Incremental Integration of Fragmented Knowledge Via the Edition Protocol of a Shared Knowledge Base

Philippe A. Martin
EA2525 LIM, I.T. Department,
University of La Réunion
97400 Saint-Denis, France
Philippe.Martin@univ-reunion.fr

***Abstract*—The main approaches to knowledge representation and sharing (KS) focus on easing the exploitation and exchange of knowledge representations (KRs) between particular agents or for particular applications. These approaches lead to the proliferation of mainly independently developed KR bases (KBs) which are mutually partially redundant and contradictory, thus restricting a more general KS. This article shows how a shared KB edition protocol can support the incremental and cooperative integration of fragmented knowledge into a (networked or not) consistent well-organized KB, without manual or automatic selection between contradictory KRs. To that end, the protocol helps and enforces i) the setting of particular relations between "competing" KRs (i.e. those that are contradictory or at least partially redundant) to compare them and justify their joint existences in the KB, and ii) a representation of them that keeps the KB consistent. Besides providing more knowledge search and inference possibilities, these relations can be exploited by default or user-defined rules for KR filtering and choices between them. This article also shows that this approach can be implemented in various ways and be used with most inference engines.**

Keywords—fragmented knowledge, knowledge integration, knowledge representation, knowledge sharing

I INTRODUCTION

Knowledge and framework for integrating and sharing fragmented knowledge. *Knowledge representations* (KRs) are logic-based representations of *semantic* relations between real or imaginary things, stored in a *Knowledge Base* (KB). A *KB object* is either a type, an individual or a statement (an assertion of relations). KBs are composed of an *ontology* – concept/relation types (and individuals) related by relations defining them – and a *base of facts* that contains the other statements. The more relations there are between KB objects within and between KBs, the better for knowledge inferencing and retrieval via queries or navigation along relations. According to [1], "fragmented knowledge" refers to a low or inadequate organization of such knowledge, and is due to the absence of a "framework in which to relate the tremendous volume of ideas, data and insights which every person meets at work and at home through the years". As noted in [1], databases and natural language documents cannot support such a framework (since, unlike KBs, they do not enable people to represent and relate their knowledge via semantic relations).

Knowledge sharing goals. Although [1] does not give requirements for the above-cited framework, the given descriptions and the claim that this framework does not exist are indications about the kind of *knowledge representation and sharing* (KS) this framework is related to. This article distinguishes two main kinds. The first one, here called "*restricted KS*" is insufficient for the above cited framework. Indeed, restricted KS is about i) easing the exchange of KRs between *particular* agents (businesses, persons or applications) that *can* discuss with each other to solve ambiguities or other problems, and ii) the *complete or efficient* exploitation of those information by these particular agents, for particular applications. The second KS kind, here called "*general KS*" (gKS), is about letting people represent and relate information within or between KBs in ways that maximize the retrievability and exploitation of the KRs by *any* person and application. Examples of well-known early works for general KS were Ontolingua (server, ontologies and vision) [2] and Freebase [3] (which has been reused to create Google's Knowledge Graph). Restricted KS and gKS are unfortunately rarely distinguished, including by the World Wide Web Consortium (W3C). Regarding KS, the W3C has a "Semantic Web vision" [4] of a Web of *Linked Data* [5] which is about restricted KS. Indeed, it mainly only encourages i) data contributors or managers to index data by few KRs, and ii) KB contributors to relate their KB objects to some others in other KBs, thus only lightly relating *mostly independently developed* KBs, not creating cooperatively built KBs. The W3C does not yet advocate particular gKS related techniques or resources such as top-level ontologies, ontology alignment, KB quality evaluation, ways to manage networked KBs (i.e. KBs composed of a network of other KBs, down to non-networked KBs) and ways to manage the cooperative building of a same (networked or not) KB.

The problems caused by the implicit contradictions and redundancies across KBs created by classic knowledge-sharing approaches. Almost all approaches for shared KB building or knowledge integration are based on i) manually or automatically selecting some objects from existing KBs – e.g. according to logical consistency between knowledge objects and their domains – and ii) with these objects, creating one or several *consistent* KBs or creating several *competing* modules to be imported in KBs. Such KBs or modules represent mutually inconsistent views or theories, e.g. one for Newtonian physics and one quantum physics. (From now on, "KBs" also refer to modules.) In these approaches, an implicit *assumption* (which the next paragraph challenges) or goal is that an inference engine exploiting a KB has to – or should – be able to *directly exploit its whole content, i.e., without an automatic pre-selection between competing views or ideas*. With this assumption, since classic logics and hence most inference engines cannot handle inconsistencies, i.e., since most KB management systems do not handle inconsistency-tolerant logics (e.g. paraconsistent ones or multi-valued ones) or reasonings (e.g. defeasible ones and those based on belief revision), the whole KB *cannot* include competing views or theories. For gKS purposes, avoiding inconsistencies in a shared KB cannot be achieved by having a person or a committee decide whether to *accept or not* each new statement submitted to the KB. Indeed, this process is too slow to be scalable and it is important for gKS to preserve the possibilities for KR end-users to make selections themselves according to

their particular needs. Similarly, gKS cannot use solutions that discard knowledge which may be of interest to some users or for some applications, e.g. solutions based on selecting only consensual KRs or only KRs from a largest consistent subset of the KB. Automatically dispatching submitted statements into various KBs for each of these KB to be internally consistent – e.g., as in the Co_4 protocol [6] for building consensual KBs – is also not scalable: with such a method, the number of required KBs can grow exponentially and these consensual KBs may be mostly redundant with one another. More generally, any approach that leads to creating new KBs which are inconsistent or at least partially redundant with each other without explicitly representing inter-KB relations between the mutually inconsistent or partially redundant objects – e.g. the classic approach of using different files for different versions of a KB, hence different KBs without inter-KB relations between objects – makes the whole body of knowledge harder and harder to manage *for gKS purposes*. Indeed, i) such an approach leads to globally less inter-related objects between the KBs (even if the absolute number of inter-relations between the KBs increases, their relative number decreases, i.e., there are many more objects without direct or indirect relations with objects in other KBs), and hence ii) such an approach forces *each* KB creator interested in reusing some KBs to choose between them or select knowledge from them and integrate them, thus creating yet another KB partially redundant or inconsistent with these KBs and with few inter-KB relations to allow automatic comparisons or choices between all the objects of these KBs.

Consistent KBs relating competing knowledge objects for enabling people and inference engines to choose between them or compare them. This article shows that the previously cited assumption is incorrect and that not making it provides a solution to the above-listed problems. Here are four underlying reasons. First, a *loss-less* integration of KBs into a shared one is possible: *competing* knowledge objects – i.e. those representing concepts or ideas that contradict each other or are at least partially redundant with each other – *can* be stored into a shared logically consistent KB using various complementary means such as contexts representing who created or believe in the ideas and the precisions making them true (times, places, modalities, etc.). Second, a cooperatively-built KB can have an *edition protocol* which, when it detects that an object addition or update would create an inconsistency or redundancy with an already existing object in the KB, requests the entering of at least one relation between these objects that *justifies* the update (e.g. a relation of correction and/or specialization, with, optionally, arguments justifying why this new relation is correct according to its author). Thus and third, in the cases when the used inference engine has to choose between competing statements for making inferences to answer a query, the engine can perform this choice according to default rules (e.g., "choose the correcting statement unless it has itself been more *justifiably corrected*) or rules given by the author of the query. If the engine is not able to perform such choices, e.g. because it is not able to exploit the given rules, the edition protocol must be generalized to an input-output (I/O) protocol that also handles search/update queries and knowledge display; to obtain the adequate results, this protocol must then exploit features of the used KB system (e.g. partitions in its workspace) and of the used engine. Fourth, to cope with the possible proliferation of competing objects, query results – and, more generally, the display or not of objects in the KB – can be adapted according to default *display rules* (e.g., "do not show the content of *justifiably* corrected statements, just show symbols showing that they exist and allowing to access their content") or display rules given by the author of the query.

Requirements; general model and shared KB editing rules based on these requirements. Section II first introduces the two general *representation requirements* that this approach entails for the KB. Then, it details the consequences of these requirements for the KB objects, especially the types of the relations needed to connect competing objects for *organizing* them, *justifying* their existence wrt. each other, and thus *choosing between them* via rules and, more generally, *avoiding the above-cited problems* that these competing objects would otherwise create. Some core default rules for the previously cited edition protocol are also given. These types are *defined* in an ontology here referred to as "TopOntolForGKS" [7], and the rules will be too. Several of these rules were hardcoded in the shared KB server created by the author of this article: WebKB-2 [8]. This tool is now re-engineered to exploit this ontology (which the users of this server can extend) and thus i) let KB creators define protocol rules for their KB, e.g. by extending the default ones, and, similarly, ii) let KB users define their own rules for automatic choices between competing objects or for display rules. For the proposed approach to work, the KB edition protocol rules ensure that when a KR update made by one user has consequences on another user's knowledge, this update is not *destructive* but *additive*, i.e., made via the addition of relations – e.g. correction relations with argument relations for the correction. In Section II, the edition rules are high-level, i.e., they are independent of the used KRL (Knowledge Representation Language) and used inference engine. Since this genericity is possible, designing a particular logic or KRL for implementing such rules would be pointless, at least for gKS. When the competing KB objects are statements, the proposed approach could be seen as a formal argumentation framework [9]; however, unlike such other frameworks, this approach remains purely logical and does not introduce new inference rules nor a new logic. Indeed, the edition protocol can exploit an existing inference engine to detect when objects are competing and then suggest probably relevant relations for connecting these objects. This approach is complete if all pairs of competing objects are detected and then related, but this is not an all-or-nothing approach: simply, the fewer pairs are missed, the less the above-cited problems are likely to occur and compound. This approach can also be applied to a networked KB for avoiding contradictions and redundancies between its component KBs and within them, but additional rules for forwarding knowledge and queries between the component KBs are needed. This article does not introduce these other rules.

Representations or implementations. Section III shows how the edition (or I/O) protocol can i) work with most KRLs, even those not particularly expressive, and ii) be represented or implemented in different ways, e.g., constraints, queries or functions (hardcoded or given by the users). These two points show that the proposed approach could be adopted to extend existing shared KB systems, even if they do not handle contexts (but implementing turnarounds could be a bit cumbersome for the developers and maybe the end-users).

Answered research questions. These sections answer the following research question: how to support an incremental integration of fragmented knowledge via a shared KB edition (or I/O) protocol that i) is *generic* wrt. the used inference engine, and ii) does not restrict what knowledge can be entered

in the KB (except when this knowledge is outside the stated domain of the KB; in this case, if the KB is part of a networked KB, the knowledge can be automatically forwarded to a more relevant KB of this networked KB)? For readability and conciseness purposes, further comparisons between approaches or methods are made within the sections, when the principles and their rationale are presented, not in a separate section.

II General Model And Rules For a Shared KB

II.A General Requirements And Terminology

Two requirements to be enforced by the KB editing protocol of a cooperatively-built KB server. The two most general requirements (or sets of requirements) of the proposed approach are presented in the next two paragraphs and were justified in the introduction. The next three subsections show which kinds of objects needs to be taken into account and how. Unlike a private KB, a (networked or not) shared KB requires a KB server (generally reusing a Web server) which manages the updates and enforces the editing rules via a KB edition protocol. To this day, almost all shared KB servers had no KB edition protocol: once they allow a user to update a KB, this user is allowed to update any object in it, even if it was created by another user (and, sometimes, as in Freebase [3], even if this makes the KB inconsistent).

The object ownership requirement. The owner of an object is a (representation of the) user that created the object (e.g. the creator of a Web document where the object was extracted) and, more generally, the (group of) agent(s) that can update the object. The present requirement is that i) this ownership is represented in the KB, ii) only the owner of this object is allowed to make a destructive update to it, and iii) if this non-additive update has consequences for the knowledge of other users, the original object should first be manually or automatically "cloned" (see details below) to avoid these consequences. The words "source" and "owner" will now be used interchangeably, depending on which seems more natural.

- **Object source representation.** For objects that can be inconsistent with each other – i.e., as explained below, for statements that are not definitions – representing its source should use a relation from the statement to express that its source is both its creator and believer. This statement is thus contextualized by this *believer* relation – it is then here called a "belief" (even though the proposed is not at all akin to belief revision) – and inconsistencies are technically avoided. E.g., "John believes that birds fly" is not inconsistent with "Jack believes that most healthy adult carinate birds are able to fly" even though what John and Jack believe here are contradictory (and hence competing) ideas.
- **Object cloning.** The cloning of an object (that is used by other objects) before it is destructively updated by its owner, means that: i) the protocol first asks the author of the update if this author can create the versions of the object before and/or after the update and relate them via a relation of type `pm:previous-version` (defined in TopOntolForGKS), ii) if the author can indicate this relation, the protocol considers the previous version as the clone; otherwise, the protocol creates the clone (i.e., from a semantic viewpoint, the original version) by giving the to-be-updated object a new owner (e.g. any of the owner of previously cited other objects, as long as this keeps the KB consistent), then iii) wherever the identifier of the to-be-updated object was used, the protocol replaces it by the identifier of the clone, and finally iv) the update of the considered object is performed. If this update is a modification, not a removal, the new object is competing with the clone and hence the protocol also follows the edition rules necessary to comply with the next requirement.

The competing object comparison requirement. Competing objects should be connected by *correction*, *specialization* or *equivalence* relations in order to i) organize these objects via these transitive comparison relations, ii) represent which object is specialized and/or corrected, and iii) when necessary, allow people and inference engines to choose which object is the most relevant according to the user's preferences or goals. E.g., a user John can represent that "John believes that `most healthy adult carinate birds are able to fly´ is a corrective specialization of Jack's belief `birds fly´ " (a *corrective specialization* relation is one that would express a specialization relation if it was not a *correction* relation). Competing objects generally have different sources, e.g. successive versions of the same KB. When changes are directly made within a shared KB, a newly modified object and its clone (in the above-described sense) are competing objects.

Rationale of the used terminology. In some KR-related terminologies, unlike in this article, the word "relation" is only used for referring to a relationship between *real-world* entities while other words are used for referring to the *representations of such relations*, e.g. "predicate" in Predicate logics, "property" in RDF and some knowledge graph formalisms [10], or "edge" in another [11]. In this article, the words "relation", "types", "statements", "meta-statements" and "contexts" have the meanings given in the introduction because i) these are common meanings in KRLs, e.g. in Conceptual Graphs [12], and ii) these words are more intuitive, general (hence not tied to a particular formalism) and easy-to-use. As a quick reminder, a (KR) "object" is either a *type*, *individual* or *statement*, and a *type* is either a *class* or a *relation type*.

II.B Consequences For Terms

The object ownership requirement for terms. Objects that are terms – atomic terms (types or individuals) or composed terms (expressions that are not atomic terms nor statements) – do not assert anything. Hence, they cannot be inconsistent with each other and cannot be contextualized. However, a term identifier can have a *source* relation (e.g. a *creator* relation) to represent its source, e.g. an ontology or a user of the KB. Instead of explicitly using such a relation, an atomic term identifier can include the identifier of its source as a prefix or suffix, as long as this makes the source identifier automatically retrievable. E.g., in W3C KRLs such as RDF/XML and Turtle, if `dc:creator` is declared as referring to a *creator* relation type (alias, "Property" in these KRLs), with `dc` being declared as an abbreviation for a URI of the Dublin Core ontology, inference engines understanding these KRLs can "dereference" `dc:creator` to retrieve and access the Dublin Core ontology. An update of a term is an update of its identifier or of its definitions. Conversely, an update of a definition (e.g. a subtype relation) is an update of a term. Only the term owner should be allowed to make this update. A modification is here seen as a removal followed by an addition.

The competing object comparison requirement for terms. Since terms do not assert anything, a set of terms cannot be inconsistent but two terms can be compared by an *exclusion* relation: for types, this means that they cannot share subtypes or instances; for individuals, this means that they are different. Similarly, terms can have partially or fully redundant meanings in the sense that two terms can be compared via a specialization or equivalence relation: a type can be subtype or equivalent to another; an individual can be instance of a type, identical to another individual or can specialize another. E.g., `pm:Camberra-in-2020` (an individual representing the Australian capital city in 2020, with `pm` referring to the author of this term and of this article) specializes `pm:Camberra-between-2000-and-2023`. When two terms in a (networked or not) KB have an *exclusion*, *specialization* or *equivalence* relationship (for terms, this is what "competing" translates into), this relationship should be represented – for gKS purposes and for the reasons given in the introduction. This representation may be direct or indirect: if this engine cannot deduce the relation based on definitions associated with the terms, the relationship needs to be manually represented. This representation is ensured by the shared KB edition protocol.

KB edition protocol for term additions. Before permanently accepting the addition of a new term that is submitted to the KB by a user, the protocol checks that this term has an automatically retrievable source and that, between this term and each other term already existing in the KB, there is i) a (direct or not) relation either stating that one of these two terms specializes the other or stating that none of these terms can specialize the other (e.g. because there is an exclusion relation between them), and ii) a (direct or not) relation either stating that these two terms are equivalent or stating that they cannot be equivalent (e.g., because one is a strict specialization of the other, as opposed to a "specialization or equivalent" term). [13] shows a way to implement such checks.

Compliance easiness. An easy way for KB contributors to comply with this protocol is to declare and organize types using subtype partitions (e.g., via `owl:disjointUnionOf` constructs for classes in OWL) and similar complementary constructs that indicate whether the specializations of a term are exclusive (in addition to specifying that the specializations are strict). The ontology that the author of this article named Sub [14] declares and organizes such constructs (as well as the relation types they are based on) and partially or fully defines them in various KRLs (e.g., in OWL [15], these constructs can only be fully defined for classes, not for relation types, nor for individuals). Using such constructs does not take more time than only using specialization relations but leads to the assertion of more relations and makes the manually entered KRs more readable.

Completeness advantage example. With respect to the existence or not of the specialization, equivalence and exclusion relations between terms, complying with the previously cited requirement makes the KB *complete* in the following sense: using the "closed-world assumption" (i.e., any statement not represented in the KB is assumed to be false) and the "unique name assumption" (i.e., different identifiers are assumed to refer to different things) do not lead to any more inferences regarding the above-cited relations – in other words, at least wrt. these relations, the KB supports these inferences without having to make these assumptions. This for example means that any term or statement in the KB can be searched not only for what they represent – via the specialization and equivalence relations; these are the classic conceptual searches – but also i) for what are they are exclusive with (as with a search for healthy birds that cannot fly), and ii) for what they are not exclusive with. E.g., assume that the notion of "Emergency-lodging" is added to the KB and represented as a specialization of "Lodging". If this type has been represented as not exclusive with "Sports-hall", some sports halls could be automatically found to be adequate emergency lodgings in case of a natural disaster. On the other hand, if "Lodging" was represented as exclusive with "Sports-hall", the notions of "Emergency-lodging" and "Classic-lodging" may first be represented as exclusive subtypes of "Lodging" and the exclusion between "Lodging" and "Sports-hall" updated to be an exclusion between "Classic-lodging" and "Sports-hall" (in the way allowed by the next paragraph).

KB edition protocol for a non-additive term update (hence, directly or not, for the removal of an identifier or a defining relation of a term). As previously implied by the above given two general requirements, if the update has no consequence on other users' objects (i.e., if this term, its equivalences and specializations are not used in these objects, and hence its update cannot change their meanings), there is nothing to do. On the other hand, if there are consequences, the protocol should first retrieve or generate the clone of the original term and then, if the update was a modification, handle it as a term addition – hence, ask for a relevant relation between the new term and its clone, after suggesting one such relation if it can. In the example given at the end of the previous paragraph, the exclusion relation between "Lodging" and "Sports-hall" is updated but, since the relation is a defining one for these two terms, the protocol handles this update as a modification of these terms. Thus, assuming this update has consequences on knowledge from sources other than the exclusion updater, the protocol first asks this agent for a `pm:previous-version` relation. Here, the agent can – and hence does – provide such a relation: "Classic-lodging" is the clone and subtype of what "Lodging" referred to before the distinction between "Emergency-lodging" and "Classic-lodging" was introduced. The update of the exclusion relation is performed if it does not make the KB inconsistent and if the required relations for competing objects are represented (here, they are represented, thanks to the representation of the specialization of "Lodging" by "Classic-lodging").

II.C Consequences For Definitions

The object ownership requirement for definitions. Since a definition relates a term identifier to a definition body, a specification of the definition creator is not mandatory: by default, it can be assumed to be the creator of the term identifier.

The competing object comparison requirement for definitions. Definitions are always "true, by definition": the meaning of the term they define is whatever the definition specifies (thus, if a definition of a term is self-contradictory, this term refers to "something impossible"). Thus, like terms, definitions cannot be "believed in" nor "corrected by someone that is not the creator of the defined term". If a new definition of a term is added by the creator of this term and is competing with earlier definitions of this term from this creator, the new definition cannot be accepted: this is a modeling mistake. If a new definition of a term is added by another user and is competing with earlier definitions of this term from its creator, this definition also cannot be accepted: this other user has misinterpreted the intended meaning of the term. Finally,

definitions are not organized via direct relations between themselves but via the terms they define.

II.D Consequences For Beliefs

The object ownership requirement for "beliefs" (statements that are not definitions). In this article, as above introduced, a belief is a statement with an outermost context representing the creator and believer of this statement. Other contexts – i.e. other meta-statements specifying precisions that make them true (times, places, modalities, etc.) – can be used.

The competing object comparison requirement for beliefs. Beliefs can be false and hence corrected. Thus, the five kinds of primitive relations with which they can be compared and organized – i.e. those that represent i) their partial or full redundancy, or ii) which statements correct which ones – are relations of *equivalence*, *specialization* (which between two statements means that the relation destination represents more information), *implication* (alias "=>"; this is sometimes also a generalization), *correction* (a subtype of `pm:previous-version`) and *exclusion*. For statements, an *exclusion* relation (alias "=>!") means that the premise implies the negation of the conclusion and, at least in classical logics, that the conclusion implies the negation of the premise. As previously justified, gKS requires that the used inference engine knows (e.g., by deduction) whether *each one* of these five primitive relations exists or not between each pair of competing beliefs.

Compliance easiness. As for terms, complying with this requirement is not difficult. One reason is that the protocol can do most of the work, as described in the next paragraphs. Another reason is that the given primitive relations can easily be combined: since some of these four relations imply some of the others or their negations, only one derived relation needs to be manually set between competing beliefs (or none if it is inferrable by the used engine). E.g., in TopOntolForGKS, i) `pm:non-corrective_specialization-only` (alias "`pm:\_`") is defined as subtype of `pm:specialization` as well as an exclusion to `pm:correction` and `pm:implication` (alias "`=>`"), and ii) `pm:corrective-implication-and-generalization` (alias "`pm:c=>_/^`") is defined as subtype of *correction*, `pm:implication` and the inverse of `pm:specialization`. As these examples illustrate, to define the type of a derived relation, one may need *inverse* relations (to get the correct direction for the required relation) and *subtype* relations (to give more precisions). Since the proposed approach is meant to be generic, `pm:implication` is a general type that does not specify a particular logic (thus, allowing the use of any inference engine able to handle the representations in the KB, even if it cannot fully exploit them from a logical completeness viewpoint). A statement requiring a particular logic can still be represented, e.g., unidirectional rules can use a type such as `pm:unidirectional-implication` (alias "`=>>`" in KIF, the Knowledge Interchange Format [16]), and default rules can use a type such as `pm:Statement-not-inconsistent-with-the-rest-of-the-KB` (alias "`consis`" in KIF). Via these last two types, the documentation of KIF represents rules that use the closed world assumption and *default rules* such as the next one: "If something is a bird and if it is not inconsistent that this thing flies, then this thing flies".

KB edition protocol for term additions. When a user of a shared KB submits a statement addition, the protocol can i) transform the statement into a belief (indeed, the user has to log in for making updates and hence is known), ii) most often detect whether the new belief competes with one already in the KB, and then iii) *suggest a relation to connect these two beliefs*. E.g., if the engine has detected that the new belief contradicts and specializes an already entered one, the protocol can ask whether the user believes that there is a *corrective-specialization* relation from this other belief to the new one. Indeed, in the classic case where the two beliefs are formally represented, the inference engine exploited by the protocol *can often* detect when the beliefs are partially/fully redundant – e.g., detect that "at least three healthy birds fly" specializes and implies that "at least two birds fly" – or contradictory. More precisely, the used inference engine *can* make this detection when the KB has the necessary knowledge and when this engine has the necessary inferencing capabilities. In the case where some statements are not formally represented, some tools such as ChatGPT-4 can (as verified by the article author) i) correctly translate a natural language sentence into a formal representation (in a common notation; without using a particular ontology but relating the used terms to those defined in the KB can also be done semi-automatically), and conversely, and ii) detect whether two sentences have an equivalence, specialization or contradiction relationship.

KB edition protocol for a non-additive belief update. When necessary, for its knowledge removal part, such an update also leads to belief cloning (and possibly term cloning) in the way implied by the above given two general requirements.

II.E Evaluation of the Above Analysis and of Belief Trustability

Completeness of the above analysis (and hence of the ontology – or generic model – required by this approach).

- Terms, definitions and beliefs form a partition (i.e. a complete set of exclusive types) for the KB objects. For example, when a KRL object is not a first-order entity in the KB (e.g. a quantifier when used in a statement), i.e., when it cannot itself have relations, it is not a KB object; on the other hand, when it can have relations (e.g. a type for a quantifier such as a type representing the classic universal quantifier, "∀"), it is a term. However, this partition *requires* each universally quantified statement to be represented by its authors as either a belief or a definition. As shown above and below, this distinction – hence, the added precision – is valuable. E.g., the rule "if S is a square then S is a rectangle with 4 equal sides" should be *represented as a definition* for a square. On the other hand, observations (including those expressed by rules) – e.g., "all cars (have been observed to) have 4 wheels" and its rule-based version "if C is a car, then (by observation generalization) C has 4 wheels" should be *represented as beliefs* (the author of which can be the observer). Axioms can always be translated into definitions. E.g., the Euclidian "parallel postulate" can be translated into a definition for the concept type `Euclide:Parallel` or the relation type `Euclide:parallel`; then, statements based on this postulate have to be represented using one of these types. Similarly, if statements use the alethic modality "necessarily true", distinguishing them as either definitions or beliefs is needed. To ease the representation of such precision, TopOntolForGKS defines types such as `pm:universal-quantifier-for-a-definition` (alias `pm:any`, or simply `any` in FCG and Formalized English (FE) [17] KRLs created by the author of this article to ease the representation of

expressive kinds of knowledge), `pm:universal-quantifier-for-a-belief` (alias `pm:each`), `pm:implication-for-a-definition` (alias "`pm:-=>`"), `pm:implication-for-a-belief` (alias "`pm:.=>`"; a `pm:=>` relation between beliefs is implicitly a `pm:.=>` relation), `pm:supertype-for-definition` (alias "`pm:-/^`") and `pm:supertype-for-a-belief` (alias "`pm:./^`").

- The two considered general requirements, as well as the primitive relations for achieving these requirements, are all those which i) when used, enable the avoidance of inconsistencies, or ii) directly represent and organize contradictory and partially or fully redundant objects. More precisions, if required by some rules, can then be given based on these representations.

Evaluation of the trustability of beliefs based on "=>" and "=>!" relations. Any statement – e.g., a correction relation or, equivalently, a statement composed of two statements related by a correction relation – can be *supported* using an implication that has it as conclusion, and, similarly, *contradicted* via a "=>!" relation. When a belief corrects another one, this correction has not been contradicted, the user has not represented a distrust in people such as the author of the correction (e.g. people that do not have an academic degree in ornithology when their statements are about birds), and a choice has to be made between the two beliefs (e.g., for making an inference), a natural default rule seems to be "choose the correcting belief". To allow rules to take into account all supports and contradictions in a logical way, TopOntolForGKS i) relies on the "=>!" and "=>" relations for contradictions and supports (it does *not* propose a `pm:argument` relation or other argumentation relations that are *not* logic-based), and ii) formally and recursively defines the statement types `pm:Successfully-supported_statement`, `pm:Successfully-contradicted_statement` and other types that help define these two types. A statement is "successfully supported", i.e., is of the first type, if i) it has not been "successfully contradicted", each of its supports has not been "successfully contradicted" and is supported by at least one other statement (except for the case a user represents that she likes particular things or prefers certain things over other ones; indeed, it is difficult to give a supporting argument in this case and the assumption is that lying about likings or preferences would generally be more detrimental than beneficial), *or* ii) it cannot be false, i.e., it is a definition. A statement is "successfully contradicted" if it has been contradicted by at least one "successfully supported" statement. Unlike other argumentation-based frameworks [9], this one does not introduce new inference rules and hence remains purely logical, e.g., it does not have rules such as the classic "a statement that has more arguments that a competing statement can be considered as more likely to be true than this last one" (majority based credulous inference [9]). Thus, as a default rule, "choose the correcting belief unless it has been successfully contradicted" seems to be a logical choice and it encourages knowledge providers to add contradictions or supports to beliefs, thus leading other users to reformulate their knowledge and making the KB more and more precise. Nothing prevents a user to create a rule that leads to more dangerous choices and thus to different knowledge inferencing, filtering and display. Since such rules are statements that are not beliefs, they should be *represented as definitions* and are then organized via the specialization relations between the terms they define. Thus, in the shared KB server, they can be retrieved and used by other users than their creators.

Resolution of conflicts (caused by different terminologies, preferences or beliefs) between knowledge providers by leading these agents to inter-relate and precise their representations. This conflict resolution approach is the one underlying the previously described shared KB edition protocol and belief evaluation framework. Since agents are led to give more precisions, and since the different agents represent the same world, the approach enables these agents not to have recurring superficial interactions (via the shared KB). An interaction between two agents committed to fully solving their conflict via the approach *stops when the added precision makes these agents agree that their conflict was caused by* different terminologies (hence definitions and used logics), preferences or fact sources for their beliefs (with, ideally, an agreement on which sources are correct). The interaction may also stop because one of the agents gives up, e.g., upon understanding that a previous mistake has been made and that giving more details will expose that mistake. Unlike in wikis or shared KB servers that do not have editing protocols, edit wars are not possible with the proposed approach.

III Examples of Implementation Methods

Rationale for the references to Web knowledge representation/query/constraint languages advocated by the W3C. The next subsections refer to *Linked Data* related languages because they now are the most well-known and to show that the proposed approach can be implemented with the relatively poorly expressive W3C languages that are referred to below (the W3C also proposes RIF-FLD – the Rule Interchange Format for Logic Dialect – to support more expressiveness). These particular languages of the W3C are relatively poorly expressive because, as argued in the introduction, the languages of Linked Data are mainly meant for Restricted KS, e.g. for the *complete or efficient* exploitation of KRs by applications; furthermore, it is then also easier for the W3C industrial members to comply with the adopted recommendations. E.g., RDF [18] is mainly a structural model for conjunctive existential logic formulas, and almost all KRLs (except those for propositional Logics) enable the representation of such formulas. However, RDF Full allows formula directly on types (i.e., types can have non-predefined relations) while *Description Logics* based KRLs often do not, e.g., most inference engines supporting RDF+OWL – or, more precisely an OWL profile [15] such as OWL-EL, OWL-RL or OWL-QL – do not support RDF Full.

III.A Representations of the Required Kinds of Objects

Object ownership. The lexical approach described in Section II.B works with any KRL, and more complex ownership schemes (e.g. for handling different kinds of authorizations) can be specified with RDF+OWL.

Contexts. A context is a meta-statement representing a condition for the inner statement to be true. Few KRLs allow non-predefined kinds of contexts but many – perhaps most – KRLs, e.g. RDF, allow the reification of (some kinds of) statements and thereby the representation of meta-statements – since creating meta-statements with RDF reification can be very cumbersome, some extensions have been proposed, e.g. "Named graphs" (which are usable via SPARQL [19]) and

RDF-star [20]. If the KRL does not have specific features for meta-statements, they can still be represented in a cumbersome but semantically correct way via the "Context Slices" design pattern [21]. If contexts are only used for representing *believer* relations, and thereby beliefs, this last solution may *not* be too cumbersome. Since few inference engines understand contexts, most engines handle them as normal meta-statements, hence as normal statements. This is not a problem if the KB edition (or I/O) protocol knows the way – or various ways – the beliefs are represented in the KB since it can then perform the necessary checks and queries via functions, sub-queries or constraints (as shown by the next sub-section) and, when necessary, perform some memory management for the inference engine to provide correct results. E.g., assuming that the protocol is implemented by a SPARQL endpoint with an OWL-2 DL entailment regime [22], since OWL-2 inference engines do not handle contexts, one of the ways for the protocol to make such an engine work on beliefs and implement the above-cited occasional "choices between competing beliefs" is to i) store beliefs from different sources in different named graphs, ii) make choices between competing beliefs according to the default rules or the preferences of the user that sent the last search/update SPARQL query to be executed, iii) copy the inner statements of the chosen beliefs to the "default graph" (the one the inference engine works on), iv) directly or indirectly make the inference engine work, v) display the result of the last query, and vi) do the necessary cleaning in the default graph before handling another query. This is only an example for explanation purposes: such memory management (i.e. such knowledge object temporary copying) is not always possible nor necessary depending on the used KB system: alternative *ad hoc* ways can be used, e.g., based on the "Context Slices" design pattern.

The advocated relations and the distinction between definitions and beliefs. Most KRLs are not expressive enough to represent – in a way that is not *ad hoc* – all the types of TopOntolForGKS, but most KRLs are able to declare them (and relate them by specialization relations). This is sufficient for the advocated relations to be used in the KB and taken into account by a protocol that understands their special meanings. Similarly, most KRLs enable the distinction between definitions and beliefs. Although most KRLs would not allow the use of types such as `pm:each` or `pm:implication-for-a-belief`, most KRLs have a special syntax for definitions. RDF+OWL is one of the exceptions: as in most Description Logics, the supertype relations (`rdfs:subClassOf` and `rdfs:subPropertyOf`) and some other relations are used for representing definitions; however, the documents about the semantics of OWL (e.g. [23]) use the classic universal quantifier, "∀", for defining the allowed universal expressions and hence these relations are not restricted to making definitions. If RDF Full is usable, a solution is to consider that if these relations have no believer context, they are definitions. In the common case where RDF Full is not usable, a solution is for knowledge providers to give a type to the OWL expressions involved in making universal statements (hence, more precisely, to give a type to particular "OWL restrictions" using `rdf:type` relations), e.g. the type `pm:Owl-restriction-for-a-belief`). An alternative is, within these expressions, to use a particular relation that the protocol can interpret as signaling that these expressions are about beliefs instead of definitions.

III.B Implementations for the KB Edition (or I/O) Protocol

Hardcoded or user-written functions. A cooperatively-built KB system can have a fully hardcoded I/O protocol, as in WebKB-2 [8]. It can also be mainly implemented by interpreted functions stored in the ontology (like other definitions) and called by the small hardcoded part when search/update queries and knowledge display are performed. Since FCG (alias FL), the main KRL of WebKB-2, now supports function definitions and has primitive functions for searching, updating, displaying knowledge objects and calling the inference engine, the I/O protocol of WebKB-2 is now re-engineered via such interpreted functions to be much more flexible and allow any of its users to write or select extended versions of such functions.

Search/update queries and transformation languages/systems. In some KB servers, e.g. SPARQL endpoints, the user can write queries for searching or updating the KB. Such interpreted queries are generally not as flexible as interpreted functions but they can be sufficient for supporting a KB edition protocol – furthermore, some SPARQL systems also allow their users to write functions that can call SPARQL queries, e.g. Corese [24], an RDF+OWL based SPARQL system which supports an OWL-2 DL entailment regime [22] for the queries. As for functions, the protocol can be hardcoded or not. E.g., when performing the updates, the KB server can execute hardcoded protocol rules, either directly or by enriching the user submitted query for these rules to be executed. In this case, the server can also allow the user to select (via some predefined relations or functions) between different predefined options. Alternatively or additionally, the KB server – or a separate server – can provide the user with a tool that transforms search/update queries into ones that also enforce a KB editing protocol. Then, this tool can allow the user to extend and combine some of the provided functions. A security risk is that some users could modify the generated enriched queries in a way that does not respect the basic rules selected by the owner of the KB server. To generate these enriched queries, transformation languages or systems that exploit knowledge representations can be reused; Corese supports a language to write such transformations on RDF knowledge or SPARQL queries.

Constraints. KB constraints are rules – generally, KB checking rules – that are triggered when the KB is in the process of being updated, and that authorize or not the update. SHACL (Shapes Constraint Language) [25] only allows particular kinds of checking on RDF structures and hence could not be used for implementing a KB editing protocol. However, SPIN (SParql Inferencing Notation) [26] could be used in RDF KB systems that handle this language since it allows the storage and triggering of SPARQL queries. Furthermore, [27] showed how SPARQL queries can be designed for checking constraints expressed in an RDF+OWL based KRL within the checked KB, hence not queries for checking particular constraints but generic queries for checking constraints. [27] illustrates this approach with various constraints, some of which check the existence of relations such as those advocated in Section II.B (and [13] also uses SPARQL to check these relations). The works of [27] and [13] have been experimentally checked in a SPARQL endpoint handled by Corese. Thus, these works could be merged and extended to create SPARQL queries for executing protocol rules expressed in an RDF+OWL based KRL within the KB where these rules should apply.

IV Conclusion

Approaches. The introduction quickly compared various approaches to knowledge sharing (KS), hence to its integration of fragmented knowledge, and the general ideas of an approach supporting *general* KS, thus *not* based on i) particular logics, typically inconsistency-tolerant ones, ii) manual or automatic knowledge selection to eliminate contradictions, and iii) the creation of partially independently developed KBs (e.g., via the manual or semi-automatic selection of knowledge from other KBs), hence KBs that are mutually partially contradictory or redundant, without inter-KB object-relating connections allowing these KBs to be automatically integrated into a single consistent KB or queried as a single KB (e.g. a networked one).

Proposed framework. Section II presented a building block for achieving this *general* KS or integration: two general high-level requirements – regarding object ownership and competing object comparison – to be enforced by a KB editing protocol and their consequences on which kinds of knowledge objects the protocol should be based on (i.e., terms, definitions and beliefs) and how: i) additive modifications via the systematic setting of particular complementary kinds of relations (with TopOntolForGKS including the necessary types to set these relations and thus let default rules or user rules exploit them for knowledge inferencing, filtering and display), and ii) object cloning before destructive modifications, as a complementary way to ensure loss-less knowledge integration. This answered the listed research questions and provided the "fragmented knowledge integration framework" hoped by [1]. As explained and illustrated, this approach also has advantages for searches – and, more generally, inferencing – since, the KB is "complete with respect to the advocated relations" (in the previously given sense) if the KB is sufficiently formal and the protocol fully implemented.

Implementations. Section III showed that most KRLs can represent the required kinds of objects, and that the protocol can be implemented in different ways, hence that most shared KB systems could be extended to adopt the approach, fully or partially. However, as with any knowledge-based system (KBS), the more genericity or flexibility is required, the more the used system must be able to handle knowledge expressiveness and be built from scratch with genericity in mind. This is why WebKB-2 was originally built without reusing an existing KBS and why it has now been re-engineered since it is now aimed to be generic with respect to i) KRLs (via a user-extendable KRL ontology), helper knowledge inferencing systems, and underlying database management systems, and ii) knowledge input-output protocols and knowledge exploitation functions (via its user-extendable ontology of default I/O rules and interpretable functions).

References

1 Union of International Associations (UIA), "The Encyclopedia of World Problems & Human Potential," encyclopedia.uia.org. http://encyclopedia.uia.org/en/problem/fragmentation-knowledge (accessed July 31, 2023).

2 A. Farquhar, R. Fikes, and J. Rice, "The Ontolingua Server: a tool for collaborative ontology construction," International Journal of Human-Computer Studies, Elsevier 1997, Volume 46, Issue 6, pp. 707–727.

3 T.P. Tanon, T. Pellissier, D. Vrandečić, and S. Schaffert, From Freebase to Wikidata: The Great Migration, WWW 2016, 25th International Conference on World Wide Web, pp. 1419–1428.

4 N. Shadbolt, T. Berners-Lee, and W. Hall, The Semantic Web Revisited, IEEE Intelligent Systems, 2006, vol. 21, no. 3, pp. 96-101.

5 World Wide Web Consortium (W3C), "LinkedData," www.w3.org. https://www.w3.org/standards/semanticweb/ (accessed July 31, 2023).

6 J. Euzenat, "Corporate memory through cooperative creation of knowledge bases and hyper-documents," KAW 1996 (36), Canada.

7 Ph.A. Martin, "Top-level Ontology For General Knowledge Sharing," www.webkb.org. http://www.webkb.org/kb/top/d_upperOntology.html (accessed July 31, 2023).

8 Ph.A. Martin, "Collaborative knowledge sharing and editing," International Journal on Computer Science and Information Systems (IJCSIS), Volume 6, Issue 1, pp. 14–29.

9 J. Delobelle, A. Haret, S. Konieczny, J.G Mailly, J. Rossit, and S. Woltran, "Merging of abstract argumentation frameworks," 2016 Principles of Knowledge Representation and Reasoning.

10 M. Kejriwal, C.A. Knoblock, and P. Szekely, Knowledge graphs: Fundamentals, Techniques, and Applications. MIT Press, 2021.

11 A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G.D. Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, and A.C.N. Ngomo, "Knowledge graphs," ACM Computing Surveys, 2021, 54(4).

12 J.F. Sowa,. "Knowledge Representation: Logical, Philosophical, and Computational Foundations," Brooks/Cole Publishing Co., CA., 2000.

13 Ph.A. Martin, O. Corby, and C. Faron Zucker, Ontology Design Rules Based On Comparability Via Certain Relations, Semantics 2019, Germany, LNCS 11702, pp. 198-214.

14 Ph.A. Martin, "The Sub Ontology in Turtle," www.webkb.org. http://www.webkb.org/kb/it/o_KR/p_kEvaluation/ontology/sub/ (accessed July 31, 2023).

15 W3C, "OWL 2 Web Ontology Language Profiles (Second Edition) – W3C Recommendation 11 December 2012", www.w3.org. http://www.w3.org/TR/owl2-profiles/ (accessed July 31, 2023).

16 M.R. Genesereth, and R.E. Fikes. "Knowledge interchange format - version 3.0: reference manual," Technical Report Logic-92-1, Stanford University, CA, USA, 1992.

17 Ph.A. Martin, "Knowledge representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English," ICCS 2002, 10th International Conference on Conceptual Structures, Springer, LNAI 2393, pp. 77-91.

18 World Wide Web Consortium (W3C), "RDF 1.1 Concepts and Abstract Syntax – W3C Recommendation 25 February 2014", www.w3.org. https://www.w3.org/TR/rdf11-concepts/ (accessed July 31, 2023).

19 W3C, "SPARQL 1.1 Query Language – W3C Recommendation 21 March 2013," www.w3.org. https://www.w3.org/TR/sparql11-query/ (accessed July 31, 2023).

20 W3C, "RDF-star and SPARQL-star." Final Community Group Report – Final Community Group Report 17 December 2021," www.w3.org. https://www.w3.org/2021/12/rdf-star.html (accessed July 31, 2023).

21 C. Welty, "Context Slices," ontologydesignpatterns.org (2010). http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices (accessed July 31, 2023).

22 W3C, "SPARQL 1.1 Entailment Regimes – W3C Recommendation 21 March 2013," www.w3.org. http://www.w3.org/TR/sparql11-entailment/ (accessed July 31, 2023).

23 W3C, "OWL 2 Web Ontology Language Direct Semantics – W3C Recommendation 11 December 2012," www.w3.org. https://www.w3.org/TR/owl2-direct-semantics/ (accessed July 31, 2023).

24 O. Corby, and C. Faron-Zucker, "STTL: A SPARQL-based Transformation Language for RDF," WEBIST 2015, 11th International Conference on Web Information Systems and Technologies, Portugal.

25 W3C, "Shapes Constraint Language (SHACL) – W3C Recommendation 20 July 2017," www.w3.org. https://www.w3.org/TR/shacl/ (accessed July 31, 2023).

26 W3C, "SPIN - Modeling Vocabulary – W3C Member Submission 22 February 2011," www.w3.org. https://www.w3.org/Submission/spin-modeling/ (accessed July 31, 2023).

27 Ph.A. Martin, Evaluating Ontology Completeness via SPARQL and Relations-between-classes based Constraints," IEEE QUATIC 2018, 11th International Conference on the Quality of Information and Communications Technology, pp. 255–263, Coimbra, Portugal.