Distributed Identity Management for Semantic Entities based on Graph Signatures and `owl:sameAs` – Supplementary Materials

Falko Schönteich[1], Ansgar Scherp[2], Andreas Kasten[3]

[1] Christian-Albrechts-Universität, Kiel, Germany
`falko.schoenteich@stu.uni-kiel.de`
[2] Ulm University, Ulm, Germany
`ansgar.scherp@uni-ulm.de`
[3] Debeka, Koblenz, Germany
`andreas.kasten@debeka.de`

The following sections provide supplementary materials. References without the prefix "A." refer to figures or section of the main article, e.,g., Figure 1 refers to the first figure of the main article, while Figure A.1 refers to the first figure in this supplementary materials collection. Section A.1 describes in detail the ontology design patterns of the core ontology for distributed information management. Section A.2 provides a detailed security analysis of semDIM.

A.1 DIM Ontology Patterns: Detailed Description

We apply best practices of ontology patterns and reusing and combining existing core ontologies [7] for representing distributed identities, groups, and roles in semDIM. This ensures strong axiomatization as well as integrateability to systems also building on these ontologies. Especially by using the foundational ontology DOLCE+DnS Ultralight (DUL) [1], our solution provides compatibility to other frameworks relying on DUL-based ontologies, as the common use of shared concepts inherently results in semantic integrateability.

Like in the overview diagram (see Figure 4), in the following pattern figures, we use the following symbols. A gray box indicates a class taken from an external ontology (such as DUL), while white boxes describe classes introduced by semDIM. Open arrowheads, e. g. like in Figure A.2, stand for property relations, while closed triangular arrowheads indicate a superclass. One example of an implementation of the DnS pattern is DUL's Workflow pattern (see Figure A.1). A workflow, a subclass of `Description`, defines roles and their tasks, is satisfied by workflow executions, a situation subclass.

Our approach uses DUL's Descriptions and Situations (DnS) pattern to represent semantics between identities of persons, groups, and roles. DnS allows to create descriptive contexts, i. e., `Descriptions`, defining various `Concepts`, such as `Roles`, `Parameters`, and `EventTypes`. These `Descriptions` may be satisfied by multiple relational contexts, called `Situations`, setting entities for the `Concepts` defined in the respective `Descriptions`. For example, a `Situation` might set an

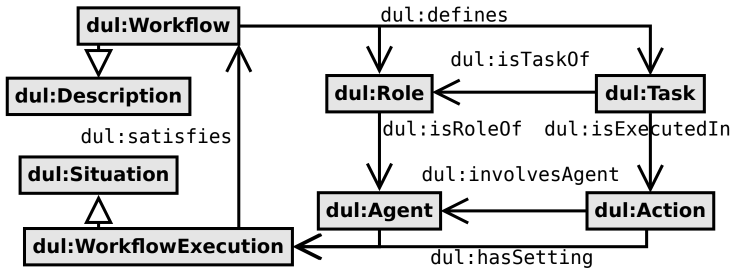

Fig. A.1: DUL Workflow

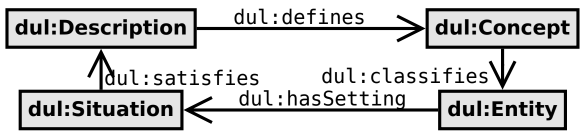

Fig. A.2: DUL DnS Pattern

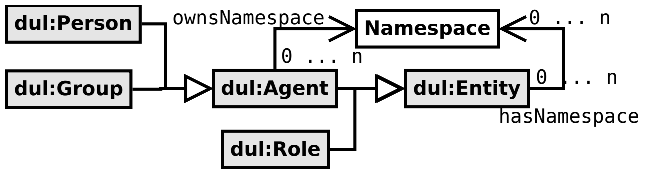

Fig. A.3: Namespace Pattern

Agent classified by a Role, or an Event classified by an EventType. By reusing this pattern, we are able to distinguish between descriptive and relational contexts and integrate our concepts into DUL.

A.1.1 Namespace Pattern for Unique Identification of Entities and Namespace Ownership

For referencing entities across namespaces, we introduce the Namespace Pattern (see Figure A.3). Implicitly, the namespace used in this pattern is identical to the namespace (including any top or sub namespaces) of the entities' URIs. Any solution relying on URIs, e. g., dgFOAF, fulfills R1 at least to some extent. Nonetheless, we explicitly introduce this pattern, as we use the namespace for reasoning, e. g., for checking, if an agent trying to add triples to a namespace has the required ownership.

A.1.2 Same Entity Pattern

Persons, groups, and roles can be connected with the Same Entity Pattern (see Figure A.4). For example, identities referring to the same person can be con-

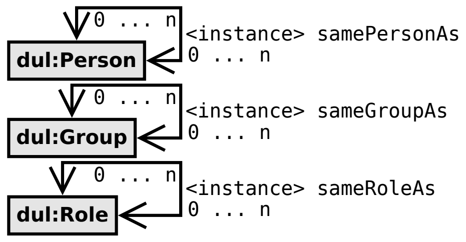

Fig. A.4: Same Entity Pattern

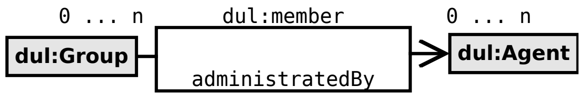

Fig. A.5: Distributed Groups Pattern

nected by stating their semantic equivalence. The properties introduced in Figure A.4 are sub-properties of `owl:sameAs`. The pattern cannot only be applied to identities but also to groups and roles in the same way. Therefore, this pattern also contributes to R3 and R4. But the major part of distributed groups and roles is their management addressed by the next patterns. The `<instance>` annotation indicates, that the `same...As` properties relate to the instances of the respective classes, `Person`, `Group`, and `Role`. The `<instance>` annotation indicates, that this property is used in context of instances and does not infer the classes themselves to be `owl:sameAs or owl:equivalentClass`. The latter is obvious, as classes are always equivalent to themselves.

A.1.3 Distributed Groups Pattern

Distributed groups must be able to contain agents from different namespaces as members (see Figure A.5). For this, we reuse the `dul:hasMember` property from DUL, which has `Group`s superclass `Collection` as domain and `Agent`s superclass `Entity` as range. Also, groups need to be administrated by agents, which we represent with the `administratedBy` property. Again, agents can be persons or groups in this context. As both, `Group`s as well as `Agent`s in this pattern may have different origin namespaces, using this pattern allows management of distributed groups across companies and addresses R3.

A.1.4 Permission Patterns (for Distributed Roles)

In strukt, a `StructuredWorkflow`, a subclass of `Description`, defines `Roles` and their `Tasks` as well as their transitions. It is satisfied by `StructuredWorkflowExecutions`, a `Situation` subclass. We build on strukt's `Workflows` to represent granting of permissions. A `PermissionAssignmentWorkflow` defines the `Role` to be assigned as well as the `Task` related to this `Role`. Also, an `AssignerRole` is defined. It describes which `Role` is needed for the `Assigner`, in order to legitimately propose the permission pattern. Which `AssigernRole` is necessary for an `Assigner` to be allowed to propose a permission pattern depends on the IM policies. The standard policy for DIM states, that only the namespace owner – as defined by the

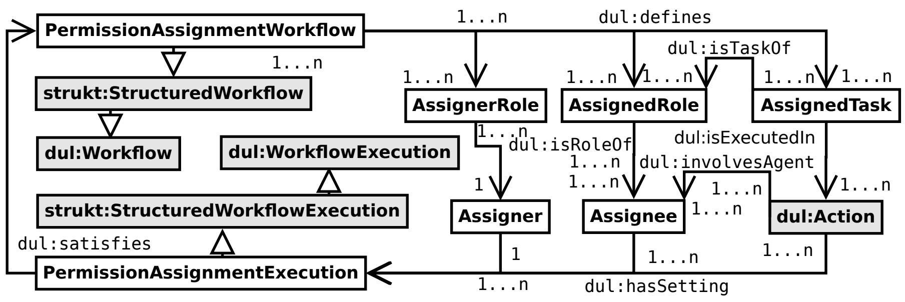

Fig. A.6: Permission Pattern

Namespace pattern from Figure A.3 in Section A.1.1 – of a specific role may assign it. The namespace owner in return may delegate parts of its namespace to a different `Agent`, e. g., the root identity of a company defining departments and delegating the namespace "a:HR" as part of the top namespace"a:" to the head of the HR department. Furthermore, the permission to assign a `Role` can explicitly be transferred. For example, assigning the role `group_admin-1` to `a-1` lets then `a-1` use this `Role` to assign the `Role group-member-1` to any other `Agent` (this example is the same as later shown in Figure A.8). Figure A.6 depicts the Permission Pattern. A `PermissionAssignmentExecution` is the concrete `Situation` for a `PermissionAssignmentWorkflow`. It sets the `Assignee` (receiving the `AssignedRole`) and the `Assigner` (assigning the `AssignedRole` to the `Assignee`). The `Action` in a `PermissionAssignmentExecution` is a placeholder for activities attempted by the `Assignee`. It can be used to reason, if an action is valid in context of the assignment, i. e., the `Action` executes one of the `Tasks` (=`dul:executesTask`) defined in the `PermissionAssignmentWorkflow`.

A `PermissionAssignmentOnGroupWorkflow` is a special type of `PermissionAssignmentWorkflow` related to a certain `Group`. Figure A.7 depicts the Group Permission Pattern. It extends a generic permission pattern by adding `AffectedGroupRole` and `AffectedGroup`. A common example of a group permission is adding/deleting a group member, group administrator, or group owner. semDIM allows custom definition of such permissions, meaning, `Groups` not necessarily need dedicated admins but could be managed by group owners or a namespace owner or a combination of any such. `PermissionAssignmentOnGroupWorkflows` define the `AffectedGroupRole` to which the `AssignedRole` relates to, e. g., `administrated_group_role-1`. Likewise, `PermissionAssignmentOnGroupExecutions` set the concrete `Group` that assignment is valid for.

Figure A.8 shows a group permission example with an admin permission assignment workflow `admin_perm-1` and its execution `admin_assign-1`. `admin_perm-1` defines that the permission assigner role `namespace_owner-1` is allowed to assign the role `group_admin-1`, which has the task `group_administration-1` for the group role `administrated_group_role-1`. With assignment `admin_assign` the assigner `root-a-1` assigns `a-1` the

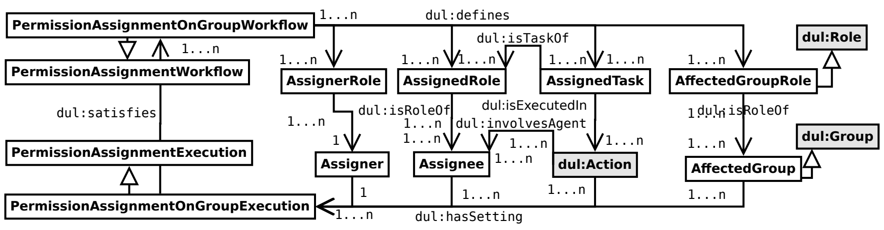

Fig. A.7: Group Permission Pattern

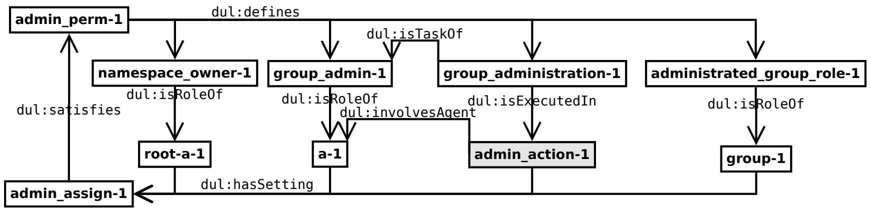

Fig. A.8: Group Permission Admin Example Instantiation

role `group_admin-1` for `group-1`. This can be aggregated to: `a-1 semDIM:administrates group-1`, signed by `root-a-1`.

A.1.5 Signature Pattern, Certificate Pattern, and Graph Signing Pattern

For signing graph data, we rely on the graph signing framework from Kasten et al. [3,4]. For integration into semDIM, we reuse the Signature Pattern, Graph Signing Method Pattern, and Certificate Pattern Agents. The *Signature Pattern* is used to represent a signature. It could be used for the signature of an arbitrary `dul:Thing`, but we use it specifically for signing graphs. We use the *Graph Signing Method Pattern* to represent the various sub-methods required for graph signing, e. g. for serialization or hashing. With the *Certificate Pattern* agents can be assigned PGP or X509 certificates, which can be processes by the signing framework. Figure A.11 shows the Graph Signing Method Pattern. It illustrates, how a signing method consists of various sub-methods, e. g., for serialization or digesting. The Certificate Pattern (see Figure A.10) demonstrates, how an Agent can either have an PGP Certificate or an X509 Certificate, both supported by the signing framework.

A.2 Security Analysis of semDIM

In general, we assume an underlying technical IT infrastructure, managed in compliance to modern security standards, as described by [2,6] or similar standards. This means we will not discuss in the following threats addressing, e. g.,

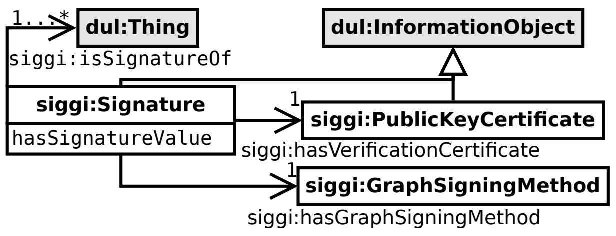

Fig. A.9: Signature Pattern

[t]

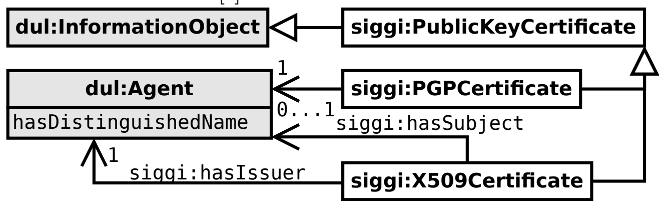

Fig. A.10: Certificate Pattern

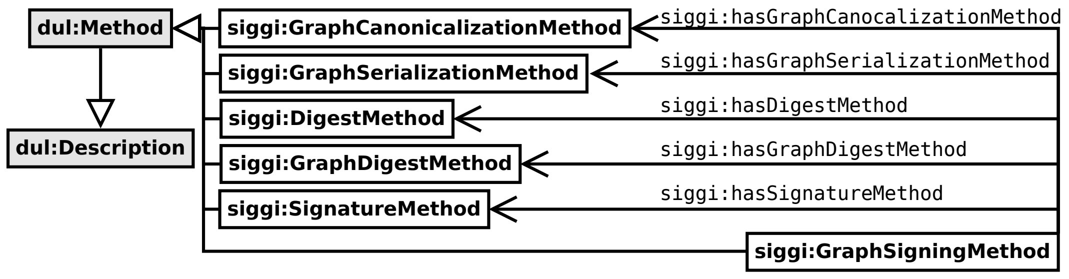
dul:Method
dul:Description
siggi:GraphCanonicalizationMethod
siggi:hasGraphCanocalizationMethod
siggi:GraphSerializationMethod
siggi:hasGraphSerializationMethod
siggi:DigestMethod
siggi:hasDigestMethod
siggi:GraphDigestMethod
siggi:hasGraphDigestMethod
siggi:SignatureMethod
siggi:hasSignatureMethod
siggi:GraphSigningMethod

Fig. A.11: Graph Signing Method Pattern

unpatched operating systems, faulty configured communication channel encryption, or clients pre-infected by remote-access-kits, etc., as these are generic to any kind of IT system. Although our solution can partially provide defense mechanisms even to such fundamental technical threats, we concentrate in the following on threats specific to distributed settings. Below, we discuss the specific security aspects of semDIM along the STRIDE model [5].

A.2.1 Spoofing

Spoofing describes the impersonation of an attacker as another person or program by falsifying data. In our context, spoofing is in the same way prevented as with any other safe transport encryption and robust authentication of sender and receiver. Both, Client-Server as well as Server-Server communication is thereby protected. The servers even exchange certificates before operations on a secure channel (mutual or Two-Way-TLS). This the validity of the certificates is checked, by inspecting the hashes of the exchanged certificates, e. g., by phone, personally, or over another already established encrypted channel, to rule out the possibility, that certificates where manipulated during transmission. As the companies have representatives personally knowing each other, an attacker could not inject an illegitimate certificate, even if he bribed or compromised a Certificate Authority to issue a certificate in one of the companies' names. Such a forged certificate – although technically appearing completely valid – would be regarded as illegit by the companies, as it is not the one they explicitly exchanged during their mutual TLS certificate exchange. Therefore, none of the company relies on a third party for validating the certificate of the other partner, i. e., trusting the Certificate Authority providing the partner's not to be compromised, malicious or bribed. By that, a Man-in-the-Middle (MitM) attack, i. e., a malicious agent intercepting the communication between the partners faking and acting on behalf of both partners' identities, is prevented. To make a MitM attack possible, the attacker would need to exchange the pr-exchanged certificates inside the secure certificate store on the companies servers'. This would require the equivalent of full access infection of both servers; a scenario, which is not a MitM anymore but a total compromise. By using graph signing, we further prevent the potential damages of spoofing. Even if an attacker could impersonate a client or server, he could only transmit signed (valid) messages from the original sender. Replay attacks are prevented by timestamps within the signed graphs. In this paper, we demonstrated for the sake of simplicity Server-Sided signatures, but semDIM also supports Client-Sided signatures. With establishing Client-Sided graph signatures and combining these with mutual TLS between Client and sever, the connection between client and server also can be further protected against spoofing. A certificates used for this signing could also be used for client authentication at the server. Ideally, the certificate of the respective server is also pre-installed on the client, to prevent an attacker from pretending to be the server towards the client. Mutual TLS between client and server is especially beneficial, if the Client module is moved to the server, as suggested as a possible modification of the architecture from Chapter 3. Using mutual TLS here prevents a client from assuming a Client module offered by a compromised servers to be legit and potentially compromising confidential information or even enabling the attacker to forge signatures for illegit graphs in the client's name. Using certificates for Client-sided graph signatures and ideally even using the same certificates for (mutual) TLS could strengthen or even replace the existing organization's authentication method for DIM. However, if reliable authentication is established in the organization, there is no need for a separate DIM-specific authentication. We do not consider spoofing between components on the server as a plausible MitM attack scenario. For this, an attacker would need code execution privilege on the server and fake one of the components on the server. As a code execution privilege usually can be escalated by skilled attacker to full system access, this is not a MitM but a total compromise. Furthermore, it is not reasonable scenario for the server to impersonate a client, as the clients only communicate with their respective server. Lastly, a server cannot spoof another server for being a third server (for the same reason an external attacker cannot do this).

A.2.2 Tampering

Tampering is the intentional manipulation of systems or messages. For external attackers, the protection mechanisms mentioned for Spoofing also apply. As all messages are encrypted, an attacker could neither read nor manipulate the messages. Again, signing the graphs further protects against tempering, as the signatures would not be valid anymore. Besides network communication, tampering is in our context also relevant regarding false claims about, e. g., group membership, assigned privileges or data classification.

The only resting data in our architecture lies on the triple store, which is not directly exposed to the network. Therefore, an attacker must either compromise the server operating system or selectively the request coordinator. Similarly, for manipulating messages to or from the Reasoning or the Sign Engine, the attacker most be able to compromise the server as a whole or the Request Coordinator. Again, we do not consider weaknesses of the OS at this point. The request coordinator, as the only server component exposed to the network, must be hardened for unauthorized manipulation. Ideally, web application firewalls restrict any communication to and from the Request Coordinator to only for exchange of graph data and the necessary commit messages. It is noteworthy, that even in case of a successful tampering attack on resting or transmitted data, the potential damage is greatly reduced by the distributed nature of semDIM. This means, that in case of tampering, a single node's messages or persistent data sets become inconsistent to the global state. Such an inconsistency will likely lead to an increase in aborted transactions, as other nodes deny transactions not compatible with their intact state. Even if only two nodes exist, e. g., as in our scenario setting form Section 3, an inconsistency would lead to aborted transactions alarming administrators to investigate, which of the two nodes is in an inconsistent, i. e., compromised, state. Only if all nodes were successfully manipulated, tampering would not lead to inconsistencies.

A.2.3 Repudiation

Repudiation is the situation where an author can deny his authorship of a statement or a message. In our context, repudiation is relevant as it is important to prove that, e. g., role assignments were really done by the referred author and not somebody else, in case, such an assignment violates, e. g., compliance or legal regulations. As mentioned before, in the above scenario, we only discussed server signatures, thereby preventing companies from repudiating messages. By using client signatures, individual user repudiation could also be prevented.

A.2.4 Information disclosure

Information disclosure happens, if confidential information, most often personal information or intellectual property, can be read by non-legitimate agents.

Again, external attackers can neither read nor manipulate data encrypted by the mutual TLS connection. We address this threat by having our reasoning engine ensure, that classified information, such as confidential product information in our scenario, can only be accessed by legitimate agents. As any server may read the whole project-related IM data set, servers can only expose information unintentionally by accidentally or maliciously transmitting non-project-related IM. This might be information related to other projects the partner is not involved in or purely internal IM data. To prevent this, e. g., in the case of misconfiguration or human error, separation mechanisms may help, e. g., having separate triple stores per project.

A.2.5 Denial of Service

Denial of Service (DoS) describes an attack pattern in which the functionality of systems or components, such as network resources or servers, is disrupted in such a way, that it is unavailable or unresponsive for significant amounts of time. An often used type of such attacks is the Distributed Denial of Service (DDoS) attack, which uses multiple attack sources to flood systems which overwhelming amounts of traffic. DDoS attacks are of a lesser relevance in our context, as we do not rely on publicly reachable systems, in contrast to dgFOAF which requires publicly available FOAF files on webservers. However, it is possible that an malicious actor would like to disrupt the availability of the components required for our solution. A plausible attack method would be a massive amount of validation requests, e. g., for group membership or legitimate privilege assignment. We address this issue, by only allowing authenticated agents to query such requests. A more generic DoS attack approach is SYN flooding, i. e., trying to distrub the operation of a server by opening huge amounts of connections by sending a `SYN` package without responding with `ACK` to complete the TCP three-way handshake. As this attack is not specific to semDIM, it is to be addressed with well-known countermeasures as listed in RFC 4987[4], e. g., network filtering, reduced `SYN-RECEIVED` timer, and recycling half-open TCP connections oldest first. In any case, DoS or DDoS attacks are not really an issue in protected environments, such as company or project networks, as malicious agents can be identified quite rapidly and, in the case of DDoS attacks, the number of potential attack sources is very limited.

A.2.6 Elevation of Privilege

Elevation of Privilege happens, if an actor can reach a higher level of privileges he is not authorized for. Most often such attacks are associated with exploiting weaknesses on system level, usually in the operating system or applications. As this attack depends on the robustness of the implications it is predominantly a technical issue and not further analyzed in this paper. A different elevation of privilege approach is trying to illegitimately assign group membership or other

[4] `https://tools.ietf.org/html/rfc4987`, last accessed 2020/12/20.

roles to reach higher privileges. This is explicitly prevented by semDIM's reasoning, controlling each assignment for legitimacy.

In summary, we analyzed the security implications of our proposed semDIM framework regarding all relevant threats adapted from the STRIDE model. Many threats exist independently from semDIM as try to exploit underlying infrastructural weaknesses, e. g., tampering data on OS or database level. Although these infrastructural weaknesses need to be addressed on a lower, more technical level, i. e., hardening OS and network components, semDIM's security mechanisms still offers increased protection here, e. g., detecting inconsistencies by the Two-Phase-Commit message exchange in case of successful tampering. As the analysis shows, a networks security level is generally increased by semDIM. Processes specifically introduced by semDIM, i. e., creating, exchanging, and processing graph-based representations of IM information, are sufficiently protected by reusing State-Of-The-Art security measurements, e. g., Two-Way-TLS and client authentication, as well as applying specifically for graph-based information exchange a certificate-based graph signing and validation framework.

References

1. Gangemi, A.: DOLCE+DnS Ultralite (DUL). ontologydesignpatterns.org (2009), `http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite`
2. ISO/IEC: ISO/IEC 27001:2013 Information technology — Security techniques — Information security management systems. ISO/IEC (2013)
3. Kasten, A., Scherp, A., Schauß, P.: A framework for iterative signing of graph data on the web. In: ESWC 2014 Proceedings. Springer (2014)
4. Kasten, A.: Secure Semantic Web Data Management: Confidentiality, Integrity, and Compliant Availability in Open and Distributed Networks. University Koblenz-Landau (2016)
5. Microsoft: The stride threat model (2009), `https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN`
6. NIST: NIST SP 800-53, Security and Privacy Controls for Federal Information Systems and Organizations. NIST (2014)
7. Scherp, A., Saathoff, C., Franz, T., Staab, S.: Designing core ontologies. IOS Press, Applied Ontology 6 (2011)