Contents

1 Introduction .................................................. 3
2 WSML Variant ................................................. 4
3 Naming .......................................................... 5
   3.1 Use of IRIs in WSML ..................................... 5
   3.2 Best practices: Naming .................................. 7
   3.3 Namespaces .............................................. 7
   3.4 Best Practices: Namespaces ............................. 9
4 Importing Ontologies and Using Mediators .................. 11
5 Non-functional Properties ................................... 13
6 Conclusions .................................................... 14
Bibliography ..................................................... 15
1 Introduction

The aim of this WSML deliverable is to provide a set of Naming and Design Rules (NDR) for WSML, essentially specifying guidelines and best practices for building WSMO Ontologies, Web Services, Goals and Mediators using the WSML formalism. This deliverable builds on top of the work already undertaken in specifying the WSMO Conceptual Model in WSMO deliverable D2 [Roman et al., 2005] and the specification of the WSML Language in WSML deliverable D16.1 [de Bruijn et al., 2005].

Where possible this deliverable will show examples of bad practice and of good practice with respect to elements of the WSML syntax that are either optional, ambiguous or unspecified. These guidelines are aimed at those building WSMO descriptions using WSML and especially at those building tools, applications or APIs for WSMO, so that they can better guide the user through in building such descriptions.

In this first version of the deliverable the focus is placed squarely on the base elements of the WSML language, namely identifiers, namespaces, WSML variants, importing ontologies and non functional properties. Later versions of this deliverable may go into more detail on other more specific parts of the language.

The document is structured as follows: Section 2 describes best practices with respect to WSML variant keyword usage. Section 3 provides a set of guidelines and best practices for naming in WSML. More precisely it discusses how and when to use IRIs and namespaces in WSML documents. Guidelines for importing ontologies and using mediators are available in Section 4. Finally, Section 5 presents a set of guidelines and best practices for describing non-functional properties of various WSMO elements using WSML and Section 6 concludes and presents our future work.
2 WSML Variant

The WSML deliverable D16.1 [de Bruijn et al., 2005] specifies that the `wsml-`\texttt{Variant} keyword at the beginning of a WSML document is optional. However it also specifies that \textit{In case no variant is specified, no guarantees can be made with respect to the specification and WSML-Full may be assumed.} This is obviously not a useful scenario and from a best practices stand point it is recommended that the `wsmlVariant` is always specified. It is also recommended that the lowest variant suitable for the task at hand is chosen, i.e. it is not recommended to write all WSML documents with the `wsmlVariant` specified as WSML-Full.
3 Naming

Named entities in WSML (ontologies, concepts, instances, web services, goals among many others, with the notable exception of variables) are named with global identifiers. There are two kinds of these identifiers - IRIs and anonymous IDs. IRI stands for Internationalized Resource Identifier, the successor of URI (Unified Resource Identifier), a well known format for identifiers on the World Wide Web. Entities identified with IRI can be unambiguously referenced in the scope of the whole WWW. Anonymous identifiers, on the other hand, are assigned to WSML entities that need not be referenced directly, for example concept instances that form a knowledge base, where only the attribute values of the instances are significant.

While WSML gives structure and hierarchy to its entities (concepts are in ontologies, choreographies are in web service interfaces etc.), it does not impose any constraints on what the identifiers should look like, and neither does the Web describe many relationships between similar but different IRIs. It can be easy for a creator of WSML descriptions to choose IRIs and their combinations that could lead to complications or difficulties for users or maintainers of the description.

This section attempts to help the designer evaluate the various naming choices in light of different uses or expectations for the WSML description. In the following subsections, we first analyze the use of IRIs as identifiers in WSML, pointing out several potential issues, and we propose best practices for assigning identifiers to WSML entities so that those issues are avoided. Later we talk about WSML name spaces (often written as one word, “namespaces”) and the related potential issues, and again we propose best practices for avoiding said issues.

3.1 Use of IRIs in WSML

IRIs are global — the meaning of an IRI (the resource identified by the IRI) is independent of the context in which the IRI is used. It is technically impossible to enforce this globality without resorting to a mandatory central registry, but various IRI schemes provide different mechanisms for helping create IRIs that do not collide.

For example, the HTTP IRI scheme (IRIs that start with http:) relies on assignment of DNS domain name owners and on directory-like path hierarchy, and it assumes that any subspace of the potential HTTP IRIs is governed by one authority. For instance, IRIs under http://deri.org/ are ultimately assigned by the webmaster of DERI International. In a very different approach, the UUID IRIs (urn:uuid: scheme [UUID, 2005]) rely on IP address assignment, current time and other randomization schemes in order to minimize the risk of two applications creating the same UUID independently, and using it for different things.

However, WSML has no requirement that would mandate that two entities have different IRIs. Instead, just like the Web, WSML treats IRIs as all existing in potentia, and any given WSML document only adds statements about resources that are named there. Therefore it is not necessarily inconsistent if two documents both define the same concept, even with different attributes.

Similarly, WSML does not mandate that different types of entities (ontolo-
gies, concepts, instances, goals etc.) shouldn’t share identifiers. In particular, treating the same thing as both a concept and an instance is a well-known modeling technique (usually called “metamodeling”). In this case, one definition says http://example.com/entity is a concept, another definition says it is an instance, and such duality (http://example.com/entity is both a concept and an instance) has well-known consequences in logical frameworks.

From the point of view of the Web, if two entities are both identified with the same IRI, they are, in fact, a single entity. It may not be obvious that it can become an issue when dealing with properties common between entities, such as used mediators, imported ontologies, or non-functional properties. That is, these common properties are shared by entities with the same name, so for example in the case of metamodeling, the concept http://example.com/entity has the same non-functional properties as the instance with the same identifier. For illustration, the following listing has two parts that are equivalent in meaning:

Listing 3.1: Illustration of effect of metamodeling on non-functional properties

```
namespace { "http://example.com/" }
ontology example1
  concept entity
    dc#description hasValue "the concept entity"
  endnfp
instance entity
  nfp
    dc#description hasValue "the metamodeled instance entity"
  endnfp
// the ontology above is the same as the ontology below
ontology example2
  concept entity
    dc#description hasValue "the concept entity"
  instance entity
    nfp
      dc#description hasValue "the metamodeled instance entity"
    endnfp
```

Outside WSML, Semantic Web technologies like RDF work with statements about resources, so any statement about http://example.com/entity will naturally apply both to the concept and to the instance with the same identifier. Therefore, metamodeling (and in general sharing of identifiers between entities of different type) should only be applied in situations where it is acceptable that the Semantic Web will not be able to distinguish between the different aspects (types) of the entity (i.e. that the Semantic Web cannot say anything about the concept http://example.com/entity that would not apply to the instance as well).

When using anonymous identifiers, no name reuse or collisions are possible (due to the mechanics of how anonymous identifiers work), but entities with anonymous identifiers cannot be referenced directly. Instead of saying “that instance” (by mentioning its IRI identifier), we can identify an anonymous instance indirectly, for example as “the instance that is the value of attribute hasCar or another (known) instance”. This kind of indirection is fragile (relationships may evolve) and thus it limits the reuse of the anonymous entities in unexpected contexts.
3.2 Best practices: Naming

The following points define guidelines that should be considered when working with IRIs in WSML:

Create distinct IRIs for every entity in your model.

Apart from clear cases of metamodeling, avoid overloading IRIs because it is likely that some users will want to reuse only one of the types of the entity (e.g. the concept or the instance) and unnecessarily overloaded entities may hinder such partial reuse. Additionally, creating new IRIs is “cheap”, especially when using fragment identifiers; see the following guideline for discussion of name spaces that use fragment identifiers.

Make your IRIs dereferencable.

According to [WebArch, 2004], it is recommended to have IRIs dereference to some document, either directly or via redirect. See more tips about this guideline in Section 3.4 where namespace IRI ending is discussed.

Give IRIs to all your entities.

Avoid using anonymous IDs if you can conceivably create useful permanent IRIs for entities that will be valid for a longer period of time. Short-lived instances can have anonymous identifiers, but if there is a chance that an entity could be useful outside a small bounded system, or for a longer period of time, it will be better if it has an IRI rather than being anonymous.

3.3 Namespaces

For the ease of authoring of WSML documents, WSML has a mechanism for shortening the notation of IRIs, called namespaces. The discussion so far has deliberately avoided issues of grouping names in namespaces, as overstressing namespaces often leads to suboptimal IRI entity identifiers.

IRIs that lexically end with a word (no special characters, only alphanumerics), can be split into a first part, called namespace IRI, and a second part, called local name. The beginning of a WSML document declares a number of namespace IRIs and assigns namespace prefixes to them, including an empty prefix for a single default namespace. The namespaces are then used by sQ-Names (serialized QNames), which represent an IRI that is the concatenation of the namespace IRI corresponding to the used namespace prefix with the local name. Listing 3.2 shows namespaces taken from a typical WSML document:

```
namespace { ”http://www.example.org/ontologies/example#”,
            dc “http://purl.org/dc/elements/1.1#”,
            foaf “http://xmlns.com/foaf/0.1/” }
ontology ”http://www.example.org/ontologies/example”
concept Human
   nonFunctionalProperties
      dc#description hasValue “concept of a human being”
   endNonFunctionalProperties
   hasName ofType foaf#name
```

The first line declares the default namespace, the following two lines declare namespaces with prefixes “dc” and “foaf”. The names of the concept Human and of its attribute hasName are written as local names in the default namespace;
Listing 3.3: Same as Listing [3.2] only without namespaces

```xml
ontology _:http://www.example.org/ontologies/example
concept _:http://www.example.org/ontologies/example#Human
  nonFunctionalProperties
    _:http://purl.org/dc/elements/1.1#description
      hasValue "concept of a human being"
  endNonFunctionalProperties
    ofType _:http://www.example.org/ontologies/example#hasName
```

the non-functional property name description is in the namespace “dc” and the type of the hasName attribute is in namespace “foaf”. The same ontology could be written without using namespaces as shown below in Listing 3.3.

The readability and usability difference is obvious, especially when a namespace is used by multiple symbols, yet there are a number of issues of which one should be aware when using namespaces in WSML.

First, namespaces are identified in WSMO with prefixes, but the prefixes have no meaning other than to point to a given namespace IRI. While it is customary to use short descriptive prefixes like “dc” for Dublin Core properties, “foaf” for the Friend-of-a-friend ontology etc, and the default namespace for the symbols defined in the current, the document would have exactly the same meaning if it used “ns1”, “ns2” etc. for its prefixes, or even “wsml” and “xsd”, which could be confusing for a human reader.

There are no constraints on the form of namespace IRIs, as the local name can only contain a restricted set of characters, so in most cases a local name can be concatenated with the namespace IRI to form a valid resulting IRI. However, it should not be forgotten that the namespace IRI will indeed be concatenated with the entity local names to form their IRIs, so the namespace IRI should end with some kind of delimiter. For instance, consider the namespace IRI `http://example.org/map` declared under prefix “map” — the QName map#entity looks very natural, yet the resulting IRI is probably not what the designer had in mind: `http://example.org/mapentity`. The latter IRI seems to be a sibling, not a child of the former namespace IRI.

A namespace IRI in WSML serves only as the first part of entity IRIs — it gets concatenated with the local name to form the whole entity IRI. On the other hand there is a social expectation that the names in one namespace have something in common. One can in fact see the namespace IRI as an identifier for the bag of names, and the first point when a user encounters a new namespace and wants to learn something about it.

Due to the socially expected function of the namespace IRI (to identify the bag of names, and to define the common characteristic of the entities behind those names), it is also expected that there will be a document retrievable at that IRI — this document is called the namespace document. Since we are talking about WSMO descriptions here, the WSMO document itself is a very good candidate for residing at the namespace IRI.

However, one WSMO document can describe entities from any number of even unrelated namespaces, and it would be ineffective and potentially confusing to have the same document accessible at multiple IRIs. If, on the other hand, we restrict the WSMO document to define only one namespace, it becomes easier to manage.

WSMO documents can contain a number of top-level entities — Ontologies, Web Services, Goals and Mediators. The latter three are usually understood as single things (one cannot easily reuse only part of a goal or mediator in WSMO), but ontologies are practically only containers for the entities defined within, e.g. concepts, relations, instances etc. In this way, the function of an
ontology and namespace is very similar. Indeed, if two different concepts are
in the same namespace, a user will likely expect that they come from the same
ontology. Furthermore, if two ontologies share one namespace, when they grow
and evolve the single namespace increases the likelihood of name collisions —
when a new name is created, the designer can be expected to check for collisions
in the ontology they are working on, but they may not be aware of the other
ontology in the same namespace. This leads us to suggest that any WSMO
document that defines a single namespace should also only contain up to one
ontology.

### 3.4 Best Practices: Namespaces

The following points define guidelines that should be considered when work-
ning with Namespaces:

**Have only one namespace per document.**

To be able to put the WSML document at the namespace IRI, it is useful
to limit it to a single namespace so that multiple copies or redirects don’t
need to be managed.

**Have only one ontology per document.**

Because the function of a namespace is very similar to (and possibly a sub-
set of) the function of an ontology, the previous guideline also implies that
a single WSML document should only contain a single ontology. This will
also help avoid name clashes between ontologies because name clashes are
usually only expected in a single ontology. One could say that ontologies
are name spaces.

**Use prefixes consistently.**

While the automated processor treats namespace prefixes as insignificant,
human designers often make prefixes with descriptive names, like “dc” for
the Dublin Core properties and “foaf” for the Friend-of-a-friend ontology.
To help readability of your WSML documents, avoid namespace prefixes
that convey no meaning, like “xyz”, “ns37”, or prefixes that would be
outright confusing, like “dc” for an ontology of Detroit cars.

**Use default namespace for things in current file.**

As a special case of the guideline above, the default namespace (the one
that does not use any namespace prefix) should be used for symbols defined
in this document. While it may be tempting to save some typing by using
the default namespace for symbols from some other namespace that is
heavily used in the edited WSML document, it will hinder the readability
and maintainability of the document.

**End namespace IRIs either with a hash sign or with a slash.**

By ending a namespace IRI with a slash “/”, the IRIs of entities in this
namespace will be able to contain small documents describing each single
one of them, which may be good for large description that contain many
only weakly related entities. Alternatively, all the IRIs can be set up
to redirect to the namespace document (the WSML file), as suggested
by [WebArch, 2004]. Some Web experts avoid namespaces ending with a
slash because the retrievable IRIs of the entities may imply that the IRIs
identify documents, not abstract entities like a Goal definition.
Namespaces ending with a hash sign “#” make all the entity names into so-called fragment IDs. An IRI with a fragment ID, for example http://example.com/ontology#entity, is dereferenced by taking the IRI without the fragment, i.e. http://example.com/ontology, retrieving the document, and then, according to the type of the document, locating what exactly the fragment ID “entity” means. In WSML documents, fragment IDs identify entities; however in HTML documents, fragment IDs identify points in the document.

Practically, there is a deployment consideration that namespaces ending with a hash sign make it necessary for the descriptions of the constituent entities to be in a single document, whereas namespaces ending with a slash create a number of IRIs where documents or redirects should be set up.
4 Importing Ontologies and Using Mediators

As described in WSMO Deliverable D2 [Roman et al., 2005], a standard mechanism for dealing with the complexities that exist with building an ontology for a particular domain can be resolving using modularisation. Once a modular collection of Ontologies exists it is important that these ontologies can successfully be used together. Within WSMO there are two approaches to achieving this.

- **Importing Ontologies**: In cases where an ontology you are creating needs to reference another ontology in a direct manner where no heterogeneity problems exist the ontology can just be imported.
- **Using Mediators**: In cases where heterogeneity problems exist that need to be resolved the ontology can be imported by referencing the mediator which resolves these heterogeneity mismatches.

Of course it is also possible to just reference elements from another ontology using the IRI’s which identify them, as in the following example:

```xml
ontology1.wsml:
  <wsmlVariant _="http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
    namespace { _ http://www.wsmo.org/examples/ }}
  ontology
    concept gender
    instance male memberOf gender
    instance female memberOf gender
    concept animal
      hasGender ofType gender
    concept human subConceptOf animal
      hasName ofType string

ontology2.wsml:
  <wsmlVariant _="http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
    namespace { _ http://www.wsmo.org/examples2/ }
    examples _ "http://www.wsmo.org/examples/" }}
  ontology
    instance mary_smith memberOf examples#human
      hasName hasValue "Mary Smith"
      hasGender hasValue examples#female
```

The issues that exist with such an approach is that in the second ontology the instance ‘mary_smith’ that is a member of the concept ‘human’ is only referencing the concept as a term. If this ontology were loaded into a reasoner, the reasoner would be unaware of the subsumption hierarchy that concept belongs to or any other semantic information encoded in the first ontology. Therefore the recommended best practice when needing to use entities from another ontology is to always import the ontology either directly or via a mediator (if heterogeneity issues need to be resolved). The corrected version of ontology2 from the previous example can be seen below:
ontology2.wsml:

```xml
<wsmlVariant _http://www.wsmo.org/wsml/wsml−syntax/wsml−rule
namespace {
  _http://www.wsmo.org/examples2/,
  examples _http://www.wsmo.org/examples/
}
ontologynology2
importsOntology examples#ontology1
instance mary_smith memberOf examples#human
  hasName hasValue "Mary Smith"
  hasGender hasValue examples#female
```
5 Non-functional Properties

This section presents a set of guidelines and best practices for describing non-functional properties of various WSMO elements using WSML. First, a set of general guidelines and best practices for the current set of WSML non-functional properties is presented. For some of the most used non-functional properties we present also particular best practices and guidelines in the last part of this section.

Since the set of non-functional properties in WSML is currently provided by Dublin Core Metadata Initiative [Weibel et al., 1998], the guidelines and best practices for any Dublin Core based metatdata description apply also to current non-functional descriptions in WSML. The CDP Metadata Working Group has developed in [DCMBP, 2005] a set of guidelines for creating metadata records for various resources. The following general guidelines and practical considerations apply to all non-functional properties descriptions:

- **punctuation**: avoiding ending punctuation.
- **abbreviations**: in general abbreviations should not be used if they make the description unclear.
- **capitalization**: proper names (places, personal and organization names) and the first word should be capitalized. Same holds for acronyms.
- **initial article**: at the beginning of a description initial articles should in general be omitted.
- **character encoding**: nonstandard characters and diacritics should be used carefully so that they are well display.
- **using controlled vocabularies**: the information which represent the value of a given non-functional property should be expresses by using terminology from controlled vocabularies and thesauri. Such terminology is preferred instead of uncontrolled keywords.

Additionally for each particular Dublin Core-based metadata element [DCMBP, 2005] proposes distinct guidelines. For **dc#title** element from example, some of the most important guidelines are: (1) multiple values should be entered in the order of their importance and (2) they should be as descriptive as possible.

The recommended best practices for **dc#identifier** is to identify the element by means of a string or number conforming to a formal identification system. In Dublin Core formal identification systems include but are not limited to the Uniform element Identifier (URI) (including the Uniform element Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN). WSMO recommends the using IRIs as Identifier as discussed in Section 3.

Some major guidelines for **dc#contributor** are: (1) multiple contributors should appear in the order of their importance and (2) personal names should be introduced in inverted form in most cases: "Last Name, First Name, Middle Name". For more details on guidelines and best practices for the most common used Dublin Core metadata elements we refer the reader to [DCMBP, 2005].

Finally, the recommended best practices **dc#description** are: (1) the description should contain descriptive text, remarks and comments, (2) multiple descriptions should be entered in the order of their importance and (3) other specialized information, not included in all other Dublin Core elements should be entered here.
6 Conclusions

This deliverable aims to provide a set of Naming and Design Rules for WSML. More precisely we have presented a set of guidelines and best practices for naming (e.g. IRIs and namespaces), WSML variants, importing ontologies, used mediators and non-functional properties descriptions. Where possible we have presented examples of good and bad practices. As future work we plan to add more examples, to detail the current content and extend the set of guidelines and best practices presented in this version of the document.

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.

The editors would like to thank to all the members of the WSML working group for their advice and input into this document.
Bibliography


