Detection of Semantic Conflicts in Ontology and Rule-Based Information Systems

Jose M. Alcaraz Calero, Juan M. Marín Pérez, Jorge Bernal Bernabé, Felix J. García Clemente, Gregorio Martínez Pérez, Antonio F. Gómez Skarmeta

Department of Information and Communications Engineering
University of Murcia
Facultad de Informatica, Campus de Espinardo, s/n
30071 Murcia, Spain

Abstract

Nowadays, managers of information systems use ontologies and rules as a powerful tool to express the desired behaviour for the system. However, the use of rules may lead to conflicting situations where the antecedent of two or more rules is fulfilled, but their consequent is indicating contradictory facts or actions. These conflicts can be categorised in two different groups, modality and semantic conflicts, depending on whether the inconsistency is owing to the rule language expressiveness or due to the nature of the actions. While there exist certain proposals to detect and solve modality conflicts, the problem becomes more complex with semantic ones. Additionally, current techniques to detect semantic conflicts are usually not considering the use of standard information models. This paper provides a taxonomy of semantic conflicts, analyses the main features of each of them and provides an OWL/SWRL modelling for certain realistic scenarios related with information systems. It also describes different conflict detection techniques that can be applied to semantic conflicts and their pros and cons. Finally, this paper provides a comparison of these techniques based on performance measurements taken in a realistic scenario and suggests a better approach. This approach is then used in other scenarios related with information systems and where different types of semantic conflicts may appear.

Keywords: Semantic Conflicts, Conflict Detection, Semantic Rules, Knowledge Representation, Ontologies

1. Introduction

Knowledge models are used to represent conceptualizations of information systems and application domains. Some proposals even foster the automated generation of such domains [1]. These models can define services, networks, applications and any entity which is to be represented. These representations are done by means of diagrams and models which represent concepts of real entities. Several languages like UML [2], MOF [3], XMI [4], RDF [5] and methodologies such as WebML [6] and RUP [7] are available for describing these knowledge models.

The knowledge modelling process should be able to represent both static and dynamic aspects of the information system. While the former is related to the structure, composition, and description of the system, the latter is related to the behaviour, events, actions and the state of the system. In order to define the behaviours of the modelled system in a formal way, some languages and methodologies are supported by other formal languages that claim to model system specifications. Thus, OCL [8] and Z [9] languages can model system behaviours and requirements on UML and MOF models. Similarly, ISO Schematron [10] can be applied to XMI, whereas OWL [11] and SWRL [12] can model behaviour specifications on RDF models.
Thanks to the formal nature of these languages, several automated processes become available [13]. The usage of ontology models like OWL, enable to perform tasks like checking constraints, simulating actions or inferring new knowledge. Thus, they can directly act on the knowledge model of the system by discovering, simulating and checking how new knowledge, events or actions could affect it.

Detecting conflictive situations can be considered a reasoning process that could be executed on the knowledge model in order to find some situations which were not initially allowed in the system. This process enables the detection of several types of anomalies in the modelled system. For example, conflict detection can discover security failures, undesired behaviours, configuration mistakes and contradictions, among others. This is a valuable process for systems where the avoidance of contradictory, forbidden or inconsistent situations should be guaranteed.

Several successful attempts for developing conflict detection processes are provided in [14], [15], [16], [17], [18] and [19]. All of them share a common basis. They provide a set of conflict detection types according to definitions of conflictive situations. The knowledge model is inserted in a conflict detection process which performs an analysis taking into account some conflict definitions. As a result of a conflict detection process three kinds of models can be obtained: conflict-free models, conflict-detected models and conflict-undetected models. The first ones represent situations where no conflict is present, the second ones represent situations containing conflicts which are detected and the third ones represent situations containing undetected conflicts. Therefore, given that conflicts may be detected or not, depending on the chosen conflict detection method, a suitable definition and selection of the detection method is the key to perform a quality conflict recognition.

According to Lupu et al [17] and Moffett et al [20], conflicts related to the management of information systems can be categorized in two different types: modality conflict and conflict of goals. The first one is a conflict that occurs because of situations which are inherently incompatible. For instance, let us suppose a subject that is authorized to perform an action on a given target and, at the same time, the subject is not authorized to perform that action on the same target. In this case, a modality conflict arises because positive and negative authorizations are granted. On the other hand, a conflict of goals occurs because the actions are incompatible or they violate some desired properties in the application domain. For example, let us suppose a scenario in which the same manager cannot authorize payments and sign the payment checks. In that case, the conflict arises if payment and sign actions are performed by the same subject on the same check. Conflict of goals have been lately referred as semantic conflict on research works as [14], [21] or [22]. So, this kind of conflict will be indistinctly referred as conflict of goals or semantic conflict henceforth.

Likewise, while modality conflicts have been studied in several research works like [17], [16], [23], [24], [25], [26], [27] and [28], there are less works providing detection methods for semantic conflicts, such as such as [14], [29], [19], [17] and [30], since this kind of conflict detection is dependent of the modelled domain.

This research work is focused on semantic conflict detection. Semantic conflicts depend on the application domain and they require working with the meaning of concepts. This fact makes their detection more complex than modality conflicts, which can be directly detected by analysing the knowledge model, independently of the modelled application domain. Moreover, research works on conflict detection usually lack of performance evaluation of the proposed techniques. In this work, different kinds of conflict detection methods are proposed and analysed, probing the feasibility, scalability and performance of each method when dealing with realistic scenarios of information systems management.

Consequently, this contribution tries to address the challenge of defining a conflict detection process applied to realistic sample scenarios. Next sections will give a deeper insight on some aspects of the overall solution as follows: Section 2 provides a background of different conflict classifications in order to contextualize and expose our classification. A justification of the selected knowledge model to represent information systems is provided in Section 3. Section 4 introduces the model that will be used as general domain to perform the semantic conflict detection. Exemplifications of the main semantic conflicts available on information system scenarios and the way in which these conflicts can be detected are explained in Section 5. Section 6 delves into different ways of carrying out the semantic conflict detection. Some statistical and performance results of these different conflict detection methods are exposed in Section 7. Finally, conclusion and future work are drawn in Section 8.

2. Taxonomy of Management Conflicts

According to The American Heritage Dictionary of the English Language [31], a conflict is defined as “A part of discord caused by the actual or perceived opposition of needs, values and interests...”. Several conflict taxonomies
have appeared for the last 20 years. Indeed, many kinds of conflict, from emotional to economic, from idiomatic to ideological, from military to organizational, from management to racial, have appeared attending to different grouping criteria. Since this paper focuses on information systems management, this section proposes a conflict taxonomy attending to the different kinds of conflicts which may appear in this context.

Focusing on system management, there are several research works providing conflict taxonomies. All of them offer different conflict types, extending, refuting or modifying other existing classifications. A full classification of conflicts which enables to group and unify all of them together could contribute to an homogenization of available conflict hierarchies. Figure 1 shows the proposed conflict taxonomy, which results from exploring different conflict taxonomies based on system management, with the aim of providing a complete classification including most of them.

The first relevant work on conflict classification is provided by Mofferr et al [20]. This taxonomy has been taken as starting point in the definition of the proposed one and it has been expanded and extended with the different contributions provided by different authors. Mofferr et al [20] promote two different types of conflict: modality conflict and conflict of goals. The first one occurs when two incompatible statements are defined together. These incompatible statements can be detected without taking into account the underline application domain, i.e., they are application-independent conflicts; therefore, conflicts of this type are independent of the information system modelled. In turn, two different types of modality conflict have been exposed: positive/negative conflicts, which arise when both affirmative and negative statements are present on the knowledge model at the same time; and imperative/authority conflicts, which arise when a subject is obligated to do something which he/she is not authorized to do.

Likewise, a conflict of goals occurs when two actions are incompatible in the context of the modelled information system. Therefore, to detect these kinds of conflict, the modelled information system needs to be taken into account, i.e., they are application-dependent conflicts. Indeed, these conflicts arise due to the nature of the performed actions and depend upon their semantics. So, these conflicts appear as semantic conflicts in our taxonomy. Four different subtypes of this kind of conflicts have also been exposed in the same proposal. Firstly, a conflict of priorities, which occurs when concurrent access is attempted on a single resource. Secondly, a conflict of authority, which occurs when the same subject is able to perform semantically incompatible management tasks. For example, an account supervisor that enters payment information and, in turn, he signs payment checks. Thirdly, a conflict of multiple-managers, which occurs when different managers are performing incompatible actions on the same target. And finally, a conflict of self-management, which arises when a subject is able to manage his own capabilities in the system. In turn, the conflict of authority is divided in two different subtypes of conflicts: conflict of interest and conflict of duties. While the former occurs when a subject is able to perform actions which may conflict with his own interest, the latter occurs when a subject is obliged to perform incompatible actions. This conflict taxonomy of Mofferr et al [20] has been
considered as the foundation of many other conflict classifications such as the provided by Lupu et al [17], Sloman et al [21] and Giorgini et al [29].

As far as they are concerned, Lupu et al [17] and Sloman et al [21] extended the positive/negative conflicts introduced by Mofferr et al [20] in two different types: positive/negative authorization conflicts and positive/negative obligation conflicts, depending on the nature of the statements. Thus, in authorization ones the statements define whether a subject can or cannot perform activities on a set of targets, while in obligation ones they define whether a manager or agent must or must not perform activities on a set of target objects.

Additionally, Giorgini et al [29] provides an alternative classification for the conflict of interests, providing three different subtypes of this kind of conflict: role conflict, where a subject is assigned to a role whose interests collide with those of the subject; self-monitoring conflict, where an actor is responsible for monitoring his own behaviour; and attorney-in-fact conflict, where some - possibly personal - interests of the delegate interfere with the interests of the delegator.

In our proposal, it is considered that role conflict provided by Giorgini et al [29] matches with the previous conflict of interests or conflict of multiple-managers provided by Mofferr et al [20], depending on whether the role and the subject belong to the same administrative domain or not, respectively. Analogously, self-monitoring conflict matches with the self-management conflict and attorney-in-fact conflict is a new type of conflict caused by delegation capabilities.

Moreover, Jajodia et al [32] provides a completely different taxonomy of conflicts. This classification defines two different subtypes for the conflict of duties in the context of an authorization scenario: static conflict of duties and dynamic conflict of duties. The former occurs when it is specified that certain sets of actions are not allowed for the same subject, whereas the later occurs when the execution of a given action inhabilitates the subject to perform some other actions. The same proposal also includes the conflict of incompatible role assignment, which arises when two incompatible roles are assigned to the same subject; and the conflict of incompatible role activations, which occurs when the activation of a given role invalidates the activation of another one. Notice that the main cause for which two roles could be incompatible is due to a conflict of interest on the role assignment or activation, for this reason they match with the conflict of interest provided by Mofferr et al [20]. Brewer et al [33] provide the Chinese Wall conflict as a special type of dynamic conflict of duties. Chinese Wall arises when a subject belongs to two competitive companies and he is attempting to perform an action on behalf of one of them. After that, he can no longer perform actions on behalf of the other company.

In our proposal, these two subtypes of the conflict of duties defined by Jajodia et al [32] have been included, whereas the Chinese Wall conflict is considered under the definition of the role conflict exposed by Giorgini et al [29]. Similarly, incompatible role assignment and incompatible role activation exposed by Jajodia et al [32] are considered under the conflict of interests definition due to their analogy with this kind of conflict.

From a different perspective, Kamoda et al [34] provides the following conflict types: modality conflict, which matches with the proposal in Lupu et al [17]; Chinese Wall conflict and conflict of duties, which matches with Brewer et al [33] and Mofferr et al [20] proposals, respectively; propagation conflict, which arises when positive or negative permissions are propagated from roles to roles or users and this propagation is conflictive; and conflict of action composition, which happens when complex actions can be performed in the information system and these complex actions are composed of elementary ones which are conflictive.

The conflict of propagation of Kamoda et al [34] can be seen as a special case of the positive/negative modality conflict of Mofferr et al [20]. So, this kind of conflict does not really provide a significant difference apart from the nature of the cause that originates the conflict. Analogously, the conflict of action composition is a special case of semantic conflict that requires composed actions modelled on the system, but it does not entail a new kind of conflict.

Another conflict taxonomy is provided by Syukur et al [35], in which three different types of conflicts are defined: modality conflict, which matches with the definition of Mofferr et al [20]; role conflict, that matches with the conflict of roles provided by Giorgini et al [29]; and conflict of entities that matches with the conflict of priorities provided by Mofferr et al [20].

On another side, Hall-May et al [22] provides another conflict taxonomy, exposing the following conflict types: positive/negative conflict, permission/obligation conflict, deductive refinement conflict, multiple-manager conflict, self-management conflict, conflict of interests, conflict of duties, dependence conflict and conflict of resources. While almost all of them match with the different kinds of conflict previously defined by other authors, it is worth to focus on the dependence conflict, the conflict of resources and the deductive refinement conflict. The conflict of resources
definition overlaps with the conflict of priorities provided by Mofferr et al [20]. Moreover, although the deductive refinement conflict is not considered as a conflict by the author, this kind of conflict remarks the importance to realize that policies may not always be in conflict at the same level of specification. Finally, the conflict of dependence arises when there exists a dependency between two policies, either temporal or spatial. Temporal and spatial dependencies are related to the time and physical state, respectively.

Recently, Chadha [25] has also provided a conflict taxonomy based on the original study of Lupu et al [17]. This author splits the conflicts in application-independent conflicts and application-dependent conflicts. The first type is in turn composed of modality conflict - following the Lupu et al [17] definition - and redundancy conflict. This last conflict arises when there are two or more identical policies on the system. On the other hand, application-dependent conflicts are divided on mutex conflict, redundancy conflict, inconsistency configuration conflict and other conflict types in which the conflict of duties and the conflict of interests are included. The mutex conflict arises when two incompatible policies are fulfilled at same time, whereas in the redundancy conflict redundant policies are detected when a policy subsumes all the possible cases of another policy, according to the application domain. The inconsistency configuration conflict is defined as a special type of mutex conflict with semantic connotation on conflict detection. For example, let us suppose an action which increases the bandwidth on a network device and another one that establishes the maximum bandwidth allowed to a given value. This conflict may arise when increasing the bandwidth implies exceeding the maximum bandwidth allowed.

Apart from inconsistency configuration conflict, all the previous conflict types are taken into account in our proposal. The inconsistency configuration has been left aside due to its overlapping with the conflict of priorities provided by Mofferr et al [20].

As can be observed, different authors propose different conflict taxonomies. Some of them share common basis while others do not. So that, a new taxonomy has been provided in Figure 1. This taxonomy has the intention of unifying the previous proposals in a single taxonomy for conflict classification. Most of the authors only expose different conflict types, but they do not provide a hierarchical organization. In this regard, a hierarchical structure has been proposed in order to classify the different types and subtypes of conflicts.

3. Knowledge Representation in Information Systems

Several attempts to design conflict detection processes for information systems management have been successfully proposed up to now. Conflict detection systems like Kaos [15], REI [16] and Ponder [14] are some successful proposals. On the other hand, regarding conflict detection algorithms, some examples like Giorgini et al [29], Kamoda et al [19], Kanno et al [18] and Chomicki et al [30] are suitable options, but almost all of them share the same lack: they do not use a standardized and implementable way to model the managed system, i.e. the information system. In this sense, almost all of them expose languages, processes, conflict detection algorithms and techniques based on their own application domain, in which these techniques and algorithms are applied for conflict detection. However, these domain models are usually customized and adapted for conflict detection purposes and difficult to implement in realistic scenarios, making ontological knowledge hard to reuse, which is one of the added values of the Semantic Web [36]. For instance, Kamoda et al [19] describes the knowledge needed to allow their conflict detection processes but they do not expose how this knowledge should be modelled. Chomicki et al [30] explains conflict detection methods based on an application domain which is not described.

In this proposal a standardized method to model information systems has been selected in order to perform the conflict detection process. By adopting this approach, a complete and well-known model to define information systems is available as underline knowledge base, instead of a partial and customized model for specific purposes. This approach makes the detection techniques applicable to real information systems, which can be represented by these standardized and generic models.

Regarding knowledge models to define information systems, several proposals are available. OIM [37] is a shared information model intended to support tool interoperability across technologies and companies encompassing all phases of information systems development, from analysis to deployment. Similarly, CWM [38] is an information model promoted by OMG to create a standardized object-oriented architectural framework for distributed applications in order to support reusability, portability and interoperability of object-oriented software components in heterogeneous environments. In the case the reader is interested in a comparison between OIM and CWM, Vetterli et al [39]
provides a detailed comparison between them. Another approach to model information systems is the SID model provided by The TM Forums Information Framework [40], embodying an extensive, flexible, and widely proven model that defines business concepts, their characteristics and relationships, all described in an implementation independent way. Additionally, CIM [41] created by the DMTF provides a common definition of management information for systems, networks, applications and services designed also for vendor extensions. CIM’s common definitions allow to describe information systems in an implementation independent way, enabling vendors to exchange semantically rich management information between systems throughout the network.

From all these information models, CIM has been selected as the most suitable alternative to perform conflict detection processes. The main reasons that have determined this election are the following ones: i) CIM is a very complete information model, as it covers almost all the different aspects required in a networking scenario, including systems, services, networks, applications, policies, security, etc. ii) There are related standards and technologies grouped under the WBEM [42] specifications which allow to dynamically gather the current state of the system by means of CIM; iii) CIM is independent of the language used to represent it; and iv) CIM is free, open source and extensible, ready for new information systems. Additionally, this information model has been used in a wide variety of research works such as [43], [44] and [45], among others.

CIM provides a consistent information model with well-defined associations which capture management content for IT domains. It is composed of a Core Model, some Common Models and Extension Schemas. The Core Model captures basic notions which are applicable to any area of management, while the Common Models capture notions which are common to particular management areas, but still independent of any particular technology or implementation. Finally, the Extension Schemas represent technology-specific extensions of the common model.

CIM employs modelling techniques from the object oriented paradigm to provide a common and extensible definition of management-related information. The descriptions of the model are provided by the CIM meta-model, which defines the terms used to express the models with its usage and semantics. Among its main elements is the term Class, which defines the properties and the methods common to a particular kind of object; and Schema, which is a group of classes with a single owner. The model also supports Indication which is the representation of the occurrence of an event, Association which represents a relationship between two or more classes using references, and Qualifier which provides additional information about the element to which it applies. Additionally, it is worth mentioning that this model defines Association as a subclass of Class. This allows two previously unrelated classes to be associated without modifying them, by creating a new Association class with references to the classes which are to be associated. Association classes are the only ones that may contain references, being a reference simply a pointer to an object instance.

CIM can be represented on different languages, from XML to RDF, from MOF to OWL, from XMI to UML. CIM is natively provided in MOF, UML and XML representations. In this sense, several research works have focused their efforts on converting CIM to formal languages such as RDF [46] and OWL [45], [47]. By adopting a formal representation of CIM, several reasoning processes become available in order to check, validate, simulate or detect conflicts in information systems. Using OWL as language to represent CIM, the semantic of the Description Logic (DL) [48] can be applied in the reasoning processes. DL is closely related to Semantic Web [49] technologies, in which information is constantly being discovered, modified and updated.

Furthermore, different OWL sublanguages are available attending to the expressiveness and computational complexity provided, such us OWL-Lite, OWL-DL and OWL-Full. Recently, OWL 2 [50] has been released as the most expressive OWL language that remains using a decidable subset of first order logic. Consequently, OWL 2 is the language selected to model CIM in our proposal, since it is the most suitable alternative according to expressiveness and complexity. Knowledge models modeled in OWL 2 can be enriched by means of SWRL [12]. This language is used to represent rules on the Semantic Web and it provides an expressiveness extension to OWL languages. SWRL allows the description of rules for the application domain. Therefore, SWRL combined with OWL 2 can be used to express more deductive processes on the modelled system. For instance, this combination allows the definition of rules with negative atoms, custom relationships between properties and the definition of actions when a given search criterion is fulfilled. Thus, both OWL 2 and SWRL are the selected languages to carry out the semantic conflict detection processes exposed in the following sections.

Regarding the mapping of CIM to OWL, several representations of CIM in OWL are available like the provided by Quirolgico et al [46], Majewska et al [45] and Garcia et al [47]. In this proposal, the chosen CIM-to-OWL mapping has been the one proposed by Majewska et al [45]. The reason for this choice is twofold. On one hand, it holds a
high expressiveness in the translation to OWL in contrast to the approach provided by Quirolgico et al [46], which only preserves RDF expressiveness. The second reason is related to the way in which the relationships between CIM elements have been mapped. In this sense, while Garcia et al [47] proposes the mapping of CIM relationships to OWL classes, Majewska et al [45] defends to map CIM relationships to OWL properties. This last approach allows to model semantic features applied to these properties and confers simplicity to the OWL CIM representation. For instance, in this approach just one OWL ObjectProperty is needed to model a CIM association, whereas in Garcia et al [47] one OWL Class plus two ObjectProperties are required to model the same association. Hence, this approach makes it possible to reduce the work load required by the reasoner to process and manage the ontology.

Finally, it is worth mentioning that knowledge bases based on DL ontologies are divided into TBox (terminological) and ABox (assertional) components. The former contains the schemes and vocabularies that define domain concepts, their properties and the relationships among them. The latter is populated with instances of these concepts and relationships, representing a specific situation in the domain, according to those schemes. In this context, an individual is defined as an entity which is modelled in the application domain by means of instances, while an instance is defined as an association which states that an individual belongs to a given class.

4. Domain Model

This section introduces the domain model used in this paper to perform the conflict detection process. The model, as exposed in the previous section, uses CIM represented in OWL as ontology, and SWRL to express the rules which define the behavior of the system. It will allow the modelling of conflictive scenarios which may appear on real information systems. Concretely, CIM version 2.20 has been used as base model and it has been extended to include the classes which represent the concepts needed to model several scenarios that will be used to illustrate different kinds of semantic conflicts in information systems management.

![Figure 2: General domain model](image)

Figure 2 shows a conceptual UML representation of the most relevant classes of this general domain model. It should be noticed that not the whole model is represented in the figure since CIM contains a big set of classes and associations. This figure intends to give an overview of the model from the conflict detection point of view.

As stated in Section 1, semantic conflicts occur when incompatible actions are requested to be executed in the system. Hence, the domain model contains an Action class which represents those actions in order to effectively detect these kinds of conflicts. The inclusion of the Action class and its hierarchy of classes is the main extension done to CIM. Action is a generic concept able to represent a wide range of activities or processes. Sticking to information systems, actions are usually performed by means of executing services, which are the logical representation of processes running on a system. The ExecuteServiceAction class represents actions which are performed by executing services. Thus, when a software agent intends to act with the system, it requests an action specifying the service it wants to be executed via the ServiceToExecute association and the parameters the action may take using the ServiceActionParameter one. Those parameters represent information relevant to the execution of the service in order to get the desired behavior in the system.

On the other hand, actions can be generated by an external entity or user requesting the system to change its status. Such entities or users should be authenticated before allowing them to execute actions. The authentication process...
associates an identity to the user, allowing its identification in the scope of the system; it is represented by the Identity class in the model. But not only external users interact with the system. Running software can also produce actions, behaving as a software agent which may change the status of the system. This running software or process is logically represented by a service, which is also associated with an identity using the AssignedIdentity association, unifying in this way, by means of identities, the subjects - human or software - which are able to perform actions in the system.

Those actions may be performed directly by subjects or they can define rules which, if conditions are fulfilled, specify the actions on its consequent or they may even change the status of the system by directly modifying the model. Hence, rules are also modelled with the Imp class. But knowing the rule which executed an action is not enough. Some kinds of semantic conflicts depend on the subject which performed the changes to the system. So, if it is a rule which performs those changes, knowing the subject which defined the rule is also necessary and it is represented in the model using the RuleDefinedByIdentity association.

Since CIM is used as knowledge model for system representation, most of the classes and associations used in this model come from CIM or are extensions to it. Although not depicted in the figure for clarity reasons, every class in the diagram inherits from ManagedElement. This class is the root class of CIM hierarchy and serves as link with the rest of the model, which is neither depicted in the figure since only the relevant classes are shown. The ManagedElement class, together with the Identity and Service classes, as well as the AssignedIdentity property are defined by CIM. The action hierarchy is an extension to this model in order to represent the activities since CIM is intended to model a static view of the system and it does not provide a representation of such concepts. This hierarchy includes the Action and ExecuteServiceAction classes with their properties, namely ActionRequestedByIdentity, ServiceToExecute and ServiceActionParameter. Rules are modelled using the SWRL Ontology [12], to which the Imp class belongs. The rest of classes of this ontology are not shown in the figure like occurs with the rest of CIM classes. The RuleDefinedByIdentity and RuleAffectsIndividual properties have been defined and they link this ontology with CIM. By linking these two ontologies, a unified knowledge base is obtained from two different information sources. The first one is modelled using CIM and it represents the status of the system as well as the actions to be executed. The second one represents the rules which govern the behavior of the system and they are modelled using SWRL Ontology.

Two different kinds of rules can be distinguished in this proposal. Although both are represented in SWRL, they are defined for different purposes. On one hand, there are rules which belong to the system description and that specify how the system should act against certain situations. For example, this is the case of rules created by an administrator to define an authorization policy for the system. On the other hand, there are the rules which are defined for conflict detection purposes, following the approach exposed along this paper. An example of the first kind of rules can be the following authorization rule, which assigns every privilege which administrators have over a document service to a recently created role for document managers:

\[
\begin{align*}
\text{DocumentManagerPrivileges} & : \text{Role}(\text{a}) \land \text{name}(\text{a}, \text{Admin}) \land \text{Role}(\text{dm}) \land \text{name}(\text{dm}, \text{DocumentManager}) \land \text{Service}(\text{s}) \land \text{name}(\text{s}, \text{DocumentService}) \\
\text{AuthorizedPrivilege}(\text{ap}) \land \text{authorizedSubject}(\text{ap}, \text{a}) \land \text{authorizedTarget}(\text{ap}, \text{s}) \land \text{authorizedSubject}(\text{ap}, \text{dm})
\end{align*}
\]

The first three lines in DocumentManagerPrivileges rule defines the two roles and the service. Line 4 determines the privileges which are currently assigned to administrators for the document service. Finally, the consequent establishes those privileges also for the role of document managers. The classes and associations which appear in this example are used to model authorization in CIM and therefore they are also available in our domain model.

Regarding the second kind of rules, several examples can be found in Section 5, where different rules of this kind are defined for the various types of semantic conflicts.

As stated before, this section has introduced only some concepts of the domain model to contextualize semantic conflict detection. Further sections will show more parts of the model, including more specific classes and associations needed to illustrate different kinds of semantic conflicts. All these new classes represent system elements and they are part of CIM. Therefore, they inherit from ManagedElement or any of the classes presented here, although not
explicitly depicted in figures for clarity reasons.

It should be also noticed that some concepts modelled here, like representation of subjects by means of identities or processes by means of services are just simplifications to make further examples and explanations clearer and easier to understand. However, actual systems may present complex mechanisms to identify users, software hierarchies, service compositions or complicated concepts more difficult to model that the ones exposed here. The usage of CIM as information model covers a wide set of concepts that allows the representation of a whole complex information system. This model, together with the power of a semantic reasoning engine, makes the conflict detection process presented in this paper applicable to real information systems.

5. Types of Semantic Conflicts

The main aim of this section is to provide a set of scenarios based on information systems in which the main semantic conflicts related to such systems appear. Detection rules are also provided to detect the semantic conflicts illustrated in these scenarios.

Previous research works such as Lupu et al [17], Giorgini et al [29], Chomicki et al [30] and Kamoda et al [19] provide techniques and algorithms to detect some types of semantic conflicts. However, almost all of them are based on a non-standardized information model. This fact remarks the lack of conflict detection techniques which can be directly applied to the modelled application domain.

Focusing on semantic conflicts, the proposed taxonomy contains: redundancy conflict, conflict of priorities, conflict of authority, multiple-managers conflict and self-management conflict.

The conflict of authority is divided into the conflict of duties and the conflict of interests. In turn, these two types have different conflict subtypes in our taxonomy. However, in this section the detection will be performed at this level since the detection of all the subtypes of these conflicts is performed by the same algorithms. The main difference resides in the semantic which causes the conflict. For instance, static and dynamic conflict of interest differ on the reason why these conflicts arise, but the underling meaning of the conflict is exactly the same.

Regarding the conflict of priorities, since this kind of conflict does not entail any kind of complexity, it will not be treated in this section. The conflict of priorities arises when concurrent access is attempted against a limited or single resource. Due to the nature of this conflict and the underling formal language, just including a simple cardinality restriction on the resource, when more than one simultaneous access is carried out, an inconsistency on the knowledge base is produced, causing the conflict detection.

Considering the semantic conflict of redundancy, it is detected by a formal verification of the available policies on the system, leading to the elimination of redundant policies. So, an algorithm to perform the redundancy detection should be provided. Several successful attempts to provide an algorithm for this purpose are exposed by Schmolze et al [51], Racine et al [52] and Chavarria-Báez et al [53]. Due to the number of works dealing successfully with this type of conflict, it will not be covered as part of this proposal. Apart form these exceptional cases, the rest of semantic conflict types will be analysed in this section.

The architecture to detect semantic conflicts on information systems will take as input the knowledge model previously introduced in Section 3 and the definitions of the different kinds of conflicts provided in the rest of this section. While the first input is described by means of CIM models represented in OWL, the second one is defined by means of semantic rules modelled in SWRL. Both knowledge model and rules will be inserted on a DL reasoner in order to perform inference processes. Since OWL is based on the Description Logic and it needs a DL reasoner to infer new knowledge, the conflict detection methods proposed in this paper will also make use of this kind of reasoners. The reasoner will perform the semantic enrichment of the knowledge model by inferring new knowledge in the application domain. Thus, since SWRL rules defining conflicts are available in the reasoning process, they will detect the semantic conflicts by inferring inconsistent facts when conflictive situations arise in the information system.

Notice that in the following subsections, the consequent part of the rules used to detect the semantic conflicts are not totally specified. These rules just define that the conflict is detected in a conceptual way, labelling it on its consequent with the word “Conflict” in inverted commas. Particular consequents of conflict detection rules will be lately specified in Section 6, where different ways of generating conflicts will be analyzed.
5.1. Conflict of Interests

A conflict of interests can be defined as a situation in which an individual or corporation is in a position to exploit a professional or official capacity in some way for their personal or corporate benefit. This kind of conflict can be illustrated with a Public Key Infrastructure (PKI) scenario in which a user performs a certificate request and then he/she is authenticated by the Registration Authority (RA) that gives the approval for the CA to issue the certificate.

This process guarantees the trust and validity of the certificates whenever the requester and RA roles are effectively detached. Suppose the same user is able to both request a certificate and validate the identity of the requester. Such a user could potentially request certificates for invalid or false identities and validate them, making the system to issue invalid public key bindings and compromising the trust of the system. This example illustrates a semantic conflict of interests in which the user may act on its own benefit by issuing invalid certificates which could then be used for illegitimate actions.

![Figure 3: Conflict of interests scenario domain](image)

Figure 3 extends the model exposed in Section 4, showing the classes needed to represent this scenario. The model includes the `CredentialManagementAction` class which represents actions that can be done against credentials, like a certificate request and its validation once the identity has been verified. The `CredentialManagementService` class represents a service which is able to manage credentials. But the two actions which take place in this scenario are not concurrent. The user requests the certificate in a time slot previous to the identity validation by the RA. Since both actions should be taken into account for conflict detection, the detection can only be done when the second action - approval of identity - is requested, but in that moment the first action - certificate request - is no more available since it was already finished. To solve this issue, the needed data of the first action can be retrieved from the system log. To simplify the CIM representation of logs, a subclass of `LogRecord` called `ExecuteActionLogRecord` has been defined to represent exactly the same information as the original action, but once finished. Thus, this class is also associated to the service, parameters and identity involved in that action, following the same approach used for the `ExecuteServiceAction`. This approach is a convenience simplification to achieve a more readable scenario, but it could be applied to a full CIM log representation by making the appropriate changes to this part of the model and the affected rules.

Both actions and log records - which in the end represent already executed actions - should be associated with the identities which requested them. In this scenario, the PKI is a corporate one and only company employees are allowed to request certificates. The requester should be authenticated in the system as employee to request a certificate and the RA responsible should also be authenticated to validate the certificate request once the requester identity has been verified. Thus, the system authentication mechanism assigns an identity to the requester and another one to the RA.
Figure 4 shows an UML object diagram depicting a scenario in which a conflict of interests is present to better understand the rule which detects this kind of conflict.

![UML object diagram](image)

**Figure 4: Conflict of interests scenario**

In this example, Alice is the RA responsible and she has requested a certificate on behalf of Bob. Since the certificate request action has already been done, it is represented by a log record. This log record is associated with the corresponding identity, certificate and credential management service, meaning that Alice has requested Bob’s certificate using the PKI service. The request can be seen as an “unsigned” certificate and it has been represented like that in this example for clarity reasons. Then, as RA responsible, Alice validates Bob’s certificate request, proceeding with the issuance of the signed certificate. The example shows a conflict of interests in which Alice is using her RA privileges on her own interest to issue a certificate on behalf of a different person which can be used to impersonate Bob with respect to the system. To detect this kind of conflict, the following rule is defined:

\[
\text{ConflictOfInterests : Identity(})i\text{) }\wedge \text{Credential(})c\text{) }\wedge \\
\text{RequestCredentialLogRecord(})req\text{) }\wedge \text{logRecordByIdentity(})req, i\text{) }\wedge \text{usedCredential(})req, c\text{) }\wedge \\
\text{ValidateCredentialAction(})validate\text{) }\wedge \text{actionRequestedByIdentity(})validate, i\text{) }\wedge \text{actionOnCredential(})validate, c\text{) }\wedge \\
\text{CredentialManagementService(})s\text{) }\wedge \text{credentialManagementServiceToExecute(})validate, s\text{) }\rightarrow \\
“\text{Conflict}”
\]

(1) (2) (3) (4) (5) (6)

Since the action specified in the log record in line 2 and the action in line 3 are a request and a validation performed by the same identity on the same credential, then there is a conflict of interests in which the identity is trying to validate a certificate which has been requested by itself. When all these conditions are fulfilled, the conflict is raised and could be detected by any of the methods which will be exposed in the Section 6. It should be noticed that this conflict detection process would be analogously applicable to any situation which may present a conflict of interests in the managed system.

The conflict of interests exposed in this section belongs to the dynamic subtype. Different subtypes of this kind of conflict can be detected in the same way. For instance, in some cases, the fact of having an identity which belongs to two different roles may also produce a static conflict of interests and they should be detected even if the identity have not yet made use of its privileges on the system. The conflict detection process can also be applied to this situation by adapting the detection rule. Similarly, the attorney-in-fact conflict of interests which is produced by privilege delegation, can also be detected following this approach.

### 5.2. Self-management conflict

A self-management conflict arises in scenarios where a subject has the chance to manage the rules which govern his capabilities in the system. It may arise when a subject is able to change its own privileges in the system, being able to use this on his own benefit to perform actions which where initially forbidden for him. Or it may also arise if a subject is able to change the definition of its duties on the system, what can be also used on his own benefit. Usually, such subjects belong to special roles such as system administrators or department managers. It should be noticed that, while the other kinds of conflict exposed in this section are based on the execution of actions, this one is based on changes of the rules which define the security policies of the system.
A possible scenario to illustrate this kind of conflict can focus on authorization. Suppose a system where authorization is managed by the administrator by specifying rules which define privileges. Figure 5 represents the model exposed in Section 4, showing the needed classes for this authorization scenario.

The model includes the **AuthorizedPrivilege** class which is a CIM class representing a privilege. To detect the self-management conflict, the rule which modifies privileges and the subject which defined such a rule should also be known. This is achieved by the **Imp** class and its associations.

Given this domain, a self-management conflict can raise when an identity tries to change its own privileges like in the situation shown in Figure 6.

In this example, Alice is a sales manager and she is trying to access a database, but she has no privileges to do it. Hence, as she is able to change privileges on the system, she defines a rule which creates a DBAccess privilege allowing sales managers to access the database. Since she belongs to the sales manager role and this role is now able to access the database, she would be able to do it although it was not intended by the system administrator, rising this way a semantic self-management conflict.

This kind of conflict can occur if the identity defines a rule changing a privilege which directly affects it, as well as if the rule changes the privileges of a role to which the identity belongs. The above example illustrates the second situation and the following rule is defined to detect it.

\[
\text{SelfManagementConflict} : \text{Identity}(i) \land \text{AuthorizedPrivilege}(ap) \land \\
\text{Imp}(\text{rule}) \land \text{ruleAffectsIndividual}(\text{rule}, \text{ap}) \land \\
\text{ruleDefinedByIdentity}(\text{rule}, i) \land \text{authorizedSubject}(ap, i) \land \\
\rightarrow \text{Conflict}
\]

The third line of this rule determines that the rule should have been defined by the same identity to which the privilege applies. If this situation occurs, then a self-management conflict occurs.

Given this rule, it is easy to also deal with this kind of conflict when the privilege indirectly affects the identity through the role relationship. A simple SWRL rule can be defined to state that for every privilege which is associated to a given role via the authorized subject property, another association of this type should also be created for any identity which is member of the role. This way, the reasoner will infer that the privilege applies to the identity and the conflict will be detected. Moreover, advantage of the semantic capability provided by OWL can be taken to detect the conflict.
in situations where role hierarchies are supported. Since transitivity is available in OWL, the MemberOfCollection association can be defined as transitive by using the TransitiveProperty construct of OWL. Then, privileges associated to an upper level role will also be assigned to sub-roles by transitivity and the conflict will be also detected. Thus, the SelfManagementConflict rule is able to detect the self-management conflict for these situations.

This kind of conflict may appear in different ways as, for example, by the rule adding the identity to a role it did not belong before. In the above example suppose that the sales manager role already has privilege to access the database, but Alice did not belong to that role and she defines a rule which creates the MemberOfCollection association. The proposed conflict detection method may still be applied to detect the occurrence of any other self-management conflict by means of a different set of classes or associations by just adding the appropriate detection rules.

5.3. Conflict of duties and conflict of multiple-managers

Once the two first conflicts have been explained in detail, and for clarity reasons, the two other conflicts will be described in short. For them, just the definition of the conflict as well as an example and the associated rule will be provided. In Section 7, the reader will find similarities between the examples provided for the four types of conflicts regarding both the conflict detection techniques being analysed and the scenarios where different performance measurements have been taken.

5.3.1. Conflict of duties

A conflict of duties arises when an identity is obliged to protect the interests of two or more third parties who are looking for different goals or benefits. To expose this kind of conflict, a secure communication scenario can be considered, where a software agent manages the corporative communications. This agent in charge of communications has a VPN service available to protect connections between corporate offices and it also has a sniffer to monitor the network traffic which flows through the network.

Suppose this agent is managed by the sales department and, at the same time, it is also managed by the security department. The first one orders the agent to secure every communication between offices by establishing a VPN to protect sales data, while the second department orders it to sniff the traffic for monitoring purposes. When a sales communication is to be established, a conflict of duties arises because of the two different duties the agent should accomplish at the same time. In such a situation, the agent requests the sniffer to monitor the traffic which flows through an Ethernet card. However, once the VPN is established, traffic on such a card uses a VPN which is cyphering data. The conflict comes up when trying to sniff the traffic flowing through a network card which is using a secure protocol endpoint whose security association contains an ESP transform and, therefore, the traffic is cyphered. In this situation the sniffer is only able to read cyphered traffic which cannot be used for monitoring purposes. The following rule has been defined to detect this conflict:

\[
\text{ConflictOfDuties} : \neg \text{SniffAction}(\text{sni}) \land \text{EthernetPort}(\text{eth}) \land \text{SniffActionOnNetworkPort}(\text{sni}, \text{eth}) \land \neg \text{portImplementsEndpoint}(\text{eth}, \text{ipsec}) \land \text{IPSecSAEndpoint}(\text{ipsec}) \land \text{transformOfSecurityAssociation}(\text{ipsec}, \text{esp}) \land \text{ESPtransform}(\text{esp}) \rightarrow \text{Conflict}
\]

In this rule, line 2 specifies that the Ethernet port is implementing an IPSec security association and line 3 states that this SA contains an ESP transform which is the one that cyphers the traffic.

5.3.2. Multiple-managers conflict

A multiple-managers conflict arises when two different subjects intend to perform two different actions which are semantically incompatible. A possible scenario where this occurs is when using a corporate mail server. That server may be configured by the administrator to scan every mail using an antivirus, acting as a mail agent. On the other hand, an employee sending a mail in such a company may want to encrypt the mail content to avoid it to be read by third parties. The conflict in this scenario is produced because of the different actions to be done following the intentions of both subjects. The employee cyphers a mail with its mail client and then he sends it through SMTP to
the corporate mail server for this one to deliver it to its destination. When this server gets the mail, it wants to scan it with an antivirus. Although it is possible to do it in practical terms, it is not semantically correct because it makes no sense to scan an already cyphered object. The conflict appears when trying to scan a cyphered element and the rule which detects it has been defined as follows:

\[
\text{MultipleManagerConflict : CypherableElement(?ce) \land cyphered(?ce,"true")}\land \\
\text{AntivirusScanActionRequest(?scan)}\land \\
\text{actionOnLogicalElement(?scan,?ce)} \\
\rightarrow \\
\text{"Conflict"}
\]

Line 1 defines the cypherable element specifying that it has to have its cyphered property set to true and line 3 specifies that the scan action is performed on this element.

6. Different strategies for Conflict Detection

The usage of OWL and SWRL as modelling languages for information systems where rule conflicts may appear provide different ways to detect conflicts in the knowledge base. Since our models are described in OWL, which is based on Description Logic, a DL reasoner is used and the methods in this section will make use of the processes carried out on this kind of reasoners to deal with inconsistent knowledge bases. When two contradictory facts are hold on the knowledge base, it is inconsistent and reasoners detect these situations. Thus, the consistency checking process of DL reasoners can be used to detect the semantic conflicts.

Although the management of the conflicts once they have been detected by the reasoner is initially out of the scope of this paper, it is worth mentioning some treatments that become available using the information inferred by the reasoner during detection. A first approach could be the generation of warnings to alert the user of an incident or inconsistent situation. Another more sophisticated option is to try to resolve the conflict (semi-)automatically with the aim of returning the system to an equilibrium state. To deal with this last issue, there are different techniques like the prioritization of policies [54] [55]. This technique allows to resolve the conflicts that could appear between two rules by assigning a priority to each rule in order to resolve which one is applied in case of conflict.

Following subsections expose different ways of detecting semantic conflicts by introducing an inconsistency in the knowledge base. This inconsistency will be detected by DL reasoners which will alert about the detection of the conflicts.

Two different ways for detecting conflicts have been mainly identified: i) the domain model does include concepts related to conflict detection; in this approach the administrator defines rules to identify conflicts where the rule infers a conflictive fact according to its definition in the model [56]; and ii) the domain model does not include concepts related to detect conflicts; in this approach the system identifies conflicts as inconsistencies and rules infer facts that generate inconsistencies in the knowledge base [15] [16]. Both approaches will be analysed and compared in order to determine what is the most suitable one to detect conflicts on DL reasoners. Hence, while Section 6.1, Section 6.2, Section 6.3 and Section 6.4 expose detection methods which are related to the second approach, Section 6.5 provides a detection method based on the first one. These sections will make use of the PKI scenario introduced in Section 5.1 to illustrate the different detection methods in order to exemplify them using a unified scenario.

6.1. Detection based on disjoint classes

The disjoint term applied to classes means that they cannot have individuals in common. Hence, if a disjoint statement is specified for two different classes, a reasoner is able to deduce an inconsistency when an individual is stated to be an instance of both classes. In OWL language, disjoint classes are modelled by means of the disjointWith construct. Thus, two classes which represent conflictive actions are defined as disjoint classes in the application domain. Additionally, two different instances defined in the knowledge model may represent the same individual. This equivalence between instances is modelled by the sameAs construct in OWL language. Following this approach, the conflict detection will be done by defining rule consequents which state that the conflictive instances represent the
same individual. Since the classes to which the instances belong are defined as disjoint classes, the knowledge base becomes inconsistent and the reasoner detects it.

Therefore, the rules to perform the semantic conflict detection can search incompatible actions in order to conclude a `sameAs` relationship between these conflictive actions, causing the detection of the conflict. The conflict will arise when the antecedent part of the rule finds a conflictive situation in the application domain. For instance, considering the PKI scenario exposed in Section 5.1, the `ValidateCredentialAction` and the `RequestCredentialActionLog` classes can be stated as disjoint classes. In this scenario, the consequent part of the `ConflictOfInterests` detection rule can be defined as follows:

\[
\rightarrow \text{sameAs}(?\text{validate}, ?\text{req})
\]  

(1)

Then, when the same identity (Alice) tries to execute both the request and sign actions on the same credential (Bob’s credential), the rule antecedent will be fulfilled and its consequent will assert the equivalence between the two instances representing the action of generating a certificate which rises the conflict of interests. This fact, together with the assertion of the disjoint between the two classes in the application domain will cause an inconsistency in the knowledge base which, in turn, will alert about the semantic conflict. Listing 1 represents the more relevant facts for the detection of the conflict for this scenario, using abstract OWL notation. The first set of facts are related to the description of the scenario depicted in Figure 4. The other two sets are the semantic information added to detect the conflict using this method and the results generated by the firing of the `ConflictOfInterests` rule, which causes the detection of the conflict.

### Listing 1 Conflict detection based on disjoint classes

```xml
<!-- Conflict of interests scenario -->
Identity(#Alice)
PublicKeyCertificate(#BobCertificate)
RequestCredentialLogRecord(#BobCertificateRequest)
logRecordByIdentity(#BobCertificateRequest, #Alice)
usedCredential(#BobCertificateRequest, #BobCertificate)
ValidateCredentialAction(#BobCertificateValidation)
actionRequestedByIdentity(#BobCertificateValidation, #Alice)
actionOnCredential(#BobCertificateValidation, #BobCertificate)
CredentialManagementService(#PKIService)
executedCredentialManagementService(#BobCertificateRequest, #PKIService)
credentialManagementServiceToExecute(#BobCertificateValidation, #PKIService)

<!-- Disjoint definition for conflict detection -->
disjointWith(ValidateCredentialAction, RequestCredentialLogRecord)

<!-- Fact inferred by the conflict detection rule -->
sameAs(#BobCertificateValidation, #BobCertificateRequest)
```

One advantage of using this method to detect conflicts is the traceability provided by the reasoner about the inconsistency detection. In fact, when the inconsistency is raised, the individual who caused the conflict together with the two disjoint classes involved in the conflict can be retrieved as explanation. This results in a worth and useful information when methods for conflict resolution are going to be applied after detection in order to resolve the conflictive situation. This conflict detection method requires the usage of, at least, OWL-DL since the `disjointWith` construct is defined by this language.

### 6.2. Detection based on contradictory properties

Negative properties were introduced in OWL 1.1 and lately in OWL 2. A negative property defined over an individual A pointing to another individual B states that there is no connection between both individuals by means of this property. Hence, the definition of both positive and negative statements for the same property of a given individual will determine that this individual is `and` is not related to another one. The presence of these contradictory facts in the knowledge base will raise an inconsistency which can be used for semantic conflict detection. OWL 2 provides `NegativePropertyAssertion` construct to define negative properties.
This method assumes that either a positive or a negative property is defined, connecting two different instances by means of this property. Then, the opposite relationship between these two instances can be introduced in the knowledge base in order to detect the semantic conflict. Detection rules will search in their antecedent part a conflictive situation in which a property between individuals is available and they will state in their consequent the opposite property. This will cause the inconsistency in the knowledge base, making the reasoning process to alert about the conflict. This detection method is particularly useful for cases in which the conflict cause resides in a property rather than in the membership to a particular class.

In the PKI scenario depicted in Section 5.1, a subject is trying to validate a certificate using a credential management service. The service which will execute the action is specified by the CredentialManagementServiceToExecute property. In a conflictive situation, the antecedent of the ConflictOfInterests rule is fulfilled and the conflict is raised by its consequent which asserts the negative property, stating that the credential management service should not be executed. The consequent of the rule for this detection method is defined as follows:

\[ \rightarrow \neg(\text{CredentialManagementServiceToExecute}(\text{validate}, \text{?s})) \] (1)

Listing 2 shows the knowledge which is inferred by the ConflictOfInterests rule when applied to the scenario described in Listing 1 using this detection method.

Listing 2 Detection based on contradictory properties

<!-- Fact inferred by the conflict detection rule -->
\neg(\text{credentialManagementServiceToExecute}(\text{#BobCertificateValidation}, \text{#PKIService}))

The advantage of this method is the intuitive way in which the consequent part of the rule is defined, stating by means of the negative assertion that the service should not be executed because of a conflict. However, one of the disadvantages of this method is the lost of traceability between the conflictive actions. In this method, traceability is lost because relationships between the conflictive actions may not be defined in the model and it is not introduced by the consequent of the rule. Another disadvantage is that this method requires OWL 2 to support the negative property specification. OWL 2 is currently the most expressive language from OWL which is decidable, i.e. reasoning processes can be done in finite time. However, this language is heavy in terms of computational complexity, since it requires complex reasoning techniques such as the provided in SROIQ [57].

6.3. Detection based on complement classes

Given a class A, a new class B can be defined as the complement of A. This implies that every instance not belonging to A will be considered an instance of B. From this definition, the existence of an individual belonging to a class and its complement violates the consistency of the knowledge base. Hence, this inconsistency can also be used to detect semantic conflicts.

The way in which a semantic conflict is detected by means of this kind of inconsistency is analogous to the method previously exposed in Section 6.1. Instead of defining a disjointWith construct, a class representing an action which is conflictive with another one is defined as complement of that one. This definition is done in OWL by means of the ComplementOf construct used during the definition of a class. Similarly to detection based on disjoint classes, rule consequents state that the conflictive instances are representing the same individual using the sameAs construct of OWL. This time, the classes to which the instances belong will be defined as complement classes, producing an inconsistency in the knowledge which will be detected by the reasoner.

In the example provided in the PKI scenario in Section 5.1, the ValidateCredentialAction class is defined as complement of the RequestCredentialLogRecord class. Then, as done in the disjoint classes method, when the antecedent of the ConflictOfInterests rule is fulfilled, its consequent defines an equivalence between the two conflictive instances using the sameAs construct. The consequent of the rule is, therefore, the same as the one defined in Section 6.1:

\[ \rightarrow \text{sameAs}(\text{validate}, \text{?req}) \] (1)
Listing 3 shows the complement definition and the knowledge which is inferred by the ConflictOfInterests rule when applied to the scenario described in Listing 1 using this detection method.

### Listing 3 Facts involved in the conflict detection based on complement classes

```xml
<!-- Complement definition for conflict detection -->
ComplementOf(ValidateCredentialAction, RequestCredentialActionLog)

<!-- Fact inferred by the conflict detection rule -->
sameAs(#BobCertificateValidation, #BobCertificateRequest)
```

Again, traceability is an advantage of this detection method, since a relationship between the two conflictive individuals is established by the consequent of the rule. Hence, the cause of the semantic conflict can be retrieved in order to be used in further conflict resolution methods. Moreover, this method requires OWL-DL to be used, since the ComplementOf construct is defined in this language. From a theoretical perspective, OWL-DL is decidable and it offers a good balance between expressiveness and computational complexity.

### 6.4. Detection based on the empty set membership

The empty set is a set which contains no individuals. Consequently, the existence in the knowledge base of an individual belonging to the empty set implies an inconsistency which can be used to detect semantic conflicts. In OWL, the Nothing class is an axiom defined as the empty set. Therefore, no individual can exist in the knowledge base belonging to this class. In this approach, when a conflictive situation is detected, the consequent of the rule states that the conflictive instance belongs to the Nothing class, meaning that the individual represented by this instance is not a possible thing in the scenario domain. The inconsistency is then detected by the reasoner which alerts about the conflict.

In the previous PKI scenario exposed in Section 5.1, a conflict of interest occurs when the subject has requested a certificate and she is trying to validate it. In that case, the ValidateCredentialAction action can not occur and it is defined as an instance of the Nothing class by the consequent of the ConflictOfInterests rule to produce the inconsistency and detect the semantic conflict. This consequent is defined as follows:

```
→ Nothing(?validate)
```

Listing 4 shows the knowledge which is inferred by the ConflictOfInterests rule when applied to the scenario described in Listing 1 using this detection method.

### Listing 4 Detection based on the empty set membership

```xml
<!-- Fact inferred by the conflict detection rule -->
Nothing(#BobCertificateValidation)
```

The Nothing axiom is defined in OWL-Lite. Therefore, this conflict detection method can be applied by using OWL-Lite which allows fast reasoning processes leading to a theoretically fast conflict detection process. The main disadvantage of this method is again the lost of traceability, since it does not establishes a relationship between the two conflictive instances and this information may be not available for further processing.

### 6.5. Detection based on modelling conflictive individuals

This conflict detection method is based on a different approach from the previous ones. In this approach, new concepts need to be added to the application domain in order to perform the conflict detection. Concretely, a definition of the conflictive concept can be provided by means of a Conflictive class to represent the individuals which produce the conflicts.

The way in which this conflictive concept is defined allows the definition of conflictive individuals in the application domain when a semantic conflict is detected. But, in order to make the reasoner process to alert about the conflict,
an inconsistency should be produced in the knowledge base. This can be achieved by defining the *Conflictive* class disjoint with the *ManagedElement* class, which is the root of the domain hierarchy and represents every individual of the managed information system. The disjoint between this class and the *Conflictive* one means that the elements of the system can not be conflictive ones. Thus, in this detection method, the consequents of rules define that the instance representing the conflictive individual is an instance of the *Conflictive* class. The existence of such an instance belonging to both the *ManagedElement* and the *Conflictive* classes represents an individual which is a conflictive element and it causes inconsistency in the knowledge base.

Applying this approach to the PKI scenario of Section 5.1, the conflict arises when the subject tries to validate the certificate and the consequent part of the rule determines that this validation action is the conflictive individual, defining it as an instance of the *Conflictive* class. The rule consequent is defined as follows:

\[ \rightarrow \text{Conflictive(?validate)} \] (1)

Listing 5 shows the *Conflictive* class definition and the knowledge which is inferred by the *ConflictOfInterests* rule when applied to the scenario described in Listing 1 using this detection method.

---

**Listing 5** Detection based on modelling conflictive individuals

<!-- Conflictive class definition for conflict detection -->

\[
\text{Class(Conflictive)}
\]

\[
\text{DisjointWith(Conflictive, ManagedElement)}
\]

<!-- Fact inferred by the conflict detection rule -->

\[
\text{Conflictive(#BobCertificateValidation)}
\]

---

The main disadvantage of this approach is that it requires the addition of a new concept to the model in order to provide a representation of conflictive individuals, which implies having an artificial concept in the model that does not really belongs to the domain. As an advantage, since this method needs the addition of new concepts to the domain model, they can also be used to represent information which may be used by further processes. For instance, a property may be added to the *Conflictive* class in order to define the relationship between the conflictive instances for traceability purposes. Finally, OWL-DL is required in order to apply this method since the *DisjointWith* construct is defined therein.

**7. Results and Performance**

Section 6 described several conflict detection methods. In order to evaluate them, these methods or strategies have been applied to a realistic information system management scenario. Concretely, the one used to illustrate the *Conflict of Interests* in Section 5.1 has been selected in order to provide a complete overview of such scenario, whose results can be extrapolated to other scenarios by analogy. Analysing all these conflict detection methods contributes to determine a quality ranking of them. Previous research works on conflict analysis usually lack of this analysis results. This section tries to go further by providing performance measures for each analysed method.

Moreover, in order to provide some metrics about the detection of different kinds of conflicts, performance analysis have also been done for the different conflict types exposed in Section 5, using the scenarios described in such section for each one. This analysis has been done after the detection methods have been examined. Thus, the best detection method resulting from this previous analysis is the one chosen to perform the measurement of the different kinds of conflicts.

The general domain exposed in Section 3 has been chosen to do the performance analysis. As stated in Section 3, in order to represent our knowledge model, the ontology provided by Majewska et al [45] has been used to represent CIM in OWL. This ontology has been slightly extended it in order to include the concepts exposed in Section 3 and Section 5. Moreover, since some of the conflict detection methods exposed in Section 6 require the adaptation of some aspects of the model to perform the detection, the ontology has been modified accordingly, thus resulting in a different ontology for each detection method. All these ontologies represent the same domain model depicted in Section 3, but they include the needed changes required by the particular detection method. Thus, for the conflict
detection based on modelling conflictive individuals a new class called \textit{Conflictive} has been added and it has been defined as \textit{disjointWith} the \textit{ManagedElement} class. Similarly, it has been necessary to include the \textit{disjointWith} and the \textit{ComplementOf} constructs for the detection based on disjoint classes and the detection based on complement classes methods, respectively. Detection based on the empty set membership does not require any ontology change, but a suitable detection rule. Finally, another issue that should be addressed is the definition of the conflict detection rules. These rules have been modelled in SWRL and their antecedent have been defined following the rules defined in Section 5 for each kind of conflict, while their consequents have been specified according to Section 6 for the different detection methods.

To obtain the performance measurements, a straightforward Java application has been developed in order to manage the ontologies and the reasoner function for the different conflict detection methods. There are several suitable implementations of DL reasoners such as Pellet [58], Jena [59], KaoN [60] and FacT++ [61], among others. The developed application performs different tests over the ontologies using Jena [59] as a general Java API to manage ontologies and Pellet [58] as DL reasoner. Namely, it uses Jena in its version 2.5.7 and Pellet in its version 2.0.0. The main reason why Pellet has been chosen is due to the high expressiveness it supports, dealing with OWL 2 language. On the other hand, Jena is nowadays the standard defacto to manage ontologies.

Due to current DL reasoner implementations, the conflict detection based on contradictory properties cannot be executed. Although current DL reasoners deal with negative properties, they do not support the \textit{NegativePropertyAssertion} construct of OWL 2 when it is defined on the consequent of a SWRL rule. Therefore, it has not been possible to consider this strategy when obtaining performance results.

Finally, in order to check the performance and behaviour of the reasoner when it is under different work loads, several tests have been done using different numbers of individuals present in the knowledge base. Next subsection goes deeper into the different populations used during the analysis.

7.1. Populations

A possible approach to measure the performance of the conflict detection methods is to perform several executions, starting from a simple one and bringing it more and more complex. This complexity is achieved by increasing the number of individuals present in the \textit{ABox} component that, in turn, increases the number of statements of the knowledge base. The number of individuals contained in the \textit{ABox} for a given execution is referred to as population. Notice the difference between the number of statements that can be hold in the \textit{ABox} component and the number of individuals that are defined by those statements.

Another issue to be considered is the mathematical function used to establish the populations size because it will be used to evaluate the performance and scalability of the conflict detection methods. The mathematical function which determines these populations size has been empirically achieved. It has been designed bearing in mind that the biggest population should be the one which shows unrealistic performance results in terms of response time or amount of required memory. Therefore, the biggest population in this testbed has about 181000 individuals, since for this amount the reasoning process spends around 10 seconds to perform the reasoning, which is usually considered as inadmissible response time. The function used to determine the populations size is the following one:

\[
 f(x) = 2000 \cdot e^{(x/2)}
\]

Where x takes integer values from the interval [0, 10). The following table shows ten populations used on the performance analysis obtained using the previous mathematical function. Notice that the last population has more than 180.000 individuals which, in turn, will result in a big amount of statements whose exact number depends on the concrete scenario. For instance, a population of 180.000 individuals leads to more than 2.2 millions of statements for the semantic conflict of self-management.

<table>
<thead>
<tr>
<th>Population</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># Individual</td>
<td>2000</td>
<td>3300</td>
<td>5500</td>
<td>9000</td>
<td>14800</td>
<td>24400</td>
<td>40200</td>
<td>67000</td>
<td>110000</td>
<td>181000</td>
</tr>
</tbody>
</table>

Table 1: Number of individuals by population
Once the population size has been defined, it is necessary to define the percentage of individuals which will belong to each class of the application domain. This definition has been done bearing in mind the peculiarities associated to realistic scenarios. In this sense, Figure 7 depicts four graphics of the four conflictive scenarios previously described in Section 5. As can be observed, each graphic shows the percentage of individuals belonging to each class. For example, in the PKI scenario which illustrates the conflict of interests, there are around 63% of individuals belonging to the Log class against 1% belonging to the CredentialService class. These percentages intend to model realistic scenarios and they will be used to generate the populations.

Figure 8 shows a graphic depicting the relationship between population size and the number of statements for each scenario. As can be observed, the number of statements is bigger than the number of individuals. Moreover, as the population grows, the amount of statements is also increased, although this increase is not necessarily proportional. On another front, the self-management conflict scenario generates a big amount of statements. This is because this scenario defines SWRL rules and every single rule requires several statements to be represented.

7.2. Simulations

The aim of this subsection is to describe the different simulations that have been done in order to evaluate the diverse conflict detection strategies. The detection methods exposed in Section 6 have been tested by measuring the time that is needed to identify a conflict in the knowledge base. As previously said, the PKI scenario depicted on Section 5.1 to illustrate the conflict of interests has been used. The machine used to run the simulations has been a Core 2 Duo T7500 at 2.2 Ghz with FSB 800Mhz and 4 Gb RAM. Regarding the operating system, a common clear Ubuntu Linux in its version 8.0.4 has been used. Finally the used Java Virtual Machine has been the Sun JDK 1.6 tuned up to 2Gb of maximum heap size.
The testbed is basically composed of three different tests executed against ten incremental populations. The first test deals with the load time, i.e., the time needed to load the full ontology (ABox & TBox) in the knowledge base. The second one measures the time that the Pellet reasoner spends in consistency checking when the knowledge base does not hold any inconsistency fact. It should be noticed that a knowledge base is consistent when it does not contain any contradictory fact. Finally, the third and most important test is about the time that the rule reasoner requires to detect the conflict when there is at least one inconsistency fact present in the ontology. This time is referred to the consistency checking operation that rises the conflict in the reasoner.

Figure 9 depicts the performance results obtained by the execution of the third aforementioned tests with the selected populations. Notice that, while the consistency checking is performed in the system, the rule that detects the conflict raises and infers the information that produces the inconsistency in the knowledge base. The consequent of this detection rule is defined according to the detection strategy which is being tested.

The achieved results conclude that the detection based on conflictive individuals is the best conflict detection method when it is compared with the other ones. This result seems to indicate that conflict detection methods which include new concepts in the application domain to model the conflictive situation provide better performances than those which do not include such concepts. Moreover, among these last type of detection methods which does not include new concepts, the one which has proven better results has been the detection based on disjoint classes. Finally, the detection based on the empty set membership has been empirically determined as the worst detection strategy, even though it uses OWL-Lite.

The results rank the detection based on conflictive individuals and the detection based on disjoint classes in a better position than the other ones. The performance difference between them is due to the different algorithms used by the reasoner to check the knowledge base consistency and it is empirically proved that the first one provides better results.

Once the different detection methods have been evaluated, the next step is making an analysis to evaluate the behaviour of the best detection strategy when it is applied on the scenarios exposed in Section 5. Since the best detection strategy is the conflictive individuals, the testbed has been launched for each scenario using this detection method.

Figure 10 contains four graphics showing the results obtained for the four analysed scenarios. Each graphic illustrates two different metrics. One of them defines the time required to perform the consistency operation in a non-conflictive scenario, while the other one defines the time spent to perform this operation in a conflictive scenario. In conflictive scenarios detection rules will be fulfilled, causing the insertion of new facts in the knowledge base which, in turn, will cause inconsistency. It should be noticed that load times are included in the metric values in order to show the overhead introduced by loads in the reasoner.

Some conclusions arise from the results shown in Figure 10. Each scenario has its own peculiarities and therefore the consistency checking operation shows different statistics. Thus, the conflict of duties scenario is the most complex due to the amount of different instances coming from different classes needed to represent it. This leads the reasoner
Figure 9: Detection methods performances

Figure 10: Scenario performances

to require more time to check the consistence. Hence, the conflict of interests and the conflict of duties show different levels of performance, in spite of sharing almost the same number of statements.

A special case is the performance achieved in the self-management conflict. In this case the graph shows that for both scenarios, with and without a conflictive knowledge base, the times are outrageous compared with the others charts. This is due to the fact that this scenario makes use of SWRL rules. Thus, when the reasoner has instances of SWRL rules present in its Knowledge Base, the time it spends to perform the consistency checking, which also
executes the rules, is quite higher.

Furthermore, these results show that the relationship between the time spent by the reasoner to check the consistency of an inconsistent knowledge base compared with the time spent to check a consistent one vary, once more, according to the complexity of the scenario which is being analysed. Although, in general, the time spent by the reasoner to check the consistency is higher when dealing with a consistent knowledge base than when dealing with an inconsistent one, it is quite difficult to give a real estimation about this relationship because of its domain dependency.

8. Conclusion and Future Works

Conflict detection is a problem that has been covered by many different research works so far. However, most of them have been focusing on detecting modality conflicts (e.g., when positive and negative authorizations over a target are granted to a subject at the same time) and not many on semantic conflicts, i.e., those depending on the particular application domain where the rules are being applied. This second group of conflicts is usually more difficult to detect. Additionally, most of the current works on semantic conflicts deal with non-standard information models, which limit the deployment of provided solutions in real information systems.

This paper focuses on the detection of conflicts of semantic nature. As these conflicts have been categorised by different authors and there is not a common agreement, this paper is first doing an effort to define a common taxonomy including most of the relevant works in this field, including both modality and semantic conflicts. From the list of semantic conflicts, this paper pays special attention to four types that can be considered as more usual when dealing with advanced information systems, i.e., conflict of interests, self-management conflict, conflict of duties, and multiple-managers conflict. For each of them, specially the first two ones, a realistic domain is described and an example of conflict is presented.

Then, five different strategies for conflict detection are presented to the reader and exemplified using a realistic conflict of interests that may happen in a PKI scenario. These five techniques are being presented by describing their pros and cons and also a performance measurement is provided, determining that the detection based on modelling conflictive individuals is the conflict detection approach which obtains the best performance.

This performance measurement has been done on realistic scenarios dealing with the management of information systems. They include the existence of conflictive rules when managing a PKI, as indicated before, but also when sending a ciphered email message, when establishing a VPN or when modifying the privileges in a database. This performance measurement is another important result of the paper and provides an innovation over current existing research works in this field.

According to the results, the semantic conflict detection method which includes new concepts in the application domain to model conflictive individuals has obtained the best performance. It has also been empirically shown that for all cases, the time needed to check the scenario consistency is higher when there is no conflict in such scenario.

Our work is now focused on the design of a distributed architecture in which semantic conflicts can be detected. Moreover, another future work will be to analyse and define conflict resolution processes and techniques for enabling the system to (semi-)automatically return to a normal state. Likewise, the design and development of an expressiveness extension for policy languages in order to get constructors and operators in these languages to define, model, and establish conflict situations and relationships between conflicts and causes would be another interesting research. Finally, a detailed study on how conflict processes could be applied to scenarios modelled by means of stochastic approaches, in which some part of information is assumed and it is not explicit on the scenario representation, is also being considered.

Acknowledgements

This work was supported by the Spanish MEC and MICINN, as well as European Commission, under the Grants AP2006-4150, CSD2006-00046, TIN2008-06441-C02-02, TIN2009-14475-C04 and FP7-ICT-2007-1 SWIFT. Thanks to the Funding Program for Research Groups of Excellence with code 04552/GERM/06 granted by the Fundacion Seneca.
References


K. Racine, Q. Yang, Redundancy detection in semistructured case bases, IEEE Transactions on Knowledge and Data Engineering 13 (2001) 513–518.


Jose M. Alcaraz Calero received the Computer Engineering and Master degrees with honors at the University of Murcia. He has been working at University of Murcia since 2004 in several European and international projects whereas he is doing his PhD. Currently he is research staff at Hewlett Packard Laboratories.

Juan M. Marín Pérez is a research staff in the Department of Information and Communications Engineering of the University of Murcia. He received Engineering and MSc degrees in Computer Science from the University of Murcia. He has been working in several European projects while doing his PhD. His research interests include security and management of distributed systems.

Jorge Bernal Bernabé received the Computer Engineering and the MSc in Computer Science from the University of Murcia. Currently, he is a research staff in the Department of Information and Communications Engineering of the University of Murcia. He has been working in several European projects while pursuing his PhD. His scientific activity is mainly devoted to the security and management of distributed systems.

Félix J. García Clemente is an assistant professor of Computer Networks at the Department of Computer Engineering of the University of Murcia. His research interests include security and management of distributed communication networks. He received an MSc and PhD degrees in Computer Science from the University of Murcia.

Gregorio Martínez Pérez is an associate professor in the Department of Information and Communications Engineering of the University of Murcia. His research interests include security and management of distributed communication networks. He received an MSc and PhD in Computer Science from the University of Murcia.

Antonio F. Gómez Skarmeta received the MS degree in Computer Science from the University of Granada and BS (Hons.) and the PhD degrees in Computer Science from the University of Murcia Spain. He is a Full Professor in the same Department and University. He has worked on different research projects at regional, national and especially at the European level in areas related to advanced services like multicast, multihoming, security and adaptive multimedia applications in IP and NGN networks.