstances, make decisions against the wishes of the patient.

In ADB's policy, a Caldicott Guardian has the power to lift access restrictions imposed by patients or clinicians (A4.1.4, A4.2.5):

(A4.1.4)
$\mathsf{canDeactivate}(cg, cli, \texttt{Concealed-by-clinician}(pat, id, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

(A4.2.5)
$\mathsf{canDeactivate}(cg, x, \texttt{Concealed-by-patient}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

Caldicott Guardians can give consent to referral of patients (A2.1.10):

(A2.1.10)
$\mathsf{canActivate}(cg, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad \mathsf{hasActivated}(x, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty))$

They can further give consent to access third-party information (A2.3.4, A2.3.17):

(A2.3.4)
$\mathsf{canActivate}(cg, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad x \in \texttt{Get-record-third-parties}(pat, id)$

(A2.3.17)
$\mathsf{canActivate}(cg, \texttt{Third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad \mathsf{hasActivated}(y, \texttt{Request-third-party-consent}(x, pat, id))$

They have the right to appoint and revoke agents for patients (A1.6.6, A1.6.8):

(A1.6.6)
$\mathsf{canActivate}(cg, \texttt{Register-agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad \mathsf{canActivate}(pat, \texttt{Patient}())$

(A1.6.8)
$\mathsf{canDeactivate}(cg, x, \texttt{Register-agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

Finally, Caldicott Guardians can revoke an emergency clinician's role (A3.7.3):

(A3.7.3)
canDeactivate(cg, cli, Emergency-clinician(pat)) ←
    hasActivated(cg, Caldicott-guardian())

5.4.3 Referrals and external clinicians

When a patient is referred to an external clinician by a treating clinician in the health organisation (e.g. a GP referring a patient to see a specialist), that clinician may need access to the local EPR. In our policy for ADB, such a referral automatically establishes a legitimate relationship that enables the external clinician to access relevant and unrestricted items of the referred patient's EPR by activating the Ext-treating-clinician role (A5.3.5):

(A5.3.5)
permits(cli, Read-record-item(pat, id)) ←
    hasActivated(cli, Ext-treating-clinician(pat, ra, org, $spcty$)),
    count-concealed-by-patient2(n, a, b),
    $n = 0$,
    $a = (pat, id)$,
    $b = (org, cli,$ Ext-group$, spcty)$,
    Get-record-subjects(pat, id) ⊆ Permitted-subjects($spcty$)

In contrast to the referral mechanism on the Spine, ADB's policy requires explicit patient consent. The rationale behind this decision is that an EPR is generally more detailed and possibly more confidential than the EHR. Any local clinician currently treating the patient can file a request Request-consent-to-referral(*pat, ra, org, cli, spcty*) to have the patient referred to an external clinician *cli* working for *org* in specialty *spcty* (A2.1.1):

(A2.1.1)
canActivate($cli1$, Request-consent-to-referral(pat, ra, org, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, Clinician($spcty1$)),
    canActivate($cli1$, ADB-treating-clinician(pat, $team$, $spcty1$))

The request can be withdrawn by the clinician herself (A2.1.2), or denied by the patient (2.1.3), his agent (A2.1.4), or a Caldicott Guardian (A2.1.5):

(A2.1.2)
canDeactivate(cli, cli, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(cli, Clinician($spcty$))

(A2.1.3)
canDeactivate(pat, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(pat, Patient())

(A2.1.4)
canDeactivate(ag, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(ag, Agent(pat))

(A2.1.5)
canDeactivate(cg, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(cg, Caldicott-guardian())

All referral requests are automatically cancelled if the patient is unregistered (A2.1.6):

(A2.1.6)
isDeactivated(x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    isDeactivated(y, Register-patient(pat))

A referral request can be granted by the patient or his agent activating a matching `Consent-to-referral` role (A2.1.8–9):

(A2.1.8)
$\mathsf{canActivate}(pat, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \mathsf{hasActivated}(x, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty))$

(A2.1.9)
$\mathsf{canActivate}(pat, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Agent}(pat)),$
$\quad \mathsf{hasActivated}(x, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty))$

Caldicott Guardians have the power to give consent on behalf of a patient, even against his wishes (A2.1.10):

(A2.1.10)
$\mathsf{canActivate}(cg, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad \mathsf{hasActivated}(x, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty))$

The consent role is automatically deactivated when all matching requests have been denied (A2.1.7, A2.1.11):

(A2.1.11)
$\mathsf{isDeactivated}(x, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty)),$
$\quad \mathsf{other\text{-}consent\text{-}to\text{-}referral\text{-}requests}(0, y, pat, ra, org, cli, spcty)$

(A2.1.7)
$\mathsf{other\text{-}consent\text{-}to\text{-}referral\text{-}requests}(\mathsf{count}\langle y\rangle, x, pat, ra, org, cli, spcty) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{Request-consent-to-referral}(pat, ra, org, cli, spcty)),$
$\quad x \neq y$

Once consent has been given, the external clinician can activate and deactivate her role (A2.2.2, A2.2.3):

(A2.2.2)
$\mathsf{canActivate}(cli, \texttt{Ext-treating-clinician}(pat, ra, org, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(ref, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(cli),$
$\quad ra@ra.\mathsf{hasActivated}(y, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}())$

(A2.2.3)
$\mathsf{canDeactivate}(cli, cli, \texttt{Ext-treating-clinician}(pat, ra, org, spcty)) \leftarrow$

The external clinician is automatically revoked from her role if consent to referral is withdrawn (A2.2.4, A2.1.12):

(A2.2.4)
$\mathsf{isDeactivated}(cli, \texttt{Ext-treating-clinician}(pat, ra, org, spcty)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Consent-to-referral}(pat, ra, org, cli2, spcty)),$
$\quad \mathsf{other\text{-}referral\text{-}consents}(0, x, pat, ra, org, cli, spcty)$

(A2.1.12)
$\mathsf{other\text{-}referral\text{-}consents}(\mathsf{count}\langle y\rangle, x, pat, ra, org, cli, spcty) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{Consent-to-referral}(pat, ra, org, cli, spcty)),$
$\quad x \neq y$

5.4.4 Workgroup management

As is usual in hospitals, we assume that receptionists register new patients and sign them up for treatment. In contrast to the Spine's policy, Addenbrooke's does not require patients to give explicit consent to treatment. Rather, patient treatment is based on workgroups (§730.13).

We distinguish between two different kinds of workgroups, medical teams and wards. A medical team is a group of clinicians collaboratively treating a patient. A typical team may be headed by a consultant, and supported by specialist registrars, senior house officers, and specialist nurses. Additionally, patients in an in-patient episode are usually treated in a ward. A ward is typically run by a head nurse and a group of other nurses.

Every team is headed by at most one current team member (A3.1.1–7), appointed by a human-resource manager to the role `Head-of-team`(*team*). A similar set of rules governs the appointment, activation and deactivation of `Head-of-ward`(*ward*) roles (A3.4.1–7).

Workgroup membership is managed by human resource managers and workgroup leaders via the registration roles `Register-team-member`(*cli, team, spcty*) and `Register-ward-member`(*cli, ward, spcty*) (A3.2.1-7, A3.5.1-7).

A legitimate relationship exists between a clinician and a patient if the clinician is a member of a workgroup and the patient is currently being treated by that workgroup. Workgroup-based treatment of patients is managed by receptionists (A3.3.1, A3.3.4, A3.6.1, A3.6.4):

(A3.3.1)
canActivate(*rec*, `Register-team-episode`(*pat*, *team*)) ←
    hasActivated(*rec*, `Receptionist`()),
    canActivate(*pat*, `Patient`()),
    team-episode-regs(0, *pat*, *team*)

(A3.3.4)
canDeactivate(*rec*, *x*, `Register-team-episode`(*pat*, *team*)) ←
    hasActivated(*rec*, `Receptionist`())

(A3.6.1)
canActivate(*rec*, `Register-ward-episode`(*pat*, *ward*)) ←
    hasActivated(*rec*, `Receptionist`()),
    canActivate(*pat*, `Patient`()),
    ward-episode-regs(0, *pat*, *ward*)

(A3.6.4)
canDeactivate(*rec*, *x*, `Register-ward-episode`(*pat*, *ward*)) ←
    hasActivated(*rec*, `Receptionist`())

Team members and heads of wards can also assign patients to teams or wards, respectively:

(A3.3.2)
canActivate(*cli*, `Register-team-episode`(*pat*, *team*)) ←
    hasActivated(*cli*, `Clinician`(*spcty*)),
    hasActivated(*x*, `Register-team-member`(*cli*, *team*, *spcty*)),
    canActivate(*pat*, `Patient`()),
    team-episode-regs(0, *pat*, *team*)

(A3.3.5)
canDeactivate(*cli*, *x*, `Register-team-episode`(*pat*, *team*)) ←
    hasActivated(*cli*, `Clinician`(*spcty*)),
    hasActivated(*x*, `Register-team-member`(*cli*, *team*, *spcty*))

(A3.6.2)
canActivate(*hd*, `Register-ward-episode`(*pat*, *ward*)) ←
    hasActivated(*hd*, `Clinician`(*spcty*)),
    canActivate(*hd*, `Head-of-ward`(*ward*)),
    canActivate(*pat*, `Patient`()),
    ward-episode-regs(0, *pat*, *ward*)

(A3.6.5)
$\mathsf{canDeactivate}(hd, x, \texttt{Register-ward-episode}(pat, ward)) \leftarrow$
$\quad \mathsf{hasActivated}(hd, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(hd, \texttt{Head-of-ward}(ward))$

Additionally, Caldicott Guardians can cancel workgroup treatment registrations (A3.6.3, A3.3.3), but cannot sign up patients for treatment:

(A3.6.3)
$\mathsf{canDeactivate}(cg, x, \texttt{Register-ward-episode}(pat, ward)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

(A3.3.3)
$\mathsf{canDeactivate}(cg, x, \texttt{Register-team-episode}(pat, team)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

The auxiliary role $\texttt{ADB-treating-clinician}(pat,\ group,\ spcty)$ expresses workgroup-based legitimate relationships, based on the clinician being a group member and the patient registered for a team- or ward-episode (A3.8.1–3).

(A3.8.1)
$\mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty)) \leftarrow$
$\quad \mathsf{canActivate}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Register-team-member}(cli, team, spcty)),$
$\quad \mathsf{hasActivated}(y, \texttt{Register-team-episode}(pat, team)),$
$\quad group = team$

(A3.8.2)
$\mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty)) \leftarrow$
$\quad \mathsf{canActivate}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Register-ward-member}(cli, ward, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Register-ward-episode}(pat, ward)),$
$\quad group = ward$

(A3.8.3)
$\mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Emergency-clinician}(pat)),$
$\quad group = \texttt{A-and-E},$
$\quad spcty = \texttt{A-and-E}$

This role is used as a prerequisite for adding (A5.1.1) and annotating (A5.1.5) EPR items:

(A5.1.1)
$\mathsf{permits}(cli, \texttt{Add-record-item}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty))$

(A5.1.5)
$\mathsf{permits}(pat, \texttt{Annotate-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty))$

Furthermore, it is a prerequisite for reading EPR items (A5.2.3, A5.3.4, A5.3.8):

(A5.2.3)
$\mathsf{permits}(cli, \texttt{Get-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty))$

(A5.3.4)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient2}(n, a, b),$
$\quad n = 0,$
$\quad a = (pat, id),$
$\quad b = (\texttt{ADB}, cli, group, spcty),$
$\quad \texttt{Get-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

(A5.3.8)
$\mathsf{permits}(cli, \texttt{Force-read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty))$

All current members of a medical team have permission to read EPR items that have been authored by that team, even if the patient is currently not treated by the team anymore (A5.3.3):

(A5.3.3)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Register-team-member}(cli, team, spcty)),$
$\quad \texttt{Get-record-group}(pat, id) = team$

Workgroup membership on ADB's system can be used to establish legitimate relationships on the Spine. But as the Spine will contact ADB's RA (RA-ADB) for workgroup credentials (S3.4.1, S3.4.2) (cf. §730.12.0), RA-ADB will in turn request credentials from ADB (R3.1.1, R3.1.2). Therefore, ADB's policy has a canReqCred rule allowing RA-ADB to query the `Register-team-member` and `Register-ward-member` predicates (A1.7.4):

(A1.7.4)
$\mathsf{canReqCred}(x, \texttt{RA-ADB}.\mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start, end))) \leftarrow$
$\quad org = \texttt{ADB}$

5.5 Registration Authorities

RAs are typically local to a particular health organisation but some may also be on a more national level (§730.24.0). We have written a policy for a fictitious RA, RA-ADB, serving Addenbrooke's Foundation Trust and associated hospitals and clinics. As an NHS-approved RA, RA-ADB possesses an RA credential issued by NHS. This credential may be requested by anyone (R1.2.1):

(R1.2.1)
$\mathsf{canReqCred}(x, \texttt{NHS}.\mathsf{hasActivated}(x, \texttt{NHS-registration-authority}(ra, start, end))) \leftarrow$
$\quad ra = \texttt{RA-ADB}$

RAs issue credentials to professional users for the purpose of managing local workgroups (§730.12.0) and the identification, registration and authentication of role membership (§730.9, §730.21). These access roles are subject to national standards yet to be developed by the NHS (§730.12.2). Our RA policy exemplarily defines user access roles only for clinicians and Caldicott Guardians.

RA credentials are required to be time-limited (§730.24.7). Cassandra is flexible enough to encode credentials with validity periods: all RA roles have a start and an end date among their parameters, and the accepting side can specify its own conditions on these dates. For example, it could ignore them, believe them, or impose even stricter freshness conditions (cf. [Riv98]). To authenticate a user's role, the user is issued a credential asserting that someone has activated the corresponding registration role.

5.5.1 Role credential management

The only main role defined in RA-ADB's policy is `RA-manager()`. RA managers sign up professional users for access roles. The `RA-manager` role itself is a standard delegated registration role: a manager can register a person who has not been so far registered as manager (R1.1.1, R1.1.3)

(R1.1.1)
$\mathsf{canActivate}(mgr, \texttt{Register-RA-manager}(mgr2)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$,
$\quad \mathsf{ra\text{-}manager\text{-}regs}(0, mgr2)$

(R1.1.3)
$\mathsf{ra\text{-}manager\text{-}regs}(\mathsf{count}\langle x\rangle, mgr) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Register-RA-manager}(mgr))$

This enables that person to activate (R1.1.4) and deactivate (R1.1.5) a manager role:

(R1.1.4)
$\mathsf{canActivate}(mgr, \texttt{RA-manager}()) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Register-RA-manager}(mgr))$

(R1.1.5)
$\mathsf{canDeactivate}(mgr, mgr, \texttt{RA-manager}()) \leftarrow$

The role is automatically revoked (R1.1.6) if the registration is cancelled by an RA manager (R1.1.2):

(R1.1.6)
$\mathsf{isDeactivated}(mgr, \texttt{RA-manager}()) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Register-RA-manager}(mgr))$

(R1.1.2)
$\mathsf{canDeactivate}(mgr, x, \texttt{Register-RA-manager}(mgr2)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$

To register a person as a certified clinician, an RA manager enters the NHS clinician certification role with parameters identifying the clinician, her health organisation, her specialty and a validity period (R2.1.1):

(R2.1.1)
$\mathsf{canActivate}(mgr, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$,
$\quad \mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start2, end2))$,
$\quad start \in [start2, end2]$,
$\quad end \in [start2, end2]$,
$\quad start < end$

The health organisation must be registered on the same RA, and furthermore, the validity period of its registration must contain that of the clinician. Administrators can grant-independently revoke certifications (R2.1.2):

(R2.1.2)
$\mathsf{canDeactivate}(mgr, x, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$

A clinician certification is automatically cancelled if the clinician's health organisation loses all certifications that are valid within the clinician's validity period (R2.1.3, R2.3.3):

(R2.1.3)
$\mathsf{isDeactivated}(mgr, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad \mathsf{other\text{-}NHS\text{-}health\text{-}org\text{-}regs}(0, x, org, start2, end2),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end$

(R2.3.3)
$\mathsf{other\text{-}NHS\text{-}health\text{-}org\text{-}regs}(\mathsf{count}\langle y \rangle, x, org, start, end) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end,$
$\quad x \neq y \vee start \neq start2 \vee end \neq end2$

The clinician herself, her health organisation, other RAs and the Spine are allowed to request the clinician's credential (R2.1.4–6, R1.2.2–3):

(R2.1.4)
$\mathsf{canReqCred}(org, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\quad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad \texttt{Current-time}() \in [start2, end2]$

(R2.1.5)
$\mathsf{canReqCred}(e, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\quad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$
$\quad \mathsf{canActivate}(e, \texttt{NHS-service}())$

(R2.1.6)
$\mathsf{canReqCred}(cli, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\quad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$

(R1.2.2)
$\mathsf{canActivate}(srv, \texttt{NHS-service}()) \leftarrow$
$\quad \mathsf{canActivate}(srv, \texttt{Registration-authority}())$

(R1.2.3)
$\mathsf{canActivate}(srv, \texttt{NHS-service}()) \leftarrow$
$\quad srv = \texttt{Spine}$

A similar set of policy rules exist for the certification of Caldicott Guardians (R2.2.1–6) and NHS health organisations (R2.3.1–9).

5.5.2 Workgroup credential management

RA-ADB manages workgroup credentials for its registered health organisations, for example for ADB. The Spine can request a `Workgroup-member` credential certifying a clinician's membership in an organisation's team or a ward (R3.1.3):

(R3.1.3)
$\mathsf{canReqCred}(\texttt{Spine}, \texttt{RA-ADB}.\mathsf{canActivate}(cli,$
$\quad \texttt{Workgroup-member}(org, group, spcty))) \leftarrow$

Membership is deduced by first checking whether the organisation is registered at RA-ADB and then requesting a `Register-team-member` or a `Register-ward-member` credential from the organisation (R3.1.1, R3.1.2):

(R3.1.1)
$\mathsf{canActivate}(cli, \texttt{Workgroup-member}(org, group, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{NHS-health-org-cert}(org, start, end)),$
$\quad org@org.\mathsf{hasActivated}(x, \texttt{Register-team-member}(cli, group, spcty)),$
$\quad \texttt{Current-time}() \in [start, end]$

(R3.1.2)
$\mathsf{canActivate}(cli, \texttt{Workgroup-member}(org, group, spcty)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{NHS-health-org-cert}(org, start, end)),$
$\quad org@org.\mathsf{hasActivated}(x, \texttt{Register-ward-member}(cli, group, spcty)),$
$\quad \texttt{Current-time}() \in [start, end]$

6 Discussion

The complexity and constantly evolving nature of the Spine's security and confidentiality requirements necessitate the use of a policy language in order to separate policy from implementation. The policy language must be efficiently machine-enforceable; it must be high-level and sufficiently simple so the policy can be easily modified and read; and it must be expressive and flexible in order to accommodate for current and unforeseeable future requirements.

We have presented a complete Cassandra policy governing access to health records, based on official NHS and DoH documents. The case study shows that Cassandra is sufficiently expressive for the ICRS project and other large-scale real-world applications with highly challenging security requirements. Our preliminary experiments with the policy running on our prototype implementation of Cassandra strongly suggest that the system will also be efficient in practice.

The detailed description of the policy rules could be seen as a translation of the formal Cassandra rules back into plain English. We plan to consult the NHS to see whether the description really matches their requirements and intentions. If approved by the NHS, such a detailed and semi-formal description would be useful for the NHS's IT suppliers, who have only been given the rather sketchy OBS. Also, the description could be given to the public to put the NHS's security policy under public and legal expert scrutiny and could help answering the question whether the proposed Spine will fulfil all legal and ethical confidentiality requirements. In the best case, it could calm the public's unease and relieve the current uncertainty about the project.

One of the lessons learnt from the case study is that the hardest part about writing policy is not the translation into a formal language, but rather understanding the intended requirements. As expected, the available specification documents are unclear, ambiguous, and – above all – incomplete, rather than contradictory. Certainly, many of the gaps could be filled in with common sense. Still, as mentioned above, it will be important to get some official feedback. However, once the requirements are understood and complete, the translation process is relatively straightforward: most of the informal, "intuitive" requirements statements are already approximately of the form *"if* $\langle condition \rangle$ *then* $\langle goal \rangle$*"*.

An obvious question to ask is: is the formal policy correct and does it do what it is supposed to do? Since translating the informal rules into Cassandra was rather straightforward, we believe that the primary source of "incorrectness" would be the requirements in the first place. In such a large and intricate system, it is difficult to fully understand the implications of the requirements. However, having translated the requirements into formal Cassandra policy rules enables us to prove meta-level correctness properties.

Our case study has significant implications for the research area of trust management as a whole. Most other systems have only been applied to relatively simple applications or academic toy examples. There has been a major lack of real-world policy examples that are

both large and complex — our EHR policy example fills this gap. It is a strong counter-example to the claim that real-world applications do not need expressive policy languages. Indeed, it is hard to imagine how the ambitious NHS project could be realised successfully without a flexible, distributed access control system that allows the security policy to be modified easily.

Our policy can be used as a benchmark for existing and future policy languages, as a guideline for language design and as a tool for the difficult task of comparing different policy languages. It would be rewarding to translate the policy into another language, and to analyse which constructs cannot be translated, and which features can perhaps be more easily expressed in a different language.

Future work There is still a lot of work to be done on the case study. Ultimately, we wish to prove the feasibility of our proposed EHR architecture and policy. Once a more complete Cassandra prototype has been implemented, we should conduct further tests of the policy. It can be deemed fully viable only if it exhibits good performance in a realistic setting. We thus need to test its behaviour in a real distributed environment, and with millions of registered users.

We also plan to implement a simple user-friendly Web interface for the Spine. At the moment, the implementation requires the user to type in the raw Cassandra requests on the command line. Even with a somewhat nicer graphical but still generic user interface for Cassandra, the system would be far too complicated for users in practice. Our envisaged interface would have to be specifically designed for the Spine policy and would hide the details of role activations and credentials requests etc. from the user.

Our experiments have highlighted another requirement for policy-based trust management systems that neither our nor existing systems currently fulfil: human users expect textual justifications of access control decisions, especially if their request is denied; they feel rather frustrated and helpless if the answer is simply "request denied", especially if the policy is complex or unknown to the user. Such explanations could be collected from annotations of policy rules used during deduction. The problem is non-trivial as deduction proofs can be long and access denials can have many and far-reaching reasons. More worryingly, the textual justification may reveal more (and perhaps, sensitive) information than could have been deduced from the fact of request denial alone: consider, for example, a response such as "access denied because your daughter has prohibited you from accessing all her records with the subject 'abortion'". Annotating rules with natural-language reasons and then traversing the proof forest to construct the message might be a first step towards a solution.

Acknowledgements I thank Peter Sewell for helpful comments and discussions. This work was funded by a Gates Cambridge Scholarship and a Trinity College Research Scholarship.

A Policy rules for NHS electronic health record system

The following is a complete list of all Cassandra rules for our EHR case study. Appendix A.1 contains the policy rules for the Spine, A.2 for the PDS, A.3 for Addenbrooke's Hospital (ADB), and A.4 contains the rules for Addenbrooke's RA (RA-ADB).

A.1 Policy for the Spine

A.1.1 Main access roles

Clinician

(S1.1.1)
$\mathsf{canActivate}(cli, \texttt{Spine-clinician}(ra, org, spcty)) \leftarrow$
$\quad ra.\mathsf{hasActivated}(x, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}()),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(cli),$
$\quad \texttt{Current-time}() \in [start, end]$

(S1.1.2)
$\mathsf{canActivate}(cli, \texttt{Spine-clinician}(ra, org, spcty)) \leftarrow$
$\quad ra@ra.\mathsf{hasActivated}(x, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)),$
$\quad \mathsf{canActivate}(ra, \texttt{Registration-authority}()),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(cli),$
$\quad \texttt{Current-time}() \in [start, end]$

(S1.1.3)
$\mathsf{canDeactivate}(cli, cli, \texttt{Spine-clinician}(ra, org, spcty)) \leftarrow$

(S1.1.4)
$\mathsf{count\text{-}spine\text{-}clinician\text{-}activations}(\mathsf{count}\langle u\rangle, user) \leftarrow$
$\quad \mathsf{hasActivated}(user, \texttt{Spine-clinician}(ra, org, spcty))$

Administrator

(S1.2.1)
$\mathsf{canActivate}(adm, \texttt{Spine-admin}()) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Register-spine-admin}(adm)),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(adm)$

(S1.2.2)
$\mathsf{canDeactivate}(adm, adm, \texttt{Spine-admin}()) \leftarrow$

(S1.2.3)
$\mathsf{isDeactivated}(adm, \texttt{Spine-admin}()) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Register-spine-admin}(adm))$

(S1.2.4)
$\mathsf{count\text{-}spine\text{-}admin\text{-}activations}(\mathsf{count}\langle u\rangle, user) \leftarrow$
$\quad \mathsf{hasActivated}(user, \texttt{Spine-admin}())$

(S1.2.5)
$\mathsf{canActivate}(adm, \texttt{Register-spine-admin}(adm2)) \leftarrow$
$\quad \mathsf{hasActivated}(adm, \texttt{Spine-admin}()),$
$\quad \mathsf{spine\text{-}admin\text{-}regs}(0, adm2)$

(S1.2.6)
$\mathsf{canDeactivate}(adm, x, \texttt{Register-spine-admin}(adm2)) \leftarrow$
$\quad \mathsf{hasActivated}(adm, \texttt{Spine-admin}())$

(S1.2.7)
$\mathsf{spine\text{-}admin\text{-}regs}(\mathsf{count}\langle x\rangle, adm) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Register-spine-admin}(adm))$

Patient

(S1.3.1)
canActivate(pat, Patient()) ←
    hasActivated(x, Register-patient(pat)),
    no-main-role-active(pat),
    PDS@PDS.hasActivated(y, Register-patient(pat))

(S1.3.2)
canDeactivate(pat, pat, Patient()) ←

(S1.3.3)
isDeactivated(pat, Patient()) ←
    isDeactivated(x, Register-patient(pat))

(S1.3.4)
count-patient-activations(count⟨u⟩, $user$) ←
    hasActivated($user$, Patient())

(S1.3.5)
canActivate(adm, Register-patient(pat)) ←
    hasActivated(adm, Spine-admin()),
    patient-regs(0, pat)

(S1.3.6)
canDeactivate(adm, x, Register-patient(pat)) ←
    hasActivated(adm, Spine-admin())

(S1.3.7)
patient-regs(count⟨x⟩, pat) ←
    hasActivated(x, Register-patient(pat))

Agent

(S1.4.1)
canActivate(ag, Agent(pat)) ←
    hasActivated(x, Register-agent(ag, pat)),
    PDS@PDS.hasActivated(y, Register-patient(ag)),
    no-main-role-active(ag)

(S1.4.2)
canDeactivate(ag, ag, Agent(pat)) ←

(S1.4.3)
isDeactivated(ag, Agent(pat)) ←
    isDeactivated(x, Register-agent(ag, pat)),
    other-agent-regs(0, x, ag, pat)

(S1.4.4)
other-agent-regs(count⟨y⟩, x, ag, pat) ←
    hasActivated(y, Register-agent(ag, pat)),
    $x \neq y$

(S1.4.5)
count-agent-activations(count⟨u⟩, $user$) ←
    hasActivated($user$, Agent(pat))

(S1.4.6)
canReqCred(ag, Spine.canActivate(ag, Agent(pat))) ←
    hasActivated(ag, Agent(pat))

(S1.4.7)
$\mathsf{canReqCred}(org, \mathtt{Spine}.\mathsf{canActivate}(ag, \mathtt{Agent}(pat))) \leftarrow$
$\quad ra.\mathsf{hasActivated}(x, \mathtt{NHS\text{-}health\text{-}org\text{-}cert}(org, start, end)),$
$\quad \mathsf{canActivate}(ra, \mathtt{Registration\text{-}authority}()),$
$\quad \mathtt{Current\text{-}time}() \in [start, end]$

(S1.4.8)
$\mathsf{canReqCred}(org, \mathtt{Spine}.\mathsf{canActivate}(ag, \mathtt{Agent}(pat))) \leftarrow$
$\quad org@ra.\mathsf{hasActivated}(x, \mathtt{NHS\text{-}health\text{-}org\text{-}cert}(org, start, end)),$
$\quad \mathsf{canActivate}(ra, \mathtt{Registration\text{-}authority}()),$
$\quad \mathtt{Current\text{-}time}() \in [start, end]$

(S1.4.9)
$\mathsf{canActivate}(pat, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Patient}()),$
$\quad \mathsf{agent\text{-}regs}(n, pat),$
$\quad n < 3$

(S1.4.10)
$\mathsf{canActivate}(cli, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \mathtt{Spine\text{-}clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \mathtt{General\text{-}practitioner}(pat))$

(S1.4.11)
$\mathsf{canDeactivate}(pat, pat, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Patient}())$

(S1.4.12)
$\mathsf{canDeactivate}(cli, x, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \mathtt{Spine\text{-}clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \mathtt{General\text{-}practitioner}(pat))$

(S1.4.13)
$\mathsf{isDeactivated}(x, \mathtt{Register\text{-}agent}(agent, pat)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \mathtt{Register\text{-}patient}(pat))$

(S1.4.14)
$\mathsf{agent\text{-}regs}(\mathsf{count}\langle x \rangle, pat) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \mathtt{Register\text{-}agent}(x, pat))$

Other

(S1.5.1)
$\mathsf{canActivate}(ra, \mathtt{Registration\text{-}authority}()) \leftarrow$
$\quad \mathtt{NHS}.\mathsf{hasActivated}(x, \mathtt{NHS\text{-}registration\text{-}authority}(ra, start, end)),$
$\quad \mathtt{Current\text{-}time}() \in [start, end]$

(S1.5.2)
$\mathsf{canActivate}(ra, \mathtt{Registration\text{-}authority}()) \leftarrow$
$\quad ra@\mathtt{NHS}.\mathsf{hasActivated}(x, \mathtt{NHS\text{-}registration\text{-}authority}(ra, start, end)),$
$\quad \mathtt{Current\text{-}time}() \in [start, end]$

(S1.5.3)
$\mathsf{no\text{-}main\text{-}role\text{-}active}(user) \leftarrow$
$\quad \mathsf{count\text{-}agent\text{-}activations}(n, user),$
$\quad \mathsf{count\text{-}spine\text{-}clinician\text{-}activations}(n, user),$
$\quad \mathsf{count\text{-}spine\text{-}admin\text{-}activations}(n, user),$
$\quad \mathsf{count\text{-}patient\text{-}activations}(n, user),$
$\quad \mathsf{count\text{-}third\text{-}party\text{-}activations}(n, user),$
$\quad n = 0$

A.1.2 Express consent

One-off consent

(S2.1.1)
canActivate(pat, `One-off-consent`(pat)) ←
    hasActivated(pat, `Patient`())

(S2.1.2)
canActivate(ag, `One-off-consent`(pat)) ←
    hasActivated(ag, `Agent`(pat))

(S2.1.3)
canActivate(cli, `One-off-consent`(pat)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    canActivate(cli, `Treating-clinician`(pat, org, $spcty$))

(S2.1.4)
canDeactivate(pat, x, `One-off-consent`(pat)) ←
    hasActivated(pat, `Patient`())

(S2.1.5)
canDeactivate(ag, x, `One-off-consent`(pat)) ←
    hasActivated(ag, `Agent`(pat))

(S2.1.6)
canDeactivate(cli, x, `One-off-consent`(pat)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    canActivate(cli, `Treating-clinician`(pat, org, $spcty$))

(S2.1.7)
isDeactivated(x, `One-off-consent`(pat)) ←
    isDeactivated(y, `Register-patient`(pat))

Third-party consent

(S2.2.1)
canActivate(pat, `Request-third-party-consent`(x, pat, id)) ←
    hasActivated(pat, `Patient`()),
    $x \in$ `Get-spine-record-third-parties`(pat, id)

(S2.2.2)
canActivate(ag, `Request-third-party-consent`(x, pat, id)) ←
    hasActivated(ag, `Agent`(pat)),
    $x \in$ `Get-spine-record-third-parties`(pat, id)

(S2.2.3)
canActivate(cli, `Request-third-party-consent`(x, pat, id)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    canActivate(cli, `Treating-clinician`(pat, org, $spcty$)),
    $x \in$ `Get-spine-record-third-parties`(pat, id)

(S2.2.4)
canDeactivate(pat, y, `Request-third-party-consent`(x, pat, id)) ←
    hasActivated(pat, `Patient`())

(S2.2.5)
canDeactivate(ag, y, `Request-third-party-consent`(x, pat, id)) ←
    hasActivated(pat, `Agent`(pat))

(S2.2.6)
canDeactivate(cli, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$))

(S2.2.7)
canDeactivate(x, y, Request-third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party())

(S2.2.8)
isDeactivated(x, Request-third-party-consent(y, pat, id)) ←
    isDeactivated(z, Register-patient(pat))

(S2.2.9)
other-third-party-consent-requests(count$\langle x \rangle$, y, z) ←
    hasActivated(x, Request-third-party-consent(z, pat, id)),
    $x \neq y$

(S2.2.10)
canActivate(x, Third-party()) ←
    hasActivated(y, Request-third-party-consent(x, pat, id)),
    no-main-role-active(x),
    PDS@PDS.hasActivated(z, Register-patient(x))

(S2.2.11)
canDeactivate(x, x, Third-party()) ←

(S2.2.12)
isDeactivated(x, Third-party()) ←
    isDeactivated(y, Request-third-party-consent(x, pat, id)),
    other-third-party-consent-requests(0, y, x)

(S2.2.13)
count-third-party-activations(count$\langle u \rangle$, $user$) ←
    hasActivated($user$, Third-party())

(S2.2.14)
canActivate(x, Third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party()),
    hasActivated(y, Request-third-party-consent(x, pat, id))

(S2.2.15)
canActivate(cli, Third-party-consent(x, pat, id)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, Treating-clinician(pat, org, $spcty$)),
    hasActivated(y, Request-third-party-consent(x, pat, id))

(S2.2.16)
isDeactivated(x, Third-party-consent(x, pat, id)) ←
    isDeactivated(y, Request-third-party-consent(x, pat, id)),
    other-third-party-consent-requests(0, y, x)

(S2.2.17)
third-party-consent(group$\langle consenter \rangle$, pat, id) ←
    hasActivated(x, Third-party-consent($consenter$, pat, id))

Consent to treatment

(S2.3.1)
canActivate($cli1$, `Request-consent-to-treatment`(pat, $org2$, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, `Spine-clinician`($ra1$, $org1$, $spcty1$)),
    canActivate($cli2$, `Spine-clinician`($ra2$, $org2$, $spcty2$)),
    canActivate(pat, `Patient`())

(S2.3.2)
canDeactivate($cli1$, $cli1$,
        `Request-consent-to-treatment`(pat, $org2$, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, `Spine-clinician`($ra1$, $org1$, $spcty1$))

(S2.3.3)
canDeactivate($cli2$, $cli1$,
        `Request-consent-to-treatment`(pat, $org2$, $cli2$, $spcty2$)) ←
    hasActivated($cli2$, `Spine-clinician`($ra2$, $org2$, $spcty2$))

(S2.3.4)
canDeactivate(pat, x, `Request-consent-to-treatment`(pat, org, cli, $spcty$)) ←
    hasActivated(pat, `Patient`())

(S2.3.5)
canDeactivate(ag, x, `Request-consent-to-treatment`(pat, org, cli, $spcty$)) ←
    hasActivated(ag, `Agent`(pat))

(S2.3.6)
canDeactivate(cli, x, `Request-consent-to-treatment`(pat, org, $cli2$, $spcty$)) ←
    hasActivated(cli, `Spine-clinician`(ra, org, $spcty$)),
    canActivate(cli, `General-practitioner`(pat))

(S2.3.7)
isDeactivated(x, `Request-consent-to-treatment`(pat, org, cli, $spcty$)) ←
    isDeactivated(y, `Register-patient`(pat))

(S2.3.8)
other-consent-to-treatment-requests(count$\langle y \rangle$, x, pat, org, cli, $spcty$) ←
    hasActivated(y, `Request-consent-to-treatment`(pat, org, cli, $spcty$)),
    $x \neq y$

(S2.3.9)
canActivate(pat, `Consent-to-treatment`(pat, org, cli, $spcty$)) ←
    hasActivated(pat, `Patient`()),
    hasActivated(x, `Request-consent-to-treatment`(pat, org, cli, $spcty$))

(S2.3.10)
canActivate(ag, `Consent-to-treatment`(pat, org, cli, $spcty$)) ←
    hasActivated(ag, `Agent`(pat)),
    hasActivated(x, `Request-consent-to-treatment`(pat, org, cli, $spcty$))

(S2.3.11)
canActivate($cli1$, `Consent-to-treatment`(pat, org, $cli2$, $spcty$)) ←
    hasActivated($cli1$, `Spine-clinician`(ra, org, $spcty$)),
    canActivate($cli1$, `Treating-clinician`(pat, org, $spcty$)),
    hasActivated(x, `Request-consent-to-treatment`(pat, org, $cli2$, $spcty$))

(S2.3.12)
isDeactivated(x, `Consent-to-treatment`(pat, org, cli, $spcty$)) ←
    isDeactivated(y, `Request-consent-to-treatment`(pat, org, cli, $spcty$)),
    other-consent-to-treatment-requests(0, y, pat, org, cli, $spcty$)

Consent to group treatment

(S2.4.1)
canActivate(cli, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(pat, Patient())

(S2.4.2)
canDeactivate(cli, cli, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$))

(S2.4.3)
canDeactivate(pat, x, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(pat, Patient())

(S2.4.4)
canDeactivate(ag, x, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(ag, Agent(pat))

(S2.4.5)
canDeactivate(cli, x, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, General-practitioner(pat))

(S2.4.6)
canDeactivate(cli, x, Request-consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    ra@ra.canActivate(cli, Workgroup-member(org, $group$, $spcty$))

(S2.4.7)
isDeactivated(x, Request-consent-to-group-treatment(pat, org, $group$)) ←
    isDeactivated(y, Register-patient(pat))

(S2.4.8)
other-consent-to-group-treatment-requests(count⟨y⟩, x, pat, org, cli, $spcty$) ←
    hasActivated(y, Request-consent-to-group-treatment(pat, org, $group$)),
    $x \neq y$

(S2.4.9)
canActivate(pat, Consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(pat, Patient()),
    hasActivated(x, Request-consent-to-group-treatment(pat, org, $group$))

(S2.4.10)
canActivate(ag, Consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated(ag, Agent(pat)),
    hasActivated(x, Request-consent-to-group-treatment(pat, org, $group$))

(S2.4.11)
canActivate($cli1$, Consent-to-group-treatment(pat, org, $group$)) ←
    hasActivated($cli1$, Spine-clinician(ra, org, $spcty$)),
    canActivate($cli1$, Treating-clinician(pat, org, $spcty$)),
    hasActivated(x, Request-consent-to-group-treatment(pat, org, $group$))

(S2.4.12)
isDeactivated(x, Consent-to-group-treatment(pat, org, $group$)) ←
    isDeactivated(y, Request-consent-to-group-treatment(pat, org, $group$)),
    other-consent-to-group-treatment-requests(0, y, pat, org, $group$)

A.1.3 Legitimate Relationship

Referral

(S3.1.1)
$\mathsf{canActivate}(\mathit{cli1}, \texttt{Referrer}(\mathit{pat}, \mathit{org}, \mathit{cli2}, \mathit{spcty1})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cli1}, \texttt{Spine-clinician}(\mathit{ra}, \mathit{org}, \mathit{spcty2})),$
$\quad \mathsf{canActivate}(\mathit{cli1}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty2}))$

(S3.1.2)
$\mathsf{canDeactivate}(\mathit{cli1}, \mathit{cli1}, \texttt{Referrer}(\mathit{pat}, \mathit{org}, \mathit{cli2}, \mathit{spcty1})) \leftarrow$

(S3.1.3)
$\mathsf{canDeactivate}(\mathit{pat}, \mathit{cli1}, \texttt{Referrer}(\mathit{pat}, \mathit{org}, \mathit{cli2}, \mathit{spcty1})) \leftarrow$

(S3.1.4)
$\mathsf{isDeactivated}(\mathit{cli1}, \texttt{Referrer}(\mathit{pat}, \mathit{org}, \mathit{cli2}, \mathit{spcty1})) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Register-patient}(\mathit{pat}))$

Emergency clinician

(S3.2.1)
$\mathsf{canActivate}(\mathit{cli}, \texttt{Spine-emergency-clinician}(\mathit{org}, \mathit{pat})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cli}, \texttt{Spine-clinician}(\mathit{ra}, \mathit{org}, \mathit{spcty})),$
$\quad \mathsf{canActivate}(\mathit{pat}, \texttt{Patient}())$

(S3.2.2)
$\mathsf{canDeactivate}(\mathit{cli}, \mathit{cli}, \texttt{Spine-emergency-clinician}(\mathit{org}, \mathit{pat})) \leftarrow$

(S3.2.3)
$\mathsf{isDeactivated}(x, \texttt{Spine-emergency-clinician}(\mathit{org}, \mathit{pat})) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Spine-clinician}(\mathit{ra}, \mathit{org}, \mathit{spcty}))$

(S3.2.4)
$\mathsf{isDeactivated}(x, \texttt{Spine-emergency-clinician}(\mathit{org}, \mathit{pat})) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(\mathit{pat}))$

Treating Clinician & GP

(S3.3.1)
$\mathsf{canActivate}(\mathit{cli}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty})) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Consent-to-treatment}(\mathit{pat}, \mathit{org}, \mathit{cli}, \mathit{spcty}))$

(S3.3.2)
$\mathsf{canActivate}(\mathit{cli}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cli}, \texttt{Spine-emergency-clinician}(\mathit{org}, \mathit{pat})),$
$\quad \mathit{spcty} = \texttt{A-and-E}$

(S3.3.3)
$\mathsf{canActivate}(\mathit{cli}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty})) \leftarrow$
$\quad \mathsf{canActivate}(\mathit{cli}, \texttt{Spine-clinician}(\mathit{ra}, \mathit{org}, \mathit{spcty})),$
$\quad \mathsf{hasActivated}(x, \texttt{Referrer}(\mathit{pat}, \mathit{org}, \mathit{cli}, \mathit{spcty}))$

(S3.3.4)
$\mathsf{canActivate}(\mathit{cli}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty})) \leftarrow$
$\quad \mathsf{canActivate}(\mathit{cli}, \texttt{Group-treating-clinician}(\mathit{pat}, \mathit{ra}, \mathit{org}, \mathit{group}, \mathit{spcty}))$

(S3.3.5)
$\mathsf{canActivate}(\mathit{cli}, \texttt{General-practitioner}(\mathit{pat})) \leftarrow$
$\quad \mathsf{canActivate}(\mathit{cli}, \texttt{Treating-clinician}(\mathit{pat}, \mathit{org}, \mathit{spcty})),$
$\quad \mathit{spcty} = \texttt{GP}$

Workgroup-based LR

(S3.4.1)
canActivate(cli, Group-treating-clinician(pat, ra, org, $group$, $spcty$)) ←
    hasActivated(x, Consent-to-group-treatment(pat, org, $group$)),
    ra.canActivate(cli, Workgroup-member(org, $group$, $spcty$)),
    canActivate(ra, Registration-authority())

(S3.4.2)
canActivate(cli, Group-treating-clinician(pat, ra, org, $group$, $spcty$)) ←
    hasActivated(x, Consent-to-group-treatment(pat, org, $group$)),
    ra@ra.canActivate(cli, Workgroup-member(org, $group$, $spcty$)),
    canActivate(ra, Registration-authority())

A.1.4 Sealing-off data

Access restriction by clinician

(S4.1.1)
canActivate(cli, Concealed-by-spine-clinician(pat, ids, $start$, end)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, Treating-clinician(pat, org, $spcty$))

(S4.1.2)
canDeactivate(cli, cli, Concealed-by-spine-clinician(pat, ids, $start$, end)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$))

(S4.1.3)
canDeactivate(cli, $cli2$, Concealed-by-spine-clinician(pat, ids, $start$, end)) ←
    hasActivated(cli, Spine-clinician(ra, org, $spcty$)),
    canActivate(cli, General-practitioner(pat))

(S4.1.4)
canDeactivate($cli1$, $cli2$, Concealed-by-spine-clinician(pat, ids, $start$, end)) ←
    hasActivated($cli1$, Spine-clinician(ra, org, $spcty1$)),
    canActivate($cli1$, Group-treating-clinician(pat, ra, org, $group$, $spcty1$)),
    canActivate($cli2$, Group-treating-clinician(pat, ra, org, $group$, $spcty2$)),
    hasActivated(x, Consent-to-group-treatment(pat, org, $group$))

(S4.1.5)
isDeactivated(x, Concealed-by-spine-clinician(pat, ids, $start$, end)) ←
    isDeactivated(y, Register-patient(pat))

(S4.1.6)
count-concealed-by-spine-clinician(count⟨x⟩, pat, id) ←
    hasActivated(x, Concealed-by-spine-clinician(pat, ids, $start$, end)),
    $id \in ids$,
    Current-time() $\in [start, end]$

Access restriction by patient

(S4.2.1)
canActivate(pat, Conceal-request($what$, who, $start$, end)) ←
    hasActivated(pat, Patient()),
    count-conceal-requests(n, pat),
    $what = (pat, ids, orgs, authors, subjects, from\text{-}time, to\text{-}time)$,
    $who = (orgs1, readers1, spctys1)$,
    $n < 100$

(S4.2.2)
$\mathsf{canActivate}(ag, \texttt{Conceal-request}(what, who, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat)),$
$\quad \mathsf{count\text{-}conceal\text{-}requests}(n, pat),$
$\quad what = (pat, ids, orgs, authors, subjects, from\text{-}time, to\text{-}time),$
$\quad who = (orgs1, readers1, spctys1),$
$\quad n < 100$

(S4.2.3)
$\mathsf{canDeactivate}(pat, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \pi_1^7(what) = pat$

(S4.2.4)
$\mathsf{canDeactivate}(ag, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat)),$
$\quad \pi_1^7(what) = pat$

(S4.2.5)
$\mathsf{canDeactivate}(cli, x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{General-practitioner}(pat)),$
$\quad \pi_1^7(what) = pat$

(S4.2.6)
$\mathsf{isDeactivated}(x, \texttt{Conceal-request}(what, whom, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat)),$
$\quad \pi_1^7(what) = pat$

(S4.2.7)
$\mathsf{count\text{-}conceal\text{-}requests}(\mathsf{count}\langle y \rangle, pat) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Conceal-request}(y)),$
$\quad what = (pat, ids, orgs, authors, subjects, from\text{-}time, to\text{-}time),$
$\quad who = (orgs1, readers1, spctys1),$
$\quad y = (what, who, start, end)$

(S4.2.8)
$\mathsf{canActivate}(cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Conceal-request}(what, who, start, end))$

(S4.2.9)
$\mathsf{canDeactivate}(cli, cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$

(S4.2.10)
$\mathsf{canDeactivate}(cli1, cli2, \texttt{Concealed-by-spine-patient}(what, who, start1, end1)) \leftarrow$
$\quad \mathsf{hasActivated}(cli1, \texttt{Spine-clinician}(ra, org, spcty1)),$
$\quad ra@ra.\mathsf{canActivate}(cli1,$
$\quad\quad \texttt{Group-treating-clinician}(pat, ra, org, group, spcty1)),$
$\quad ra@ra.\mathsf{canActivate}(cli2,$
$\quad\quad \texttt{Group-treating-clinician}(pat, ra, org, group, spcty2))$

(S4.2.11)
$\mathsf{isDeactivated}(cli, \texttt{Concealed-by-spine-patient}(what, who, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{Conceal-request}(what, who, start, end))$

(S4.2.12)
count-concealed-by-spine-patient(count$\langle x \rangle$, a, b) $\leftarrow$
hasActivated(x, `Concealed-by-spine-patient`($what$, who, $start$, end)),
$a = (pat, id)$,
$b = (org, reader, spcty)$,
$what = (pat, ids, orgs, authors, subjects, from\text{-}time, to\text{-}time)$,
$whom = (orgs1, readers1, spctys1)$,
`Get-spine-record-org`(pat, id) $\in orgs$,
`Get-spine-record-author`(pat, id) $\in authors$,
$sub \in$ `Get-spine-record-subjects`(pat, id),
$sub \in subjects$,
`Get-spine-record-time`(pat, id) $\in [from\text{-}time, to\text{-}time]$,
$id \in ids$,
$org \in orgs1$,
$reader \in readers1$,
$spcty \in spctys1$,
`Current-time`() $\in [start, end]$,
`Get-spine-record-third-parties`(pat, id) $= \{\}$,
`non-clinical` $\in \Omega -$ `Get-spine-record-subjects`(pat, id)

Authenticated express consent

(S4.3.1)
canActivate(pat, `Authenticated-express-consent`(pat, cli)) $\leftarrow$
hasActivated(pat, `Patient`()),
count-authenticated-express-consent(n, pat),
$n < 100$

(S4.3.2)
canActivate(ag, `Authenticated-express-consent`(pat, cli)) $\leftarrow$
hasActivated(ag, `Agent`(pat)),
count-authenticated-express-consent(n, pat),
$n < 100$

(S4.3.3)
canActivate($cli1$, `Authenticated-express-consent`(pat, $cli2$)) $\leftarrow$
hasActivated($cli1$, `Spine-clinician`(ra, org, $spcty$)),
canActivate($cli1$, `General-practitioner`(pat))

(S4.3.4)
canDeactivate(pat, x, `Authenticated-express-consent`(pat, cli)) $\leftarrow$
hasActivated(pat, `Patient`())

(S4.3.5)
canDeactivate(ag, x, `Authenticated-express-consent`(pat, cli)) $\leftarrow$
hasActivated(ag, `Agent`(pat))

(S4.3.6)
canDeactivate($cli1$, x, `Authenticated-express-consent`(pat, $cli2$)) $\leftarrow$
hasActivated($cli1$, `Spine-clinician`(ra, org, $spcty$)),
canActivate($cli1$, `General-practitioner`(pat))

(S4.3.7)
isDeactivated(x, `Authenticated-express-consent`(pat, cli)) $\leftarrow$
isDeactivated(y, `Register-patient`(pat))

(S4.3.8)
count-authenticated-express-consent(count$\langle cli \rangle$, pat) $\leftarrow$
hasActivated(x, `Authenticated-express-consent`(pat, cli))

A.1.5 Access permissions

Adding item

(S5.1.1)
$\mathsf{permits}(cli, \texttt{Add-spine-record-item}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

(S5.1.2)
$\mathsf{permits}(pat, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}())$

(S5.1.3)
$\mathsf{permits}(ag, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat))$

(S5.1.4)
$\mathsf{permits}(pat, \texttt{Annotate-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

Reading item IDs

(S5.2.1)
$\mathsf{permits}(pat, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}())$

(S5.2.2)
$\mathsf{permits}(ag, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat))$

(S5.2.3)
$\mathsf{permits}(cli, \texttt{Get-spine-record-item-ids}(pat)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

Reading items

(S5.3.1)
$\mathsf{permits}(pat, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \mathsf{hasActivated}(x, \texttt{One-off-consent}(pat)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}clinician}(m, pat, id),$
$\quad \mathsf{third\text{-}party\text{-}consent}(consenters, pat, id),$
$\quad n = 0,$
$\quad m = 0,$
$\quad a = (pat, id),$
$\quad b = (\texttt{No-org}, pat, \texttt{No-spcty}),$
$\quad \texttt{Get-spine-record-third-parties}(pat, id) \subseteq consenters$

(S5.3.2)
$\mathsf{permits}(ag, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(ag, \texttt{Agent}(pat))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b)$,
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}clinician}(m, pat, id)$,
$\mathsf{third\text{-}party\text{-}consent}(consenters, pat, id)$,
$n = 0$,
$m = 0$,
$a = (pat, id)$,
$b = (\texttt{No-org}, ag, \texttt{No-spcty})$,
$\texttt{Get-spine-record-third-parties}(pat, id) \subseteq consenters$

(S5.3.3)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\texttt{Get-spine-record-org}(pat, id) = org$,
$\texttt{Get-spine-record-author}(pat, id) = cli$

(S5.3.4)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$,
$\mathsf{count\text{-}concealed\text{-}by\text{-}spine\text{-}patient}(n, a, b)$,
$n = 0$,
$a = (pat, id)$,
$b = (org, cli, spcty)$,
$\texttt{Get-spine-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

(S5.3.5)
$\mathsf{permits}(cli, \texttt{Read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$,
$\mathsf{hasActivated}(x, \texttt{One-off-consent}(pat))$,
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$,
$\mathsf{hasActivated}(y, \texttt{Authenticated-express-consent}(pat, cli))$,
$\texttt{Get-spine-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

(S5.3.6)
$\mathsf{permits}(cli, \texttt{Force-read-spine-record-item}(pat, id)) \leftarrow$
$\mathsf{hasActivated}(cli, \texttt{Spine-clinician}(ra, org, spcty))$,
$\mathsf{canActivate}(cli, \texttt{Treating-clinician}(pat, org, spcty))$

A.2 Policy for Patient Demographic Service

A.2.1 Main roles

Administrator

(P1.1.1)
$\mathsf{canActivate}(adm, \texttt{PDS-manager}()) \leftarrow$
$\mathsf{hasActivated}(x, \texttt{Register-PDS-manager}(adm))$,
$\mathsf{no\text{-}main\text{-}role\text{-}active}(adm)$

(P1.1.2)
$\mathsf{canDeactivate}(adm, adm, \texttt{PDS-manager}()) \leftarrow$

(P1.1.3)
$\mathsf{isDeactivated}(adm, \texttt{PDS-manager}()) \leftarrow$
$\mathsf{isDeactivated}(x, \texttt{Register-PDS-manager}(adm))$

(P1.1.4)
count-PDS-manager-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `PDS-manager`())

(P1.1.5)
canActivate($adm1$, `Register-PDS-manager`($adm2$)) $\leftarrow$
    hasActivated($adm1$, `PDS-manager`()),
    pds-admin-regs(0, $adm2$)

(P1.1.6)
canDeactivate($adm1$, x, `Register-PDS-manager`($adm2$)) $\leftarrow$
    hasActivated($adm1$, `PDS-manager`())

(P1.1.7)
pds-admin-regs(count$\langle x \rangle$, adm) $\leftarrow$
    hasActivated(x, `Register-PDS-manager`(adm))

Patient

(P1.2.1)
canActivate(pat, `Patient`()) $\leftarrow$
    hasActivated(x, `Register-patient`(pat)),
    no-main-role-active(pat)

(P1.2.2)
canDeactivate(pat, pat, `Patient`()) $\leftarrow$

(P1.2.3)
isDeactivated(pat, `Patient`()) $\leftarrow$
    isDeactivated(x, `Register-patient`(pat))

(P1.2.4)
count-patient-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Patient`())

Agent

(P1.3.1)
canActivate(ag, `Agent`(pat)) $\leftarrow$
    hasActivated(x, `Register-patient`(ag)),
    no-main-role-active(ag),
    `Spine@Spine`.canActivate(ag, `Agent`(pat))

(P1.3.2)
canDeactivate(ag, ag, `Agent`(pat)) $\leftarrow$

(P1.3.3)
isDeactivated(ag, `Agent`(pat)) $\leftarrow$
    isDeactivated(x, `Register-patient`(ag))

(P1.3.4)
isDeactivated(ag, `Agent`(pat)) $\leftarrow$
    isDeactivated(x, `Register-patient`(pat))

(P1.3.5)
count-agent-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Agent`(pat))

NHS staff

(P1.4.1)
canActivate(x, Professional-user(ra, org)) ←
    no-main-role-active(cli),
    ra.hasActivated(x, NHS-clinician-cert(org, cli, $spcty$, $start$, end)),
    canActivate(ra, Registration-authority()),
    Current-time() ∈ [$start$, end]

(P1.4.2)
canActivate(x, Professional-user(ra, org)) ←
    no-main-role-active(cli),
    ra@ra.hasActivated(x, NHS-clinician-cert(org, cli, $spcty$, $start$, end)),
    canActivate(ra, Registration-authority()),
    Current-time() ∈ [$start$, end]

(P1.4.3)
canActivate(x, Professional-user(ra, org)) ←
    no-main-role-active(cg),
    ra.hasActivated(x, NHS-Caldicott-guardian-cert(org, cg, $start$, end)),
    canActivate(ra, Registration-authority()),
    Current-time() ∈ [$start$, end]

(P1.4.4)
canActivate(x, Professional-user(ra, org)) ←
    no-main-role-active(cg),
    ra@ra.hasActivated(x, NHS-Caldicott-guardian-cert(org, cg, $start$, end)),
    canActivate(ra, Registration-authority()),
    Current-time() ∈ [$start$, end]

(P1.4.5)
canDeactivate(x, x, Professional-user(ra, org)) ←

(P1.4.6)
count-professional-user-activations(count⟨u⟩, $user$) ←
    hasActivated($user$, Professional-user(ra, org))

Other

(P1.5.1)
no-main-role-active($user$) ←
    count-agent-activations(n, $user$),
    count-patient-activations(n, $user$),
    count-PDS-manager-activations(n, $user$),
    count-preofessional-user-activations(n, $user$),
    $n = 0$

(P1.5.2)
canActivate(ra, Registration-authority()) ←
    NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

(P1.5.3)
canActivate(ra, Registration-authority()) ←
    ra@NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

A.2.2 Patient registration

Registration

(P2.1.1)
canActivate(adm, Register-patient(pat)) ←
    hasActivated(adm, PDS-manager()),
    patient-regs(0, pat)

(P2.1.2)
canDeactivate(adm, x, Register-patient(pat)) ←
    hasActivated(adm, PDS-manager())

(P2.1.3)
patient-regs(count$\langle x \rangle$, pat) ←
    hasActivated(x, Register-patient(pat))

Credentials

(P2.2.1)
canReqCred(pat, PDS.hasActivated(x, Register-patient(pat))) ←
    hasActivated(pat, Patient())

(P2.2.2)
canReqCred(ag, PDS.hasActivated(x, Register-patient(pat))) ←
    hasActivated(ag, Agent(pat))

(P2.2.3)
canReqCred(usr, PDS.hasActivated(x, Register-patient(pat))) ←
    hasActivated(usr, Professional-user(ra, org))

(P2.2.4)
canReqCred(org, PDS.hasActivated(x, Register-patient(pat))) ←
    ra.hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    canActivate(ra, Registration-authority())

(P2.2.5)
canReqCred(org, PDS.hasActivated(x, Register-patient(pat))) ←
    $org@ra$.hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    canActivate(ra, Registration-authority())

(P2.2.6)
canReqCred(ra, PDS.hasActivated(x, Register-patient(pat))) ←
    canActivate(ra, Registration-authority())

(P2.2.7)
canReqCred(Spine, PDS.hasActivated(x, Register-patient(pat))) ←

A.3 Policy for Addenbrooke's Hospital

A.3.1 Main access roles

Clinician

(A1.1.1)
canActivate(mgr, Register-clinician(cli, $spcty$)) ←
    hasActivated(mgr, HR-mgr()),
    clinician-regs(0, cli, $spcty$)

(A1.1.2)
canDeactivate(mgr, x, Register-clinician(cli, $spcty$)) ←
    hasActivated(mgr, HR-mgr())

(A1.1.3)
clinician-regs(count$\langle x \rangle$, cli, $spcty$) $\leftarrow$
    hasActivated(x, `Register-clinician`(cli, $spcty$))

(A1.1.4)
canActivate(cli, `Clinician`($spcty$)) $\leftarrow$
    hasActivated(x, `Register-clinician`(cli, $spcty$)),
    no-main-role-active(cli)

(A1.1.5)
canDeactivate(cli, cli, `Clinician`($spcty$)) $\leftarrow$

(A1.1.6)
isDeactivated(cli, `Clinician`($spcty$)) $\leftarrow$
    isDeactivated(x, `Register-clinician`(cli, $spcty$))

(A1.1.7)
count-clinician-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Clinician`($spcty$))

Caldicott Guardian

(A1.2.1)
canActivate(mgr, `Register-Caldicott-guardian`(cg)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`()),
    cg-regs(0, cg)

(A1.2.2)
canDeactivate(mgr, x, `Register-Caldicott-guardian`(cg)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`())

(A1.2.3)
cg-regs(count$\langle x \rangle$, cg) $\leftarrow$
    hasActivated(x, `Register-Caldicott-guardian`(cg))

(A1.2.4)
canActivate(cg, `Caldicott-guardian`()) $\leftarrow$
    hasActivated(x, `Register-Caldicott-guardian`(cg)),
    no-main-role-active(cg)

(A1.2.5)
canDeactivate(cg, cg, `Caldicott-guardian`()) $\leftarrow$

(A1.2.6)
isDeactivated(cg, `Caldicott-guardian`()) $\leftarrow$
    isDeactivated(x, `Register-Caldicott-guardian`(cg))

(A1.2.7)
count-caldicott-guardian-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Caldicott-guardian`())

HR manager

(A1.3.1)
canActivate(mgr, `Register-HR-mgr`($mgr2$)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`()),
    hr-manager-regs(0, mgr)

(A1.3.2)
canDeactivate(mgr, x, `Register-HR-mgr`($mgr2$)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`())

(A1.3.3)
hr-manager-regs(count$\langle x \rangle$, mgr) $\leftarrow$
    hasActivated(x, `Register-HR-mgr`(mgr))

(A1.3.4)
canActivate(mgr, `HR-mgr`()) $\leftarrow$
    hasActivated(x, `Register-HR-mgr`(mgr)),
    no-main-role-active(mgr)

(A1.3.5)
canDeactivate(mgr, mgr, `HR-mgr`()) $\leftarrow$

(A1.3.6)
isDeactivated(mgr, `HR-mgr`()) $\leftarrow$
    isDeactivated(x, `Register-HR-mgr`(mgr))

(A1.3.7)
count-hr-mgr-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `HR-mgr`())

Receptionist

(A1.4.1)
canActivate(mgr, `Register-receptionist`(rec)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`()),
    receptionist-regs(0, rec)

(A1.4.2)
canDeactivate(mgr, x, `Register-receptionist`(rec)) $\leftarrow$
    hasActivated(mgr, `HR-mgr`())

(A1.4.3)
receptionist-regs(count$\langle x \rangle$, rec) $\leftarrow$
    hasActivated(x, `Register-receptionist`(rec))

(A1.4.4)
canActivate(rec, `Receptionist`()) $\leftarrow$
    hasActivated(x, `Register-receptionist`(rec))

(A1.4.5)
canDeactivate(rec, rec, `Receptionist`()) $\leftarrow$

(A1.4.6)
isDeactivated(rec, `Receptionist`()) $\leftarrow$
    isDeactivated(x, `Register-receptionist`(rec)),
    no-main-role-active(rec)

(A1.4.7)
count-receptionist-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Receptionist`())

Patient

(A1.5.1)
canActivate(rec, `Register-patient`(pat)) $\leftarrow$
    hasActivated(rec, `Receptionist`()),
    patient-regs(0, pat)

(A1.5.2)
canDeactivate(rec, x, `Register-patient`(pat)) $\leftarrow$
    hasActivated(rec, `Receptionist`())

(A1.5.3)
patient-regs(count$\langle x \rangle$, pat) $\leftarrow$
    hasActivated(x, `Register-patient`(pat))

(A1.5.4)
canActivate(pat, `Patient`()) $\leftarrow$
    hasActivated(x, `Register-patient`(pat)),
    no-main-role-active(pat),
    `PDS@PDS`.hasActivated(y, `Register-patient`(pat))

(A1.5.5)
canDeactivate(pat, pat, `Patient`()) $\leftarrow$

(A1.5.6)
isDeactivated(pat, `Patient`()) $\leftarrow$
    isDeactivated(x, `Register-patient`(pat))

(A1.5.7)
count-patient-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Patient`())

Agent

(A1.6.1)
canActivate($agent$, `Agent`(pat)) $\leftarrow$
    hasActivated(x, `Register-agent`($agent$, pat)),
    `PDS@PDS`.hasActivated(x, `Register-patient`($agent$)),
    no-main-role-active($agent$)

(A1.6.2)
canActivate($agent$, `Agent`(pat)) $\leftarrow$
    canActivate(pat, `Patient`()),
    no-main-role-active($agent$),
    `PDS@PDS`.hasActivated(x, `Register-patient`($agent$)),
    `Spine@Spine`.canActivate($agent$, `Agent`(pat))

(A1.6.3)
isDeactivated(ag, `Agent`(pat)) $\leftarrow$
    isDeactivated(x, `Register-agent`(ag, pat)),
    other-agent-regs(0, x, ag, pat)

(A1.6.4)
count-agent-activations(count$\langle u \rangle$, $user$) $\leftarrow$
    hasActivated($user$, `Agent`(pat))

(A1.6.5)
canActivate(pat, `Register-agent`($agent$, pat)) $\leftarrow$
    hasActivated(pat, `Patient`())

(A1.6.6)
canActivate(cg, `Register-agent`($agent$, pat)) $\leftarrow$
    hasActivated(cg, `Caldicott-guardian`()),
    canActivate(pat, `Patient`())

(A1.6.7)
canDeactivate(pat, pat, `Register-agent`($agent$, pat)) $\leftarrow$
    hasActivated(pat, `Patient`())

(A1.6.8)
canDeactivate(cg, x, `Register-agent`($agent$, pat)) $\leftarrow$
    hasActivated(cg, `Caldicott-guardian`())

(A1.6.9)
isDeactivated(x, Register-agent($agent$, pat)) ←
    isDeactivated(y, Register-patient(pat))

(A1.6.10)
other-agent-regs(count⟨y⟩, x, ag, pat) ←
    hasActivated(y, Register-agent(ag, pat)),
    $x \neq y$

Other

(A1.7.1)
no-main-role-active($user$) ←
    count-agent-activations(n, $user$),
    count-caldicott-guardian-activations(n, $user$),
    count-clinician-activations(n, $user$),
    count-ext-treating-clinician-activations(n, $user$),
    count-hr-mgr-activations(n, $user$),
    count-patient-activations(n, $user$),
    count-receptionist-activations(n, $user$),
    count-third-party-activations(n, $user$),
    $n = 0$

(A1.7.2)
canActivate(ra, Registration-authority()) ←
    NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

(A1.7.3)
canActivate(ra, Registration-authority()) ←
    ra@NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

(A1.7.4)
canReqCred(x, RA-ADB.hasActivated(y, NHS-health-org-cert(org, $start$, end))) ←
    org = ADB

A.3.2 Consent and referrals

Consent to referral

(A2.1.1)
canActivate($cli1$, Request-consent-to-referral(pat, ra, org, $cli2$, $spcty2$)) ←
    hasActivated($cli1$, Clinician($spcty1$)),
    canActivate($cli1$, ADB-treating-clinician(pat, $team$, $spcty1$))

(A2.1.2)
canDeactivate(cli, cli, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(cli, Clinician($spcty$))

(A2.1.3)
canDeactivate(pat, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(pat, Patient())

(A2.1.4)
canDeactivate(ag, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(ag, Agent(pat))

(A2.1.5)
canDeactivate(cg, x, Request-consent-to-referral(pat, ra, org, cli, $spcty$)) ←
    hasActivated(cg, Caldicott-guardian())

(A2.1.6)
isDeactivated(x, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$)) ←
isDeactivated(y, `Register-patient`(pat))

(A2.1.7)
other-consent-to-referral-requests(count$\langle y \rangle$, x, pat, ra, org, cli, $spcty$) ←
hasActivated(y, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$)),
$x \neq y$

(A2.1.8)
canActivate(pat, `Consent-to-referral`(pat, ra, org, cli, $spcty$)) ←
hasActivated(pat, `Patient`()),
hasActivated(x, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$))

(A2.1.9)
canActivate(pat, `Consent-to-referral`(pat, ra, org, cli, $spcty$)) ←
hasActivated(pat, `Agent`(pat)),
hasActivated(x, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$))

(A2.1.10)
canActivate(cg, `Consent-to-referral`(pat, ra, org, cli, $spcty$)) ←
hasActivated(cg, `Caldicott-guardian`()),
hasActivated(x, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$))

(A2.1.11)
isDeactivated(x, `Consent-to-referral`(pat, ra, org, cli, $spcty$)) ←
isDeactivated(y, `Request-consent-to-referral`(pat, ra, org, cli, $spcty$)),
other-consent-to-referral-requests(0, y, pat, ra, org, cli, $spcty$)

(A2.1.12)
other-referral-consents(count$\langle y \rangle$, x, pat, ra, org, cli, $spcty$) ←
hasActivated(y, `Consent-to-referral`(pat, ra, org, cli, $spcty$)),
$x \neq y$

External clinician

(A2.2.1)
canActivate(cli, `Ext-treating-clinician`(pat, ra, org, $spcty$)) ←
hasActivated(x, `Consent-to-referral`(pat, ra, org, cli, $spcty$)),
no-main-role-active(cli),
ra.hasActivated(y, `NHS-clinician-cert`(org, cli, $spcty$, $start$, end)),
canActivate(ra, `Registration-authority`())

(A2.2.2)
canActivate(cli, `Ext-treating-clinician`(pat, ra, org, $spcty$)) ←
hasActivated(ref, `Consent-to-referral`(pat, ra, org, cli, $spcty$)),
no-main-role-active(cli),
ra@ra.hasActivated(y, `NHS-clinician-cert`(org, cli, $spcty$, $start$, end)),
canActivate(ra, `Registration-authority`())

(A2.2.3)
canDeactivate(cli, cli, `Ext-treating-clinician`(pat, ra, org, $spcty$)) ←

(A2.2.4)
isDeactivated(cli, `Ext-treating-clinician`(pat, ra, org, $spcty$)) ←
isDeactivated(x, `Consent-to-referral`(pat, ra, org, $cli2$, $spcty$)),
other-referral-consents(0, x, pat, ra, org, cli, $spcty$)

(A2.2.5)
count-ext-treating-clinician-activations(count$\langle u \rangle$, $user$) ←
hasActivated($user$, `Ext-treating-clinician`(pat, ra, org, $spcty$))

Third-party consent

(A2.3.1)
$\mathsf{canActivate}(pat, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad x \in \texttt{Get-record-third-parties}(pat, id)$

(A2.3.2)
$\mathsf{canActivate}(ag, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat)),$
$\quad x \in \texttt{Get-record-third-parties}(pat, id)$

(A2.3.3)
$\mathsf{canActivate}(cli, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad x \in \texttt{Get-record-third-parties}(pat, id)$

(A2.3.4)
$\mathsf{canActivate}(cg, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}()),$
$\quad x \in \texttt{Get-record-third-parties}(pat, id)$

(A2.3.5)
$\mathsf{canDeactivate}(pat, pat, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}())$

(A2.3.6)
$\mathsf{canDeactivate}(ag, ag, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Agent}(pat))$

(A2.3.7)
$\mathsf{canDeactivate}(cli, cli, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty))$

(A2.3.8)
$\mathsf{canDeactivate}(cg, x, \texttt{Request-third-party-consent}(y, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

(A2.3.9)
$\mathsf{canDeactivate}(x, y, \texttt{Request-third-party-consent}(x, pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Third-party}())$

(A2.3.10)
$\mathsf{isDeactivated}(x, \texttt{Request-third-party-consent}(x2, pat, id)) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(pat))$

(A2.3.11)
$\mathsf{count\text{-}third\text{-}party\text{-}activations}(\mathsf{count}\langle u\rangle, user) \leftarrow$
$\quad \mathsf{hasActivated}(user, \texttt{Third-party}())$

(A2.3.12)
$\mathsf{canActivate}(x, \texttt{Third-party}()) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{Request-third-party-consent}(x, pat, id)),$
$\quad \mathsf{no\text{-}main\text{-}role\text{-}active}(x),$
$\quad \texttt{PDS@PDS}.\mathsf{hasActivated}(z, \texttt{Register-patient}(x))$

(A2.3.13)
$\mathsf{canDeactivate}(x, x, \texttt{Third-party}()) \leftarrow$

(A2.3.14)
$\mathsf{other\text{-}third\text{-}party\text{-}requests}(\mathsf{count}\langle y\rangle, x, \textit{third-party}) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{Request-third-party-consent}(\textit{third-party}, pat, id)),$
$\quad x \neq y$

(A2.3.15)
isDeactivated(x, Third-party()) ←
    isDeactivated(y, Request-third-party-consent(x, pat, id)),
    other-third-party-requests(0, y, x)

(A2.3.16)
canActivate(x, Third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party()),
    hasActivated(y, Request-third-party-consent(x, pat, id))

(A2.3.17)
canActivate(cg, Third-party-consent(x, pat, id)) ←
    hasActivated(cg, Caldicott-guardian()),
    hasActivated(y, Request-third-party-consent(x, pat, id))

(A2.3.18)
canDeactivate(x, x, Third-party-consent(x, pat, id)) ←
    hasActivated(x, Third-party())

(A2.3.19)
canDeactivate(cg, x, Third-party-consent(x, pat, id)) ←
    hasActivated(cg, Caldicott-guardian())

(A2.3.20)
isDeactivated(x, Third-party-consent(x, pat, id)) ←
    isDeactivated(y, Register-patient(pat))

(A2.3.21)
third-party-consent(group⟨$consenter$⟩, pat, id) ←
    hasActivated(x, Third-party-consent($consenter$, pat, id))

A.3.3 LR and clinical teams

Head of team

(A3.1.1)
canActivate(hd, Head-of-team($team$)) ←
    hasActivated(x, Register-head-of-team(hd, $team$))

(A3.1.2)
canDeactivate(hd, hd, Head-of-team($team$)) ←

(A3.1.3)
isDeactivated(hd, Head-of-team($team$)) ←
    isDeactivated(x, Register-head-of-team(hd, $team$))

(A3.1.4)
canActivate(mgr, Register-head-of-team(hd, $team$)) ←
    hasActivated(mgr, HR-mgr()),
    hasActivated(x, Register-team-member(hd, $team$, $spcty$)),
    head-of-team-regs(0, hd, $team$)

(A3.1.5)
canDeactivate(mgr, x, Register-head-of-team(hd, $team$)) ←
    hasActivated(mgr, HR-mgr())

(A3.1.6)
isDeactivated(x, Register-head-of-team(hd, $team$)) ←
    isDeactivated(y, Register-team-member(hd, $team$, $spcty$))

(A3.1.7)
head-of-team-regs(count⟨x⟩, hd, $team$) ←
    hasActivated(x, Register-head-of-team(hd, $team$))

Team membership

(A3.2.1)
canActivate(mgr, Register-team-member(mem, $team$, $spcty$)) ←
    hasActivated(mgr, HR-mgr()),
    canActivate(mem, Clinician($spcty$)),
    team-member-regs(0, mem, $team$, $spcty$)

(A3.2.2)
canActivate(hd, Register-team-member(mem, $team$, $spcty$)) ←
    hasActivated(hd, Clinician($spcty2$)),
    canActivate(hd, Head-of-team($team$)),
    canActivate(mem, Clinician($spcty$)),
    team-member-regs(0, mem, $team$, $spcty$)

(A3.2.3)
canDeactivate(mgr, x, Register-team-member(mem, $team$, $spcty$)) ←
    hasActivated(mgr, HR-mgr())

(A3.2.4)
canDeactivate(hd, x, Register-team-member(mem, $team$, $spcty$)) ←
    hasActivated(hd, Clinician($spcty2$)),
    canActivate(hd, Head-of-team($team$))

(A3.2.5)
isDeactivated(x, Register-team-member(mem, $team$, $spcty$)) ←
    isDeactivated(y, Register-clinician(mem, $spcty$))

(A3.2.6)
canReqCred(ra, ADB.hasActivated(x, Register-team-member(cli, $team$, $spcty$))) ←
    ra = RA-ADB

(A3.2.7)
team-member-regs(count⟨x⟩, mem, $team$, $spcty$) ←
    hasActivated(x, Register-team-member(mem, $team$, $spcty$))

Team episode

(A3.3.1)
canActivate(rec, Register-team-episode(pat, $team$)) ←
    hasActivated(rec, Receptionist()),
    canActivate(pat, Patient()),
    team-episode-regs(0, pat, $team$)

(A3.3.2)
canActivate(cli, Register-team-episode(pat, $team$)) ←
    hasActivated(cli, Clinician($spcty$)),
    hasActivated(x, Register-team-member(cli, $team$, $spcty$)),
    canActivate(pat, Patient()),
    team-episode-regs(0, pat, $team$)

(A3.3.3)
canDeactivate(cg, x, Register-team-episode(pat, $team$)) ←
    hasActivated(cg, Caldicott-guardian())

(A3.3.4)
canDeactivate(rec, x, Register-team-episode(pat, $team$)) ←
    hasActivated(rec, Receptionist())

(A3.3.5)
canDeactivate(cli, x, `Register-team-episode`($pat, team$)) ←
    hasActivated(cli, `Clinician`($spcty$)),
    hasActivated(x, `Register-team-member`($cli, team, spcty$))

(A3.3.6)
isDeactivated(x, `Register-team-episode`($pat, team$)) ←
    isDeactivated(y, `Register-patient`(pat))

(A3.3.7)
team-episode-regs(count$\langle x \rangle, pat, team$) ←
    hasActivated(x, `Register-team-episode`($pat, team$))

Head of ward

(A3.4.1)
canActivate(cli, `Head-of-ward`($ward$)) ←
    hasActivated(x, `Register-head-of-ward`($cli, ward$))

(A3.4.2)
canDeactivate(cli, cli, `Head-of-ward`($ward$)) ←

(A3.4.3)
isDeactivated(cli, `Head-of-ward`($ward$)) ←
    isDeactivated(x, `Register-head-of-ward`($cli, ward$))

(A3.4.4)
canActivate(mgr, `Register-head-of-ward`($cli, ward$)) ←
    hasActivated(mgr, `HR-mgr`()),
    hasActivated(x, `Register-ward-member`($cli, ward, spcty$)),
    head-of-ward-regs(0, $cli, ward$)

(A3.4.5)
canDeactivate(mgr, x, `Register-head-of-ward`($cli, ward$)) ←
    hasActivated(mgr, `HR-mgr`())

(A3.4.6)
isDeactivated(x, `Register-head-of-ward`($cli, ward$)) ←
    isDeactivated(y, `Register-ward-member`($cli, ward, spcty$))

(A3.4.7)
head-of-ward-regs(count$\langle x \rangle, cli, ward$) ←
    hasActivated(x, `Register-head-of-ward`($cli, ward$))

Ward membership

(A3.5.1)
canActivate(mgr, `Register-ward-member`($cli, ward, spcty$)) ←
    hasActivated(mgr, `HR-mgr`()),
    canActivate(cli, `Clinician`($spcty$)),
    ward-member-regs(0, $cli, ward, spcty$)

(A3.5.2)
canActivate(hd, `Register-ward-member`($cli, ward, spcty$)) ←
    hasActivated(cli, `Clinician`($spcty2$)),
    canActivate(hd, `Head-of-ward`($ward$)),
    canActivate(cli, `Clinician`($spcty$)),
    ward-member-regs(0, $cli, ward, spcty$)

(A3.5.3)
canDeactivate(mgr, x, `Register-ward-member`($cli, ward, spcty$)) ←
    hasActivated(mgr, `HR-mgr`())

(A3.5.4)
canDeactivate(hd, x, `Register-ward-member`($cli, ward, spcty$)) ←
    hasActivated(hd, `Clinician`($spcty2$)),
    canActivate(hd, `Head-of-ward`($ward$))

(A3.5.5)
canReqCred(ra, `ADB`.hasActivated(x, `Register-ward-member`($cli, ward, spcty$))) ←
    ra = `RA-ADB`

(A3.5.6)
isDeactivated(x, `Register-ward-member`($cli, ward, spcty$)) ←
    isDeactivated(y, `Register-clinician`($cli, spcty$))

(A3.5.7)
ward-member-regs(count$\langle x \rangle$, $cli, ward, spcty$) ←
    hasActivated(x, `Register-ward-member`($cli, ward, spcty$))

Ward episode

(A3.6.1)
canActivate(rec, `Register-ward-episode`($pat, ward$)) ←
    hasActivated(rec, `Receptionist`()),
    canActivate(pat, `Patient`()),
    ward-episode-regs($0, pat, ward$)

(A3.6.2)
canActivate(hd, `Register-ward-episode`($pat, ward$)) ←
    hasActivated(hd, `Clinician`($spcty$)),
    canActivate(hd, `Head-of-ward`($ward$)),
    canActivate(pat, `Patient`()),
    ward-episode-regs($0, pat, ward$)

(A3.6.3)
canDeactivate(cg, x, `Register-ward-episode`($pat, ward$)) ←
    hasActivated(cg, `Caldicott-guardian`())

(A3.6.4)
canDeactivate(rec, x, `Register-ward-episode`($pat, ward$)) ←
    hasActivated(rec, `Receptionist`())

(A3.6.5)
canDeactivate(hd, x, `Register-ward-episode`($pat, ward$)) ←
    hasActivated(hd, `Clinician`($spcty$)),
    canActivate(hd, `Head-of-ward`($ward$))

(A3.6.6)
isDeactivated(x, `Register-ward-episode`($pat, ward$)) ←
    isDeactivated(y, `Register-patient`(pat))

(A3.6.7)
ward-episode-regs(count$\langle x \rangle$, $pat, ward$) ←
    hasActivated(x, `Register-ward-episode`($pat, ward$))

Emergency clinician

(A3.7.1)
canActivate(cli, Emergency-clinician(pat)) ←
    hasActivated(cli, Clinician($spcty$)),
    canActivate(pat, Patient())

(A3.7.2)
canDeactivate(cli, cli, Emergency-clinician(pat)) ←

(A3.7.3)
canDeactivate(cg, cli, Emergency-clinician(pat)) ←
    hasActivated(cg, Caldicott-guardian())

(A3.7.4)
isDeactivated(x, Emergency-clinician(pat)) ←
    isDeactivated(y, Register-patient(pat))

(A3.7.5)
isDeactivated(x, Emergency-clinician(pat)) ←
    isDeactivated(x, Clinician($spcty$))

(A3.7.6)
is-emergency-clinician(group$\langle x \rangle$, pat) ←
    hasActivated(x, Emergency-clinician(pat))

Treating clinician

(A3.8.1)
canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$)) ←
    canActivate(cli, Clinician($spcty$)),
    hasActivated(x, Register-team-member(cli, $team$, $spcty$)),
    hasActivated(y, Register-team-episode(pat, $team$)),
    $group = team$

(A3.8.2)
canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$)) ←
    canActivate(cli, Clinician($spcty$)),
    hasActivated(x, Register-ward-member(cli, $ward$, $spcty$)),
    hasActivated(x, Register-ward-episode(pat, $ward$)),
    $group = ward$

(A3.8.3)
canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$)) ←
    hasActivated(cli, Emergency-clinician(pat)),
    $group$ = A-and-E,
    $spcty$ = A-and-E

A.3.4 Sealing-off data

Access restriction by clinician

(A4.1.1)
canActivate(cli, Concealed-by-clinician(pat, id, $start$, end)) ←
    hasActivated(cli, Clinician($spcty$)),
    canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$))

(A4.1.2)
canDeactivate(cli, cli, Concealed-by-clinician(pat, id, $start$, end)) ←
    hasActivated(cli, Clinician($spcty$))

(A4.1.3)
$\mathsf{canDeactivate}(\mathit{cli1}, \mathit{cli2}, \texttt{Concealed-by-clinician}(\mathit{pat}, \mathit{id}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cli1}, \texttt{Clinician}(\mathit{spcty1}))$,
$\quad \mathsf{canActivate}(\mathit{cli1}, \texttt{ADB-treating-clinician}(\mathit{pat}, \mathit{group}, \mathit{spcty1}))$,
$\quad \mathsf{canActivate}(\mathit{cli2}, \texttt{ADB-treating-clinician}(\mathit{pat}, \mathit{group}, \mathit{spcty2}))$

(A4.1.4)
$\mathsf{canDeactivate}(\mathit{cg}, \mathit{cli}, \texttt{Concealed-by-clinician}(\mathit{pat}, \mathit{id}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cg}, \texttt{Caldicott-guardian}())$

(A4.1.5)
$\mathsf{isDeactivated}(x, \texttt{Concealed-by-clinician}(\mathit{pat}, \mathit{id}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(\mathit{pat}))$

(A4.1.6)
$\mathsf{count\text{-}concealed\text{-}by\text{-}clinician}(\mathsf{count}\langle x\rangle, \mathit{pat}, \mathit{id}) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Concealed-by-clinician}(\mathit{pat}, \mathit{id}, \mathit{start}, \mathit{end}))$,
$\quad \texttt{Current-time}() \in [\mathit{start}, \mathit{end}]$

Access restriction by patient

(A4.2.1)
$\mathsf{canActivate}(\mathit{pat}, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{who}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{pat}, \texttt{Patient}())$,
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient}(n, \mathit{pat})$,
$\quad \mathit{what} = (\mathit{pat}, \mathit{ids}, \mathit{authors}, \mathit{groups}, \mathit{subjects}, \mathit{from\text{-}time}, \mathit{to\text{-}time})$,
$\quad \mathit{who} = (\mathit{orgs1}, \mathit{readers1}, \mathit{groups1}, \mathit{spctys1})$,
$\quad n < 100$

(A4.2.2)
$\mathsf{canActivate}(\mathit{ag}, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{who}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{ag}, \texttt{Agent}(\mathit{pat}))$,
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient}(n, \mathit{pat})$,
$\quad \mathit{what} = (\mathit{pat}, \mathit{ids}, \mathit{authors}, \mathit{groups}, \mathit{subjects}, \mathit{from\text{-}time}, \mathit{to\text{-}time})$,
$\quad \mathit{who} = (\mathit{orgs1}, \mathit{readers1}, \mathit{groups1}, \mathit{spctys1})$,
$\quad n < 100$

(A4.2.3)
$\mathsf{canDeactivate}(\mathit{pat}, x, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{whom}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{pat}, \texttt{Patient}())$,
$\quad \pi_1^7(\mathit{what}) = \mathit{pat}$

(A4.2.4)
$\mathsf{canDeactivate}(\mathit{ag}, x, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{whom}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{ag}, \texttt{Agent}(\mathit{pat}))$,
$\quad \pi_1^7(\mathit{what}) = \mathit{pat}$

(A4.2.5)
$\mathsf{canDeactivate}(\mathit{cg}, x, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{whom}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{hasActivated}(\mathit{cg}, \texttt{Caldicott-guardian}())$

(A4.2.6)
$\mathsf{isDeactivated}(x, \texttt{Concealed-by-patient}(\mathit{what}, \mathit{whom}, \mathit{start}, \mathit{end})) \leftarrow$
$\quad \mathsf{isDeactivated}(y, \texttt{Register-patient}(\mathit{pat}))$,
$\quad \pi_1^7(\mathit{what}) = \mathit{pat}$

(A4.2.7)
$\mathsf{count\text{-}concealed\text{-}by\text{-}patient}(\mathsf{count}\langle y\rangle, \mathit{pat}) \leftarrow$
$\quad \mathsf{hasActivated}(x, \texttt{Concealed-by-patient}(y))$,
$\quad \mathit{what} = (\mathit{pat}, \mathit{ids}, \mathit{authors}, \mathit{groups}, \mathit{subjects}, \mathit{from\text{-}time}, \mathit{to\text{-}time})$,
$\quad \mathit{who} = (\mathit{orgs1}, \mathit{readers1}, \mathit{groups1}, \mathit{spctys1})$,
$\quad y = (\mathit{what}, \mathit{who}, \mathit{start}, \mathit{end})$

(A4.2.8)
count-concealed-by-patient2(count$\langle x \rangle$, a, b) $\leftarrow$
    hasActivated(x, Concealed-by-patient($what$, $whom$, $start$, end)),
    $a = (pat, id)$,
    $b = (org, reader, group, spcty)$,
    $what = (pat, ids, authors, groups, subjects, from\text{-}time, to\text{-}time)$,
    $whom = (orgs1, readers1, groups1, spctys1)$,
    Get-record-author(pat, id) $\in$ $authors$,
    Get-record-group(pat, id) $\in$ $groups$,
    $sub \in$ Get-record-subjects(pat, id),
    $sub \in subjects$,
    Get-record-time(pat, id) $\in [from\text{-}time, to\text{-}time]$,
    $id \in ids$,
    $org \in orgs1$,
    $reader \in readers1$,
    $group \in groups1$,
    $spcty \in spctys1$,
    Current-time() $\in [start, end]$

A.3.5 Access permissions

Adding item

(A5.1.1)
permits(cli, Add-record-item(pat)) $\leftarrow$
    hasActivated(cli, Clinician($spcty$)),
    canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$))

(A5.1.2)
permits(cli, Add-record-item(pat)) $\leftarrow$
    hasActivated(cli, Ext-treating-clinician(pat, ra, org, $spcty$))

(A5.1.3)
permits(ag, Annotate-record-item(pat, id)) $\leftarrow$
    hasActivated(ag, Agent(pat))

(A5.1.4)
permits(pat, Annotate-record-item(pat, id)) $\leftarrow$
    hasActivated(pat, Patient())

(A5.1.5)
permits(pat, Annotate-record-item(pat, id)) $\leftarrow$
    hasActivated(cli, Clinician($spcty$)),
    canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$))

Reading item IDs

(A5.2.1)
permits(pat, Get-record-item-ids(pat)) $\leftarrow$
    hasActivated(pat, Patient())

(A5.2.2)
permits(ag, Get-record-item-ids(pat)) $\leftarrow$
    hasActivated(ag, Agent(pat))

(A5.2.3)
permits(cli, Get-record-item-ids(pat)) $\leftarrow$
    hasActivated(cli, Clinician($spcty$)),
    canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$))

Reading items

(A5.3.1)
$\mathsf{permits}(ag, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(ag, \texttt{Agent}(pat)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient2}(n, a, b),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}clinician}(m, pat, id),$
$\quad \mathsf{third\text{-}party\text{-}consent}(consenters, pat, id),$
$\quad a = (pat, id),$
$\quad b = (\texttt{No-org}, ag, \texttt{No-group}, \texttt{No-spcty}),$
$\quad n = 0,$
$\quad m = 0,$
$\quad \texttt{Get-record-third-parties}(pat, id) \subseteq consenters$

(A5.3.2)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \texttt{Get-record-author}(pat, id) = cli$

(A5.3.3)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{hasActivated}(x, \texttt{Register-team-member}(cli, team, spcty)),$
$\quad \texttt{Get-record-group}(pat, id) = team$

(A5.3.4)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Clinician}(spcty)),$
$\quad \mathsf{canActivate}(cli, \texttt{ADB-treating-clinician}(pat, group, spcty)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient2}(n, a, b),$
$\quad n = 0,$
$\quad a = (pat, id),$
$\quad b = (\texttt{ADB}, cli, group, spcty),$
$\quad \texttt{Get-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

(A5.3.5)
$\mathsf{permits}(cli, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cli, \texttt{Ext-treating-clinician}(pat, ra, org, spcty)),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient2}(n, a, b),$
$\quad n = 0,$
$\quad a = (pat, id),$
$\quad b = (org, cli, \texttt{Ext-group}, spcty),$
$\quad \texttt{Get-record-subjects}(pat, id) \subseteq \texttt{Permitted-subjects}(spcty)$

(A5.3.6)
$\mathsf{permits}(pat, \texttt{Read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(pat, \texttt{Patient}()),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}patient2}(n, a, b),$
$\quad \mathsf{count\text{-}concealed\text{-}by\text{-}clinician}(m, pat, id),$
$\quad \mathsf{third\text{-}party\text{-}consent}(consenters, pat, id),$
$\quad n = 0,$
$\quad m = 0,$
$\quad a = (pat, id),$
$\quad b = (\texttt{No-org}, pat, \texttt{No-group}, \texttt{No-spcty}),$
$\quad \texttt{Get-record-third-parties}(pat, id) \subseteq consenters$

(A5.3.7)
$\mathsf{permits}(cg, \texttt{Force-read-record-item}(pat, id)) \leftarrow$
$\quad \mathsf{hasActivated}(cg, \texttt{Caldicott-guardian}())$

(A5.3.8)
permits(cli, Force-read-record-item(pat, id)) ←
    hasActivated(cli, Clinician($spcty$)),
    canActivate(cli, ADB-treating-clinician(pat, $group$, $spcty$))

A.4 Policy for Addenbrooke's Registration Authority

A.4.1 Main roles

Administrator

(R1.1.1)
canActivate(mgr, Register-RA-manager($mgr2$)) ←
    hasActivated(mgr, RA-manager()),
    ra-manager-regs(0, $mgr2$)

(R1.1.2)
canDeactivate(mgr, x, Register-RA-manager($mgr2$)) ←
    hasActivated(mgr, RA-manager())

(R1.1.3)
ra-manager-regs(count⟨x⟩, mgr) ←
    hasActivated(x, Register-RA-manager(mgr))

(R1.1.4)
canActivate(mgr, RA-manager()) ←
    hasActivated(x, Register-RA-manager(mgr))

(R1.1.5)
canDeactivate(mgr, mgr, RA-manager()) ←

(R1.1.6)
isDeactivated(mgr, RA-manager()) ←
    isDeactivated(x, Register-RA-manager(mgr))

Other

(R1.2.1)
canReqCred(x, NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end))) ←
    ra = RA-ADB

(R1.2.2)
canActivate(srv, NHS-service()) ←
    canActivate(srv, Registration-authority())

(R1.2.3)
canActivate(srv, NHS-service()) ←
    srv = Spine

(R1.2.4)
canActivate(ra, Registration-authority()) ←
    NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

(R1.2.5)
canActivate(ra, Registration-authority()) ←
    ra@NHS.hasActivated(x, NHS-registration-authority(ra, $start$, end)),
    Current-time() ∈ [$start$, end]

A.4.2 NHS staff authentication

Clinician

(R2.1.1)
$\mathsf{canActivate}(mgr, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}()),$
$\quad \mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end$

(R2.1.2)
$\mathsf{canDeactivate}(mgr, x, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$

(R2.1.3)
$\mathsf{isDeactivated}(mgr, \texttt{NHS-clinician-cert}(org, cli, spcty, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad \mathsf{other\text{-}NHS\text{-}health\text{-}org\text{-}regs}(0, x, org, start2, end2),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end$

(R2.1.4)
$\mathsf{canReqCred}(org, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\qquad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$
$\quad \mathsf{hasActivated}(y, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad \texttt{Current-time}() \in [start2, end2]$

(R2.1.5)
$\mathsf{canReqCred}(e, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\qquad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$
$\quad \mathsf{canActivate}(e, \texttt{NHS-service}())$

(R2.1.6)
$\mathsf{canReqCred}(cli, \texttt{RA-ADB}.\mathsf{hasActivated}(x,$
$\qquad \texttt{NHS-clinician-cert}(org, cli, spcty, start, end))) \leftarrow$

Caldicott Guardian

(R2.2.1)
$\mathsf{canActivate}(mgr, \texttt{NHS-Caldicott-guardian-cert}(org, cg, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}()),$
$\quad \mathsf{hasActivated}(x, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end$

(R2.2.2)
$\mathsf{canDeactivate}(mgr, x, \texttt{NHS-Caldicott-guardian-cert}(org, cg, start, end)) \leftarrow$
$\quad \mathsf{hasActivated}(mgr, \texttt{RA-manager}())$

(R2.2.3)
$\mathsf{isDeactivated}(mgr, \texttt{NHS-Caldicott-guardian-cert}(org, cg, start, end)) \leftarrow$
$\quad \mathsf{isDeactivated}(x, \texttt{NHS-health-org-cert}(org, start2, end2)),$
$\quad \mathsf{other\text{-}NHS\text{-}health\text{-}org\text{-}regs}(0, x, org, start2, end2),$
$\quad start \in [start2, end2],$
$\quad end \in [start2, end2],$
$\quad start < end$

(R2.2.4)
canReqCred(e, RA-ADB.hasActivated(x,
NHS-Caldicott-guardian-cert(org, cg, $start$, end))) ←
$e = cg$

(R2.2.5)
canReqCred(e, RA-ADB.hasActivated(x,
NHS-Caldicott-guardian-cert(org, cg, $start$, end))) ←
hasActivated(y, NHS-health-org-cert(org, $start2$, $end2$)),
$e = org$,
Current-time() $\in [start2, end2]$

(R2.2.6)
canReqCred(e, RA-ADB.hasActivated(x,
NHS-Caldicott-guardian-cert(org, cg, $start$, end))) ←
canActivate(e, NHS-service())

Health organisation

(R2.3.1)
canActivate(mgr, NHS-health-org-cert(org, $start$, end)) ←
hasActivated(mgr, RA-manager())

(R2.3.2)
canDeactivate(mgr, x, NHS-health-org-cert(org, $start$, end)) ←
hasActivated(mgr, RA-manager())

(R2.3.3)
other-NHS-health-org-regs(count$\langle y \rangle$, x, org, $start$, end) ←
hasActivated(y, NHS-health-org-cert(org, $start2$, $end2$)),
$start \in [start2, end2]$,
$end \in [start2, end2]$,
$start < end$,
$x \neq y \vee start \neq start2 \vee end \neq end2$

(R2.3.4)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert(org, $start$, end))) ←
hasActivated(y, NHS-Caldicott-guardian-cert(org, cg, $start2$, $end2$)),
Current-time() $\in [start2, end2]$,
$e = cg$

(R2.3.5)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert(org, $start$, end))) ←
hasActivated(y, NHS-clinician-cert(org, cli, $spcty$, $start2$, $end2$)),
Current-time() $\in [start2, end2]$,
$e = cli$

(R2.3.6)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert(org, $start$, end))) ←
$e = org$

(R2.3.7)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert($org2$, $start$, end))) ←
ra.hasActivated(y, NHS-health-org-cert(org, $start2$, $end2$)),
canActivate(ra, Registration-authority()),
$e = org$

(R2.3.8)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert($org2$, $start$, end))) ←
$org@ra$.hasActivated(y, NHS-health-org-cert(org, $start2$, $end2$)),
canActivate(ra, Registration-authority()),
$e = org$

(R2.3.9)
canReqCred(e, RA-ADB.hasActivated(x, NHS-health-org-cert(org, $start$, end))) ←
    canActivate(e, NHS-service())

A.4.3 Workgroup management

(R3.1.1)
canActivate(cli, Workgroup-member(org, $group$, $spcty$)) ←
    hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    org@org.hasActivated(x, Register-team-member(cli, $group$, $spcty$)),
    Current-time() ∈ [$start$, end]

(R3.1.2)
canActivate(cli, Workgroup-member(org, $group$, $spcty$)) ←
    hasActivated(x, NHS-health-org-cert(org, $start$, end)),
    org@org.hasActivated(x, Register-ward-member(cli, $group$, $spcty$)),
    Current-time() ∈ [$start$, end]

(R3.1.3)
canReqCred(Spine, RA-ADB.canActivate(cli,
        Workgroup-member(org, $group$, $spcty$))) ←

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[And96] Ross Anderson. A security policy model for clinical information systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 30–42, 1996.

[Arn03] Sarah Arnott. Confidentiality is top priority for patients (08/10/03). *Computing*, 2003. See `http://www.computing.co.uk/news/1144171`.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[BMY02] Jean Bacon, Ken Moody, and Walt Yao. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security*, 5(4):492–540, 2002.

[BS04a] Moritz Y. Becker and Peter Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *Policy Workshop*, June 2004.

[BS04b] Moritz Y. Becker and Peter Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *Computer Security Foundations Workshop*, June 2004.

[Col03a] Tony Collins. Doctors express alarm at plans to store patient data without consent (15/07/03). *Computer Weekly*, 2003. See `http://www.computerweekly.com/Article123355.htm`.

[Col03b] Tony Collins. How the national programme came to be the health service's riskiest IT project (16/09/03). *Computer Weekly*, 2003. See `http://www.computerweekly.com/Article124870.htm`.

[Col04] Tony Collins. Gps vote to boycott patient record database (29/06/04). *Computer Weekly*, 2004. See `http://www.computerweekly.com/Article131577.htm`.

[Cor02] Amanda Cornwall. Electronic health records: an international perspective. *Health Issues*, 73, 2002.

[Cro03] Michael Cross. NHS spree revealed (12/06/03). *The Guardian*, 2003. See `http://www.guardian.co.uk/online/story/0,3605,975139,00.html`.

[CS03] Tony Collins and Mike Simons. NHS plan branded a 'farce' (03/06/03). *Computer Weekly*, 2003. See `http://www.computerweekly.com/Article122277.htm`.

[Dep01a] Department of Health, UK. Building the information core: Implementing the NHS plan. 2001.

[Dep01b] Department of Health, UK. Building the information core: Protecting and using confidential patient information. 2001.

[Dep02] Department of Health, UK. Legal and policy constraints on electronic records. 2002.

[EFL+99] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI certificate theory, RFC 2693, September 1999. See `http://www.ietf.org/rfc/rfc2693.txt`.

[Ell99] Carl M. Ellison. SPKI requirements, RFC 2692, September 1999. See `http://www.ietf.org/rfc/rfc2692.txt`.

[Fou03] Foundation for Information Policy Research. NHS confidentiality consultation – FIPR response. February 2003. See `http://www.cl.cam.ac.uk/users/rja14/fiprmedconf.html`.

[Gau03] Nick Gaunt. Confidentiality and consent: Use cases applicable to shared electronic health record. *S&W Devon ERDIP Project*, 2003.

[Haw03] Nigel Hawkes. Patient records go online (21/07/03). *The Times Online*, 2003. See `http://www.timesonline.co.uk/newspaper/0,,2710-751992,00.html`.

[HBM98] R. Hayton, J. Bacon, and K. Moody. OASIS: Access control in an open distributed environment. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 3–14, 1998.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[JMMS98] Joxan Jaffar, Michael J. Maher, Kim Marriott, and Peter J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

[Ley04] John Leyden. Us wins david blunkett lifetime menace award (29/07/04). *The Register*, 2004. See `http://www.theregister.co.uk/2004/07/29/big_brother_awards/`.

[LMW02] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[Nat97] National Health Service, UK. The Caldicott committee: report on the review of patient-identifiable information. 1997.

[Nat98] National Health Service, UK. Information for health: an information strategy for the modern NHS 1998-2005. 1998.

[Nat02] National Health Service, UK. ERDIP evaluation: Technical options for the implementation of electronic health record nationally. 2002.

[Nat03] National Health Service, UK. Integrated Care Records Service: Output based specification version 2. 2003.

[Pal03] Maldwyn Palmer. A complex operation for the NHS spine (14/08/03). *Computer Weekly*, 2003. See `http://www.computerweekly.com/Article124110.htm`.

[Rev02] Peter Revesz. *Introduction to constraint databases*. Springer Verlag, 2002.

[Riv98] Ronald L. Rivest. Can we eliminate certificate revocations lists? In *Financial Cryptography*, pages 178–183, 1998.

[Rog03] James Rogers. GPs voice patient confidentiality concerns (20/05/03). *Computer Weekly*, 2003. See `http://www.computerweekly.com/Article121897.htm`.

[SMW93] Ann Sommerville, Natalie-Jane Macdonald, and R. Weston. *Medical Ethics Today: Its Practice and Philosophy*. British Medical Association, BMJ Publishing Group, 1993.

[YMB02] Walt Yao, Ken Moody, and Jean Bacon. A model of OASIS role-based access control and its support of active security. *ACM Transactions on Information and System Security*, 5(4), 2002.