CoSMo: A multilingual modular language for Content Selection Modelling

Kutz Arrieta
Independent scholar
Sunnyvale, USA
kutzaki@gmail.com

Pablo R. Fillottrani
Universidad Nacional del Sur
Bahía Blanca, Argentina
prf@cs.uns.edu.ar

C. Maria Keet
University of Cape Town
Cape Town, South Africa
mkeet@cs.uct.ac.za

ABSTRACT

Representing snippets of information abstractly is a task that needs to be performed for various purposes, such as database view specification and the first stage in the natural language generation pipeline for generative AI from structured input, i.e., the content selection stage to determine what needs to be verbalised. For the Abstract Wikipedia project, requirements analysis revealed that such an abstract representation requires multilingual modelling, content selection covering declarative content and functions, and both classes and instances. There is no modelling language that meets either of the three features, let alone a combination. Following a rigorous language design process inclusive of broad stakeholder consultation, we created CoSMo, a novel Content Selection Modeling language that meets these and other requirements so that it may be useful both in Abstract Wikipedia as well as other contexts. We describe the design process, rationale and choices, the specification, and preliminary evaluation of the language.

CCS CONCEPTS

• **Computing methodologies** → **Natural language generation**; Semantic networks; • **Information systems** → *Database design and models.*

KEYWORDS

Modeling Language, Query Language, Wikidata, Multilingualism

ACM Reference Format:
Kutz Arrieta, Pablo R. Fillottrani, and C. Maria Keet. 2024. CoSMo: A multilingual modular language for Content Selection Modelling. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3605098.3635889

1 INTRODUCTION

Databases technologies have been evolving from RDBMS silos to also comprise RDF triple stores that can be queried on the Web more easily as Linked Data accessible through a SPARQL endpoint, and knowledge graphs [14]. Such data access influenced conceptual modelling, notably with ontology-based data access techniques (e.g., [9, 28]), where the model serves the dual purpose of abstract representation of the data store's content and as starting point for query construction. One such publicly editable RDF data store is Wikidata, which will be used in the new Abstract Wikipedia project [26, 27]. Abstract Wikipedia aims to be a 'next generation' truly multilingual Wikipedia, where articles are generated from data stored in Wikidata and functions from Wikifunctions in at least all supported languages of the Wikipedia ecosystem. This requires an open, publicly editable multilingual natural language generation (NLG) systems for verbalising structured data, information, and knowledge that is selected from Wikidata and Wikifunctions.

Such a system is currently under development and related research is being conducted, such as verbalising all RDF statements in a selected topic into basic English [1] and ad hoc content selection that focuses on multiple languages instead [22]. What is supposed to be positioned there, is what the community refers to as "constructors" and the "abstract representation" [26], for which no proposal exist yet, only an exploration of what it may, or may not, involve[1]. The selection of content from Wikidata, and possible processing with functions from Wikifunctions, is expected to be done by end users, in snippets or small modules (the basic 'constructors') at a time, where the constructors can be put together to automatically generate larger pieces of text to make up an article on a topic in the desired language, and declaring such constructors would be done in the user's preferred natural language. This content selection step is suggestive of being alike creating a database view, but with a few extras, notably calculating something, such as the age based on date of birth. That is, it needs at least a 'view creation' syntax for RDF, alike SQL's `CREATE VIEW`, but in such a way that it is implementation-independent, since the Wikidata database may be converted to JSON or moved into an RDBMS for performance at a later stage and one would not want to have to update all user-defined constructors because of it. Alternatively, such a prospective constructor language can be seen as one aimed at information modelling.

There is no modelling language that meets these requirements, nor is there, to the best of our knowledge, a standard for content selection in the NLG systems processes. We propose a new modelling language for the constructors, i.e., the content selection specification step, which can be used with Abstract Wikipedia and other content selection tasks in NLG to verbalise structured data, information, and knowledge, as well as in other tasks where users need to select a fragment of a model. The key features and novelties compared to other conceptual data modelling and view specification languages are:

[1] https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Wikidata_Abstract_Representation; version of 29 Nov 2022.

SAC '24, April 8–12, 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/10.1145/3605098.3635889

- Multilinguality is embedded in the modelling language;
- A fully modular approach, where modules (as so-called 'constructors') are first-class citizens in the language;
- Declarations can be made at both class and instance level in the same language;
- Declarations can contain both static content and functions to compute content that can be derived from the static content.

We followed a rigorous language design process that was introduced by [6, 8], with this being its first industry-initiated use case. We describe the design process, rationale and choices made, the specification of the language—called CoSMo, Content Selection Modeling language—and a preliminary evaluation. The evaluation showed that it meets these and other requirements so that it may be useful both in Abstract Wikipedia as well as other contexts. More examples, the BNF grammar, and the shorthand notation are available as technical report [2].

The remainder of the paper is structured as follows. Section 2 provides background on NLG, Abstract Wikipedia, and the notion of modelling language design. Section 3 describes the process followed, design principles chosen, and the specification of CoSMo. Section 4 reports on the evaluation of CoSMo. We discuss and compare it to related work in Section 5 and conclude in Section 6.

2 BACKGROUND

The related work consists of three converging themes: features and limitations of current conceptual modelling languages, procedures for the design of modelling languages, and NLG in general and for Abstract Wikipedia in particular. We defer comparing modelling and selection languages against the features of CoSMo, that we will propose in the next section, to the discussion in Section 5.

There are many ways to generate natural language from structured input, which is typically either data, such as from spreadsheets and databases, or structured information, such as verbalising conceptual data models, or ontologies. What they all have in common is a pipeline approach. The 'reference pipeline' [23] is illustrative of the kinds of tasks that need to be decided upon, which may practically entail additional sub-pipelines and loops back to an earlier stage (see [19] for an overview). The steps are: A) text planning with (1) content determination and (2) discourse planning; B) sentence planning with (3) aggregation, (4) lexicalisation, and (5) referring expression generation; and C) linguistic realisation involving (6) grammar and orthography rules to produce correct sentences.

Within the context of Abstract Wikipedia, the input is to be extracted from Wikidata, the computations are expected to be done on Wikifunctions, and the output is to be a Wikipedia article [27]. Wikidata is an RDF triple store with over 14 billion triples. Selecting content from it to create an article is, in general idea, akin to declaring a view over a small part of the RDF triple store, but then with the additional option to declare functions, such as computing a person's age from their date of birth that is stored in Wikidata. Currently, there is no way to do so other than writing individual SPARQL queries at its SPARQL endpoint. It lacks a way of declaring what Abstract Wikipedia calls the 'constructors'. The two proof-of-concept tools, Ninai/Udiron [20] and the one with Scribunto, use a temporary mechanism, where the former merges data structures from across the pipeline whilst it resolves query answering through the SPARQL endpoint for the Wikidata content and lexeme data, and the latter has ad hoc constructors to focus on the realiser[2]. Also Grammatical Framework is being explored, but it focuses on multilingual surface realisation and also has no solution for the content selection step (section 8 in [22]). Thus, the critical steps of systematic, robust, multilingual, and end-user accessible content selection and ordering is missing from the overall Abstract Wikipedia pipeline to date, and thus also where something needs to be chosen from the design options.

It is that language for the constructors in step 1 that we aim to design, which entails a notion of language design. This requires a design procedure, which has been proposed recently in [8] and adapted for conceptual models in [6]. We avail of this here, adapted to the, to the best of our knowledge, first use case of that proposal, as shown in Fig. 1, and use it for a different type of modelling language.

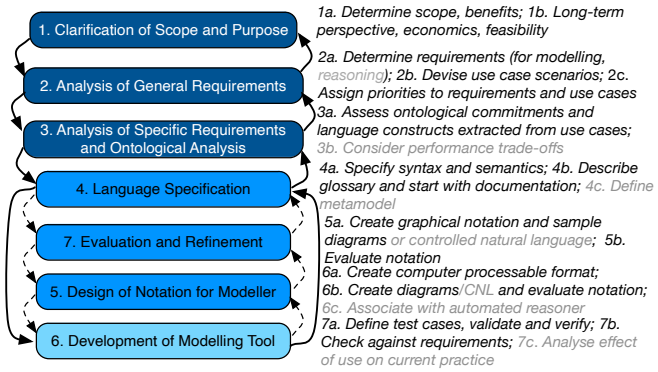


Figure 1: Development process of [6] adapted to the constructor language design case as follows: dark blue: wide consultative process; medium blue: language creation by the authors; light blue: planned; grey text: task not used because deemed not needed.

3 LANGUAGE DESIGN

We demarcate the prospective language to be intended only for step 1 of the reference pipeline: content selection. It leaves open the option whether modules in a later step of the NLG pipeline may devise a strict extension for discourse planning and other language information, but this is beyond the scope of this paper. We first describe the procedure followed and then the specification.

3.1 Requirements elicitation (steps 1-3)

The methodological approach followed is overlaid over the diagram in Fig. 1. For the first three steps in the development process, several stakeholder meetings were held online on Google Meet in September and October 2022. An initial design document was drafted by Arrieta to which other members contributed and which was discussed and extended in the meetings and over email. These

[2]Ninai/Udiron by Mahir Morshed: https://gitlab.com/mahir256/ninai/-/tree/main/ninai/constructors; Scribunto-based one by Ariel Gutman: https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Template_Language_for_Wikifunctions/Scribunto-based_implementation.

meetings comprised both theory-based approaches and example-based approaches. This was subsequently reworked into two documents, one for public consumption and discussion on the Abstract Wikipedia metawiki[3] and one internal document with 'leftovers'. Around the same time, Keet created a list of possible requirements emanating from the examples and additional examples were devised, and two 'exploration through implementation' approaches were pursued, which are the aforementioned Ninai/Udiron system and the Scribunto-based proof-of-concept. An open meeting was held on Jitsi on 21 Feb 2023 to explain the features list to clarify terminology across specialists from the different disciplines and obtain more feedback on preferences. This meeting was the 'cut-off point' to proceed to select the required and desired parameters, goals, and paradigms, and proceed to step 4.

3.2 Step 4: Language Specification

Broad *a priori* parameters exist for the Abstract Wikipedia project, which can be recast as goals and high-level requirements that the 'abstract representation language' has to meet. Notably, they include adherence to Wikimedia Foundation founding principles and guidelines, an open source licence, availability and editability in all languages of the Wikimedia projects, and with, ultimately, the local Wikipedia communities being in control of the content[4]. These non-language goals have a partial overlap with the original aims for OWL [15], but the emphasis on multilinguality and community, in the sense of anyone (not just the modeller or logician), is a mandatory requirement rather than a goal to aspire. It has, to the best of our knowledge, no overlap with any broad parameters for either of the conceptual modelling or query languages either.

3.2.1 Design principles. Language feature requirements can be separated into principles that have to be met, i.e., on the 'what' that has to be in the language, and the 'how', or requirements on how that will be incorporated in a language. For instance, one may require roles (argument places), but there may be different ways to incorporate them, such as unnamed with an automatic counter for identifier or requiring a human-readable name. A same list of principles may thus still result in different modelling languages. Nonetheless, they help setting tighter bounds. Based on the Abstract Wikipedia meetings, presentations, discussions, proof-of-concept tools, feature comparison document, and additional examples, the principles chosen are as follows:

- Roles of relationships, not the Entity-Attribute-Value style. This aligns well with existing information modelling languages as well as the language and linguistic view on constructors where roles are used heavily for predicates and verbs, such as in VerbNet [21] and FrameNet [3].
- Separation of the content from the natural language rendering aspects, i.e., to aim for not having linguistic markers in the constructors.
- For a first version of the language at least: any noise in the data is expected to be corrected in the source rather than building verification mechanisms into the modelling language. This deviates from the aim of constraints in conceptual data modelling languages (CDMLs), which reflects its position in the pipeline, it being after adding data to the database rather than before, and that modifying Wikidata is open to anyone.
- Vocabulary creation in constructors is prohibited; rather, Wikidata L (lexicographic), Q (object [type]), or P (property) items are to be used and if it is not in Wikidata yet, then the user must add it there first. This enables its multilinguality and prevents universe of discourse semantic mismatches and costly mapping assertions.
- The language must permit both declarative content and functions, where a constructor may have either or both sort of content. This bears a resemblance to UML class diagram's attributes and methods, but then realised differently (e.g., function execution is assumed to happen on Wikifunctions).
- The representation should be the least perspectivist possible (to allow representing several points of view on content, such as class vs attribute or property), so that fewer constructors need to be specified that are more versatile in their use in the NLG pipeline.
- To facilitate the multilinguality of the language yet provide stability to the language specification, there must be a 1:1 mapping on reserved strings in the language specification to Wikidata items or lexemes. The key benefit of the 1:1 mapping is that then the language is *de facto* multilingual, because it can fetch the label in one's desired language from Wikidata. This ensures that if an item is deleted in Wikidata, then the language will not be broken as knock-on effect.
- Constructors can be reused and combined in other constructors and they can be defined for either particular instances or at the type-level where then all the Wikidata instances that satisfy it are retrieved; hence, multiple articles or paragraphs or sentences may be created at once or a value can be set for a one rendering such that only one item will satisfy it.
- Constructors can be refined or generalised, in that one can take an existing constructor that a contributor declared and add or remove features.

The last one on constructor size, reuse, and instance versus type level caused confusion in the meetings, since many wanted type level constructors but typically started with instance-level examples and declarations. After much deliberation as to the constructor size, it was concluded that there is no unambiguous way to demarcate it; size will depend on the amount of generalisation and compositionality for reusability. This is related to the issue of whether to add a reserved wrapper element like, say, 'article': what may end up as an article in one language may be only a paragraph of an article in another language. Different stances might be taken to improve readability or tolerate NLG systems that cannot yet compute sentence aggregation and referring expressions. In addition, when it comes to realisation in a specific language, further down in the pipeline, the same constructor might be realized in one word (in highly agglutinative languages), a compound, a phrase, a long sentence or a sequence of sentences. This will also depend on the maturity of the realizer. A version 1 may see a more telegraphic style realizer, which facilitates generation while preserving content.

[3] https://meta.wikimedia.org/wiki/Special:MyLanguage/Abstract_Wikipedia/Wikidata_Abstract_Representation, d.d. 29 Nov 2022.
[4] See https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Requirements.

What was not extensively discussed during the meetings, is whether the language should be ontology-driven, in part due to aversion to the idea by some and it seems to be a general tendency within the Wiki projects as it being too prescriptivist. An ontology-driven way can take several forms. For instance, it may be alike OntoUML where each class is stereotyped with an entity from the foundational ontology UFO [12], at the level of the language itself by embedding ontological commitments into it [8], or an optional alignment to a foundational ontology category. Either way, o Ontological commitments cannot be avoided for the language itself. The design principle and requirement of roles mean a so-called *positionalist* stance on the nature of relations rather than "standard view" or "anti-positionalist" [10]. The separation of the subject domain semantics from the language features also has implications for how one assumes the world to be.

Other features are either more contentious or less certain about possible ramifications or level of desirability. They can be designed as strict extensions to a base language, alike with the design of OWL and OWL 2. Here, a lean language that is simple to use is preferred, since community adoption is aimed for. If a more experienced constructor author wants to add a more precise specification, they should be able to, which may be offered in an 'advanced' cf. a 'standard' interface for the language that meets the goals listed above; specifically:

- Mandatory participation of an element. This imposes the requirement that both that data must be available in Wikidata and there must be NLG algorithms in the target language to render at least that part of the constructor.
- Union of Q or P items. It could be argued that duplication should be removed from Wikidata and therefore this would not be needed, or it could be achieved by merging two otherwise the same constructors. The former may not be easy to effectuate and for the latter, the union operator amounts to a simple syntactic sugar to simplify constructor creation.

Negation did not appear in any of the examples and thus made it in neither the principal nor the extended list. It may be an unexplored omission, as it is certainly not a conscious 'no'.

3.2.2 The language: specification and notation for modellers. Practically, this design process proceeded in iterations between steps 4 and 5, and then finalised with an iteration between steps 5 and 7 of the procedure.

CoSMo *syntax.* The basic elements are listed in Table 1. The graphical icons were selected as inspired by ORM2 notation whenever possible, except for two constructor icons, one for types and one for instances, that are absent from ORM2. In addition to the Wikidata and Wikifunctions items, we can observe there is only the possibility to define new constructor types and objects. The third column provides some details on how these elements can be used.

Table 2 shows connectors between two of the basic elements in Table 1. Again, graphical icons from ORM2 and UML were selected when possible. The only new icon is the trident icon linking a new type or instance to its definition. This definition must be given in basis a relationship participation, and is done by means of local variables. In order to foster simplicity of the language, we define the scope of these variables to be local to the current model. Global variables like in RDF would increase the reusability of the models, but this is not a current requirement for the language.

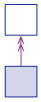
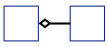
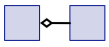
Table 1: CoSMo basic elements. The longform notation displays the English labels for readability, where the respective Wikidata items are listed in Table 5. The formal syntax is given in additional material.

Element Icon	Longform representation	Usage
	`Object(Q) or ObjectType(Q)`	Typically a Q (possibly a P) object (i.e., instance) or object type (i.e., class)
	`Function(Z(...))`	$n-ary$ function to compute something that is not an NLG-related function (e.g., age), Z 'object'. The function arguments have the representation given in the next row.
	`Z(Q1,...,Qn)`	Notation for n-ary function arguments; here, the $n-1$ first elements correspond to the arguments for the function call while the last compartment connects to the function icon (indicated by the dashed connection, as usual in ORM)
	`Property(P(Role1, Role2))`	Typically a P (possibly Q) item, i.e., a property that relates (at least) two things.
	`TypeConstructor:T()`	Constructor (at the type level); must be used in conjunction with at least one definition. T is a local variable
	`InstanceConstructor:O()`	Constructor (at the instance level); must be used in conjunction with at least one definition. O is a local variable

Table 3 presents the constraint representations of CoSMo: value constraints, role names, join relationship, role mandatory participation and instantiation. Actually, role names are not constraints but just syntactic sugar that later may be useful in further steps in the NLG pipeline.

CoSMo *semantics.* We formalize the meaning of the textual representation in first order logic. Our language is formed with one constant q, p, z for each P, Q, Z items referenced in Wikipedia and Wikifunctions; one unary predicate for each Q item, and each object variable, object type variable, and role variable; one binary predicate for each P item; and one n-ary function for each n-ary Z item. We also include the special predicates `QItem()`, `PItem()`, `ZItem()`, `Has()`, `PPartOf()` and `Contains()`. The first three are unary predicates for each type of Wiki item. `Has()` is a set of predicates to represent the function application to a given set of arguments. `PPartOf()` is a binary predicate formalising the proper parthood relation (assuming ground mereology [25]), which is transitive, irreflexive, and asymmetric. `Contains()` is quaternary predicate relating an object in the domain with the predicate name and predicate instances it reifies. Remember that the only possibility of introducing a new element is by defining it as a particular predicate instantiation. The semantic mapping is shown in Table 4. In a given model definition, we can have object and type constructor, subconstructor, and instance definitions. Within the type constructor we can have at least one predicate declaration, two or more role declarations, and zero or more function declarations. Role declaration may include object and object type declaration, with optional role names and value constraints. The object constructor has the same structure as the type constructor, with the addition of at least one instantiation declaration. A CoSMo interpretation is a tuple $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}}, ob, name \rangle$. We assume a countable infinite domain Δ, which includes all P, Q, Z items referred in Wikipedia and Wikifunctions, which we call P, Q and Z. Thus, $P \cup Q \cup Z \subseteq \Delta$. There is also in $\mathcal{I}$ a unique object assigned to each tuple by the injective function $ob : P \times Q \times Q \rightarrow \Delta$

Table 2: Connector vocabulary of CoSMo. The longform notation displays the English labels for readability; the respective Wikidata items are listed in Table 5.

Connector Icon	Longform representation	Usage
	`SubConstructorOf(T1, T2)`	Subtype constructor between local variables
	`InstanceOf(O, T)`	Instance constructor between local variables
	`role:Object(Q)` or `role:ObjectType(Q)` or `Function(Z)`	Connector/link to connect a Q to a P, or a Z to a list of arguments
	`PartOf(T1,T2)`	Connector for a container type `T1` to a contained type `T2`, both as local variables.
	`PartOf(O1,O2)`	Connectorfor a container object `O1` to a contained object `O2`, both as local variables.
	`TypeConstructor: T(`*definition*`)`	Connector to define a new type as a participation in a relationship; *definition* contains relevant predicate and role declaration.
	`InstanceConstructor: O(`*definition*`)`	Connector to define a new object as a participation in a relationship; *definition* contains relevant predicate, role and instantiation declarations.

assigning a unique domain element which objectifies each predicate tuple, and the interpretation function $\cdot^{\mathcal{I}}$ is defined as usual in predicate logic, with the additional following axioms:

$$QItem(q)^{\mathcal{I}} \equiv q^{\mathcal{I}} \in Q^{\mathcal{I}} \qquad Q_1(q_2)^{\mathcal{I}} \equiv q_2^{\mathcal{I}} \in Q_1^{\mathcal{I}}$$
$$PItem(p)^{\mathcal{I}} \equiv p^{\mathcal{I}} \in P^{\mathcal{I}} \qquad P(x,y)^{\mathcal{I}} \equiv (x^{\mathcal{I}}, y^{\mathcal{I}}) \in P^{\mathcal{I}}$$
$$ZItem(z)^{\mathcal{I}} \equiv z^{\mathcal{I}} \in Z^{\mathcal{I}} \qquad Role(x)^{\mathcal{I}} \equiv x^{\mathcal{I}} \in Role^{\mathcal{I}}$$
$$Has(f,x,...,x,r)^{\mathcal{I}} \equiv r^{\mathcal{I}} = f^{\mathcal{I}}(x^{\mathcal{I}},\ldots,x^{\mathcal{I}})$$
$$Contains(x,p,y_1,y_2)^{\mathcal{I}} \equiv ob(p^{\mathcal{I}}, y_1^{\mathcal{I}}, y_2^{\mathcal{I}}) = x^{\mathcal{I}}$$

Then, all interpretations that satisfy a given CoSMo model definition are models for that definition.

4 EVALUATION

The main aim of the evaluation is to test the expressiveness of the language: can it do what it should do? To examine this, we check CoSMo against the requirements and design decisions and revisit the examples that were used to devise the set of language requirements to determine whether they can be represented in the new modelling language. Due to space limitations, we focus on an illustrations and revisiting the requirements; see the supplementary material [2] for more examples.

Table 3: Constructor language adornment vocabulary of CoSMo.

Adornment Icon	Longform representation	Usage
{ ... }	`definition{ValueConstraint}`	Value constraint for a type, an object or a function definition. Constraints are expressed as in ORM2
[...]	`Role[Name]:Object(Q)` or `Role[Name]:ObjectType(Q)`	Defines the role as in the declaration, assigning a name to the participation. The definition of object or object type as in the first row of table 1.
	`Join(Q1, Q2)` or `Join(P1, P2)`	Merge/join between two objects or two relations of the same arity, within the definition of a type or an object.
	`IsMandatory(Role)`	Mandatory participation of the representation of the element for the constructor to be allowed to be realised, within the definition of a type or an object.
	`ObjectType(Q1)={Q2}`	Instantiation constraint for the characterisation of an instance, relating one object QItem to another QItem representing a type, within the definition of a type or an object.

Table 4: Semantics of CoSMo as first order logic formulae.

Longform	FOL representation
`Object(Q)`	$\texttt{QItem}(q)$
`ObjectType(Q)`	$\texttt{QItem}(q)$
`ObjectType(Q1)=Q2`	$\texttt{QItem}(q_1) \wedge \texttt{QItem}(q_2) \wedge Q_1(q_2)$
`Property(P(Role1,Role2))`	$\texttt{PItem}(p) \wedge \forall x, y(P(x,y) \rightarrow Role1(x) \wedge Role2(y))$
`Role:ObjectType(Q)`	$\forall x(Rol(x) \rightarrow Q(x))$
`Function(Z(Q1,...,Q2))`	$\texttt{Zitem}(z) \wedge \texttt{Has}(z, q_1, \ldots, q_n, Z(o))$
`SubConstructorOf(Q1,Q2)`	$\forall x(Q_1(x) \rightarrow Q_2(x))$
`InstanceOf(O,T)`	$\forall x(O(x) \rightarrow T(x)$
`PartOf(T1,T2)`	$\forall x(T1(x) \rightarrow \exists y(T2(y) \wedge \texttt{PPartOf}(x,y)))$
`TypeConstructor:T(`definition on p`)`	$\forall x(T(x) \rightarrow (\exists y_1, y_2 \texttt{Contains}(x,p,y_1,y_2) \wedge$ definition on tuple $p(y_1,y_2))$
`InstanceConstructor:O(`definition on p`)`	$\forall x O(x) \rightarrow (\exists y_1, y_2 \texttt{Contains}(x,p,y_1,y_2) \wedge$ definition on tuple $p(y_1,y_2))$
`Join(Q1,Q2)`	$\forall x(Q(x) \leftrightarrow (Q1(x) \vee Q2(x)))$
`Join(P1,P2)`	$\forall x, y(P(x,y) \leftrightarrow (P1(x,y) \vee P2(x,y)))$

4.1 Example constructors

The first example from the Abstract Representation discussion document on the metawiki, about Edith Eger, may be modelled as follows. First, Edith Eger is a child of two parents, for which we create a type-level constructor (C1) and instantiate it where Edith Eger (Q62070381 in Wikidata), in C2, which is shown in the top-half of Fig. 2. Subtyping of a constructor is illustrated with C3 being a subconstructor of C1, since it has an additional Age function. The instantiation and subtyping may also be drawn in different figures. The longform serialisation of the diagrams, or straight written in text format, can be as follows, rendered in two different natural languages in accordance with Table 5.

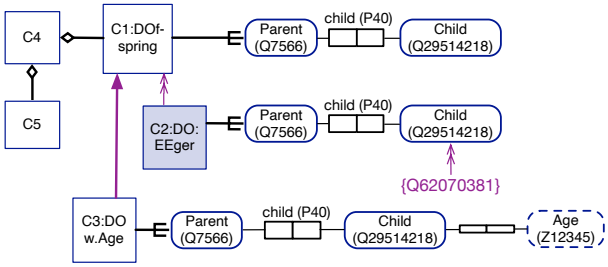


Figure 2: Type-level constructors C1, C3-C5 and instance-level constructor C2. Names are placeholders (content of C4, C5 suppressed), which are expected to be assigned an identifier in its own namespace and with at least one label.

Longform in English

```
TypeConstructor:C1(
   Property(P40(r1,r2)),
   r1:ObjectType(Q7566),
   r2:ObjectType(Q29514218))

InstanceOf(C2, C1)

InstanceConstructor:C2(
   Property(P40(r1,r2)),
   r1:ObjectType(Q7566),
   r2:ObjectType(Q29514218),
  ObjectType(Q29514218)={Q62070381})

SubConstructorOf(C3, C1)

TypeConstructor:C3(
   Property(P40(r1,r2)),
   r1:ObjectType(Q7566),
   r2:ObjectType(Q29514218),
   Function(Z12345(Q29514218)))
```

Longform in Spanish

```
ConstructorDeTipo:C1(
   Propriedad(P40(r1,r2)),
   r1:TipoDeEntidad(Q7566),
   r2:TipoDeEntidad(Q29514218))

InstanciaDe(C2, C1)

ConstructorDeInstancia:C2(
   Propriedad(P40(r1,r2)),
   r1:TipoDeEntidad(Q7566),
   r2:TipoDeEntidad(Q29514218),
  TipoDeEntitdad(Q29514218)={Q62070381})

SubConstructorDe(C3, C1)

ConstructorDeTipo:C3(
   Propriedad(P40(r1,r2)),
   r1:TipoDeEntidad(Q7566),
   r2:TipoDeEntidad(Q29514218),
   Funcion(Z12345(Q29514218)))
```

At the back-end, a language-independent CoSMo constructor is stored as:

```
CSM0007:C3(CSM003(P40(r1,r2)), CSM004:CSM002(Q7566), CSM004:CSM002(Q29514218),
   CSM005(Z12345(Q29514218)))
```

Besides the longform notation, a shorthand notation is specified as well and it is possible to name both roles, which, at this stage of modelling, are 'ontological' (non-linguistic) roles; e.g., r2 in C1 may be named with Q239526 (offspring/descendencia/avkomma/etc.).

4.2 Meeting the requirements

We now revisit the requirements and design principles to determine if, and if so where, they have been met, roughly in the order as listed in Section 3.2.1.

First, roles are present and first-class citizens in CoSMo. As is customary practice in conceptual data modelling, they may be named with subject domain-specific names (a Q item from Wikidata), but need not [16]. There are no linguistic markers in the CoSMo specification, nor content verification checking, and by design, one cannot create new vocabulary *in* the constructor. The constructor itself needs an identifier and at least one label to ensure its multilinguality as well, which requires an adaptation of Wikidata, in line with the ongoing deliberations on constructors[5]. The mapping of the language elements to Wiki's Q/P items to ensure multilinguality is listed in Table 5. At present, it is being deliberated in the community what the best implementation strategy would be: to either reuse the indicated tentative Q and P items and add the remainder of items and their respective labels, or to add new Q and P items specifically for CoSMo elements. For this reason, as well as possible Wikidata-independence to increase CoSMo's potential for wider application use and for languages currently not yet supported by Wikidata, CoSMo elements have their own identifiers that are eventually mapped to Wikidata items.

[5]https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Updates/2023-03-15

Table 5: Constructor language vocabulary of CoSMo, with identifiers and labels in three languages. The Q and P items are assumed to become at least *de facto* immutable ('locked') in the same way as other key elements in Wikidata and Wikifunctions. TBA: item to be added to Wikidata; text in italics: item label to be added to Wikidata.

CoSMo ID	*Tentative* Q/P item in Wikidata	Feature (English / Spanish / Basque label)
CSM001	Q35120 ??	Object / Entidad / Izaki
CSM002	(TBA)	*Object type* / *Tipo de entidad* / *izaki mota*
CSM003	Q18616576	Property /Propiedad / propietatea
CSM004	Q117747915 ??	Role / *Rol* / *Rol*
CSM005	Q11348	Function / Función / Funtzio
CSM006	Q1049476 ??	*Instance Constructor* / *Constructor de instancia* / *instantzia eraikitzaile*
CSM007	(TBA)	*Type constructor* / *Constructor de tipo* / *tipo eraikitzaile*
CSM008	(TBA)	*Subconstructor of* / *Subconstructor de* / *honako berreraikitzaile*
CSM009	Property:P31	InstanceOf / Instancia de / honako hau da
CSM010	Property:P527	Part of / *Parte de* / Osatuta
CSM011	Q17853087	Join / Unión / *bildura*
CSM012	Q29557567	*IsMandatory* / *Es obligatorio* / *nahitaezko*
CSM013	Q82799	Name / Nombre / Izen
CSM014	Q42750658	Value constraint Restricción de valor / *balio murrizte*

Also, CoSMo has the requisite elements for both declarative components, being any P and Q item, and for functions, by incorporating the Z identifiers from Wikifunctions. It does this in such a way that it need not be perspectivist for the declarative components: they are all elements on par in an arbitrary, re-orderable sequence. The functions are of something, and do need to relate to something, which can be denoted as if on par in the long notation, and in the shorthand notation with the dot-notation, it is tied to a Q item. Further functionality includes the specialisation/generalisation and instantiation of constructors, each indicated with a different type of arrow in the diagrammatic notation and an additional shaded box. Instead of overloading line notation, we have added an additional line type to connect constructors to an element of its content.

Finally, the two 'extras' were added as well: mandatory participation of an element is indicated with the purple dot and a union of items with the open circle/`Join` element.

5 DISCUSSION

This paper presented a novel modelling language for content selection from databases, be they relational or triple stores, and inclusive of functions, therewith going beyond database view specifications. Yet, while modelling information, it is also distinct from conceptual data modelling languages for it does not strictly require data integrity constraint specification. In addition, one may argue there are additional requirements 'for the 21st century', such as the expected use in a setting where there is a stronger demand on multilinguality, openness, collaboration, and sharing. There are obviously also things it cannot do, by design. We will first compare CoSMo to related work and then elaborate on the broader setting of the content selection language.

5.1 Comparison to related work

To the best of our knowledge, there is no standard way for content selection in NLG, let alone it being end-user usable. There is also no multilingual modular conceptual modelling language or graphical query builder, as can be seen from comparison included in Table 6. Two conceptual data modelling languages are included for illustrative purpose and we also include three visual SPARQL query builders, since in a way CoSMo will be capable to function as such as well, albeit also with functions that is not part of SPARQL query building. The three graphical query builders included are VSB[6], the RDF explorer [24] that also has a web-based interface[7], and the early web-based Wonder system [5] whose graphical rendering was inspired by ORM notation.

Neither is multilingual nor are they modular. The new multilingual TexToData [4] is multilingual in so far as that it takes text in any natural language, performs a machine translation to English, creates the model, and then back-translates, but is not inherently multilingual. Regarding modules, while the query builders are not modular by design and it has not been pursued beyond saving queries [5], there are extensions for modularity of conceptual data models. Such modularity is either geared toward scalability of the graphical notations (see [17] for a recent overview) or as conceptual model interoperability scenario [7] where the models-to-integrate are recast as modules. Therefore, it received a "±" in the comparison. Further, UML class diagrams do not have instances, but the models can link to object models, and ORM permits derivation rules as a sort of functions and joins over fact types (but not entity types), earning them a "±" each for those features.

It is possible to arrive at a different abstract representation language, be it based on a different set of requirements or ways to 'shape' the elements. The diagrammatic notation aimed to reuse as much as possible an existing notation with open source code (ORM2 [13] in this case) to ease implementation. With the logic-based reconstruction, one is free to devise different textual and diagrammatic notations. Either way, any tooling implementation requires rendering all the item identifiers by their respective label.

[6]https://leipert.github.io/vsb/
[7]https://www.rdfexplorer.org/

Table 6: Comparison of CoSMo to UML Class Diagrams and ORM and representative graphical SPARQL query builders; –: no; ±: partial or extensions have been proposed; +: yes; R.E.: RDF Explorer; W.: WONDER system.

Feature	UML	ORM2	W.	VSB	R.E.	CoSMo
Multilinguality	–	–	–	–	–	+
Modularity	–	±	–	–	–	+
Class-based	+	+	+	+	+	+
Instances	±	–	–	–	+	+
Role naming	+	+	–	–	–	+
Functions	+	±	–	–	–	+
Mandatory and optional	+	+	–	+	–	+
Value constraint	+	+	+	+	+	+
Join/merge	–	±	+	+	–	+
Disjoint., cardinality, etc.	+	+	–	–	–	–
'Perspectivist'	+	–	–	+	–	–
Positionalist	+	+	–	–	–	+

5.2 CoSMo in the context of Abstract Wikipedia and beyond

The CoSMo language addresses the declarative part of the content selection specifically and in such a way that it is essentially an enhanced, non-RDF/SQL/ JSON/XML-expert multilingual modular view specification language that is platform-independent. While it indeed couples the elements of CoSMo to P and Q items, their respective labels easily can reside in a separate array for some tool that does not reside in the Wiki ecosystem. As to the semantics of the language, we opted for a technology-independent pivot to prevent CoSMo possibly 'breaking' as soon as Wikidata would be moved into another technology, or users elsewhere would face the semantic specification hurdle.

Since any NLG technology requires the same sort of access to the source content, instead of requiring each NLG pipeline developer to specify their own constructor language for content selection, this can now be done once for all such tools. For Abstract Wikipedia specifically, it means that users can write a constructor *once*, and thus *regardless of* which NLG technique is used to convert it into natural language—be this GF [22], Ninai [20], SimpleNLG [11], under-resourced Niger-Congo B languages [18], or another technique and tool—and *regardless of* which data store technology Wikidata is housed in.

Down the pipeline, verbalisation considerations come to play. The semantics, via role assignment, needs to address both world knowledge and linguistic knowledge. Roles should be assigned in the early stages of the constructor creation. Semantic roles, in the linguistic sense, are concerned with abstract roles such as agent, patient, material, etc., usually attached to arguments and complements. Syntax comes at the language-dependent realisation stage, along with phonological and/or orthographic adjustments and generation of referring expressions (steps 5 and 6 in Fig 5). Syntactic roles, such as subject, are effected via language-specific rules and

features. Linguistic roles will be added after the content selection phase that we are focussing on here with CoSMo. Further down in the pipeline, one may want to add that r1 is the actor and r2 the undergoer. This can be added to the constructor or a new one may be created. An illustration of a possible extension could be as follows (in shorthand notation):

```
TC:C1:L(P40(r1,r2), r1:Q7566, r2:Q29514218; r1-Actor, r2-Undergoer)
```

where the constructor's name is modified to allow for 1:n between the content constructor and other 'constructors' for the linguistics-oriented abstract representation later in the NLG pipeline.

Whether that information is padded into the content constructor like illustrated above, or addressed differently is out of this paper's scope. Likewise out of scope but perhaps useful in visualising the pipeline: after the possible linguistic additions, one can move on to syntax trees or template specifications.

Last, one may want to automate generating draft constructors to augment constructor authoring by community members. This could be by, e.g., assisting graph navigation to select content from Wikidata or to induce them from sample sentences. They are tasks that will contribute to the success of CoSMo, as well as depend on CoSMo for their successful algorithm design.

6 CONCLUSION

This paper presented a novel modelling language, CoSMo, with three key novelties 1) multilingual modelling, 2) content selection covering declarative content and functions, and 3) inclusion of both classes and instances. It was developed following a rigorous language design process with stakeholder consultation. The preliminary evaluation showed that it met these and other requirements, so that it may be useful both in the early stage of the NLG pipeline of Abstract Wikipedia as well as other contexts. Constructors built with CoSMo are expected to facilitate community contributions to a multilingual Wikipedia and it meets the constraints provided by the platform, also enabling people of less resourced and less documented languages to specify and generate content for Wikipedia. Current and future work includes soliciting community feedback and a larger evaluation of CoSMo. A tool to assist with authoring constructors is also planned.

Acknowledgements. We are grateful for for the participants' contribution to the discussions in the various meetings; the meeting participants were: Kutz Arrieta, Maria Keet, James Forrester, Ariel Gutman, Cory Massaro, Arthur Lorenzi, Denny Vrandecic, Ellen Dodge, Nick Wilson. This work was financially supported in part by the National Research Foundation (NRF) of South Africa (Grant Number 120852) (MK) and Google.org (KA).

REFERENCES

[1] Gabriel Amaral, Odinaldo Rodrigues, and Elena Simperl. 2022. WDV: A Broad Data Verbalisation Dataset Built from Wikidata. In *The Semantic Web – ISWC 2022*, Ulrike Sattler, Aidan Hogan, Maria Keet, et al. (Eds.). Springer, Cham, 556–574.

[2] Kutz Arrieta, Pablo Fillottrani, and C. Maria Keet. 2023. *CoSMo: A constructor specification language for Abstract Wikipedia's content selection process.* Technical Report. arXiv:2210.12027 https://arxiv.org/abs/2308.02539

[3] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet Project. In *36th Association for Computational Linguistics and 17th Intl. Conf. on Computational Linguistics, Vol 1.* ACL, 86–90.

[4] Drazen Brdjanin, Mladen Grumic, Goran Banjac, Milan Miscevic, Igor Dujlovic, Aleksandar Kelec, Nikola Obradovic, Danijela Banjac, Dragana Volas, and Slavko Maric. 2023. Towards an Online Multilingual Tool for Automated Conceptual Database Design. In *Intelligent Distributed Computing XV*, Lars Braubach, Kai Jander, and Costin Bădică (Eds.). Springer, Cham, 144–153.

[5] Diego Calvanese, C. Maria Keet, Werner Nutt, Mariano Rodríguez-Muro, and Giorgio Stefanoni. 2010. Web-based Graphical Querying of Databases through an Ontology: the WONDER System. In *Proceedings of ACM Symposium on Applied Computing (ACM SAC'10)*, Sung Y. Shin, Sascha Ossowski, Michael Schumacher, et al. (Eds.). ACM, 1389–1396.

[6] Pablo Fillottrani and C. Maria Keet. 2021. Evidence-based lean conceptual data modelling languages. *Journal of Computer Science and Technology* 21, 2 (Oct. 2021), e10. https://doi.org/10.24215/16666038.21.e10

[7] Pablo R. Fillottrani, Enrico Franconi, and Sergio Tessaris. 2012. The ICOM 3.0 Intelligent Conceptual Modelling tool and methodology. *Semantic Web Journal* 3, 3 (2012), 293–306.

[8] P. R. Fillottrani and C. M. Keet. 2020. An analysis of commitments in ontology language design. In *11th International Conference on Formal Ontology in Information Systems 2020 (FOIS'20) (FAIA)*, B. Brodaric and F. Neuhaus (Eds.), Vol. 330. IOS Press, 46–60.

[9] P. R. Fillottrani and C. M. Keet. 2020. KnowID: An architecture for efficient Knowledge-driven Information and Data access. *Data Intelligence* 2, 4 (2020), 487–512.

[10] Kit Fine. 2000. Neutral Relations. *The Philosophical Review* 109, 1 (2000), 1–33.

[11] A. Gatt and E. Reiter. 2009. SimpleNLG: A Realisation Engine for Practical Applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG'09)*, E. Krahmer and M. Theune (Eds.). ACL, 90–93.

[12] Giancarlo Guizzardi, Claudenir M. Fonseca, Alessander Botti Benevides, João Paulo A. Almeida, Daniele Porello, and Tiago Prince Sales. 2018. Endurant Types in Ontology-Driven Conceptual Modeling: Towards OntoUML 2.0. In *Proc. of ER 2018 (LNCS)*, J. C. Trujillo et al. (Eds.), Vol. 11157. Springer, 136–150.

[13] Terry Halpin and Tony Morgan. 2008. *Information modeling and relational databases* (2nd ed.). Morgan Kaufmann.

[14] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. *Knowledge Graphs.* Technical Report. arXiv:2003.02320 https://arxiv.org/abs/2003.02320

[15] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. 2003. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics* 1, 1 (2003), 7–26.

[16] C. Maria Keet. 2023. An analysis of positionalism's roles in use. In *14th International Conference on Formal Ontology in Information Systems 2023 (FOIS'23) (FAIA)*, Vol. xx. IOS Press, in press. 18-20 July, 2023, Sherbrooke, Canada.

[17] Zubeida C. Khan and C. Maria Keet. 2021. Structuring Abstraction to Achieve Ontology Modularisation. In *Advanced Concepts, Methods, and Applications in Semantic Computing*, Olawande Daramola and Thomas Moser (Eds.). IGI Global, 72–92.

[18] Zola Mahlaza. 2022. *Foundations for reusable and maintainable surface realisers for isiXhosa and isiZulu.* PhD Thesis. Department of Computer Science, University of Cape Town, South Africa.

[19] Z. Mahlaza and C. Maria Keet. 2023. Surface realisation architecture for low-resourced African languages. *ACM Transactions on Asian and Low-Resource Language Information Processing* 22, 3 (2023), 1–26.

[20] Mahir Morshed. 2023. Using Wikidata Lexemes and Items to Generate Text from Abstract Representations. *Semantic Web Journal* (2023), (submitted).

[21] Martha Palmer, Claire Bonial, and Jena D Hwang. 2017. VerbNet: Capturing English verb behavior, meaning and usage. In *The Oxford Handbook of Cognitive Science*, Susan E. F. Chipman (Ed.). Oxford University Press, 315–336.

[22] Aarne Ranta. 2023. *Multilingual Text Generation for Abstract Wikipedia in Grammatical Framework: Prospects and Challenges.* Springer, Cham, 125–149.

[23] E. Reiter and R. Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering* 3 (1997), 57–87.

[24] Hernán Vargas, Carlos Buil-Aranda, Aidan Hogan, and Claudia López. 2019. RDF Explorer: A Visual SPARQL Query Builder. In *The Semantic Web – ISWC 2019*, Chiara Ghidini, Olaf Hartig, Maria Maleshkova, et al. (Eds.). Springer, Cham, 647–663.

[25] A. C. Varzi. 2004. Mereology. In *Stanford Encyclopedia of Philosophy* (fall 2004 ed.), E. N. Zalta (Ed.). Stanford. http://plato.stanford.edu/archives/fall2004/entries/mereology/.

[26] Denny Vrandecic. 2020. *Architecture for a multilingual Wikipedia.* Technical Report. arXiv:2004.04733 https://arxiv.org/abs/2004.04733

[27] Denny Vrandecic. 2021. Building a multilingual Wikipedia. *Commun. ACM* 64, 4 (2021), 38–41.

[28] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. 2019. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intelligence* 1 (2019), 201–223.