Table of Contents

Table of Figures

Table of Tables

0

1 Introduction to SMIF

The following is a high-level, non-normative, description of some of the fundamental SMIF concepts.

The fundamental concepts will be described in a way that most practitioners can relate it to their familiar experiences. In this chapter we will gradually build a semantic-conceptual architecture (an architecture that is completely independent of any particular technology and in which there is a clear distinction between the world of the things and the world of the representations of those things).

The prime aim of this chapter is to demonstrate the value that a SMIF semantic-conceptual model and transformations can offer to the ever-increasing need of federation of (information) systems in business and government practice. Note that this section amplifies the reference documentation in section 8.

0.1 Introduction to the SMIF Conceptual Model Foundation

The SMIF conceptual model serves two purposes:

- It defines the SMIF language
- It provides foundation concepts which other models may use, including domain models

SMIF has been built with the expectation that by providing *reference models* that define *common shared concepts* we can either *directly reuse* those concepts or *map* them to related concepts in different models or data structures.

Many of the concepts used to define the SMIF language **may** also be used as reference concepts for domain models. The fundamental concepts needed for the SMIF language are also found in many, if not all, domain models. Examples would be entities, identifiers, situations and values. That said, there is no requirement that these concepts be used or referenced by SMIF domain models – the choice of what reference models to use is made by the domain architect, not by SMIF.

SMIF, as a language for modeling, needs to interoperate with and share concepts with other languages such as UML, OWL or XML-Schema. This is really the same problem as an application containing, for example, company information may need to share information with other applications providing or consuming company information. The basis for sharing information, at any level, is that there are different ways to represent information but they must, ultimately, be sharing the same meaning (concept) for useful communication to take place. Communications takes place when you *understand* what another party has said based on some concept you share about the world, system or domain you are communicating about.

To understand what is said you must have some way to *reference* a concept you share. We reference a shared concept by using *terms*, or "*signs*". Natural language uses words or phrases as these signs. But, since words can have many or fuzzy meanings SMIF also references concepts with model based identifiers. These model based identifiers serve as signs to connect a more formalized definition of a concept with the various ways that concept may be used or expressed.

The following section identifies the SMIF concepts that may also be used in domain models as well as the way concepts have been partitioned in SMIF to enable its use as a reference model. This approach to partitioning models *may* be useful in other domains as well.

0.1.1 Thing

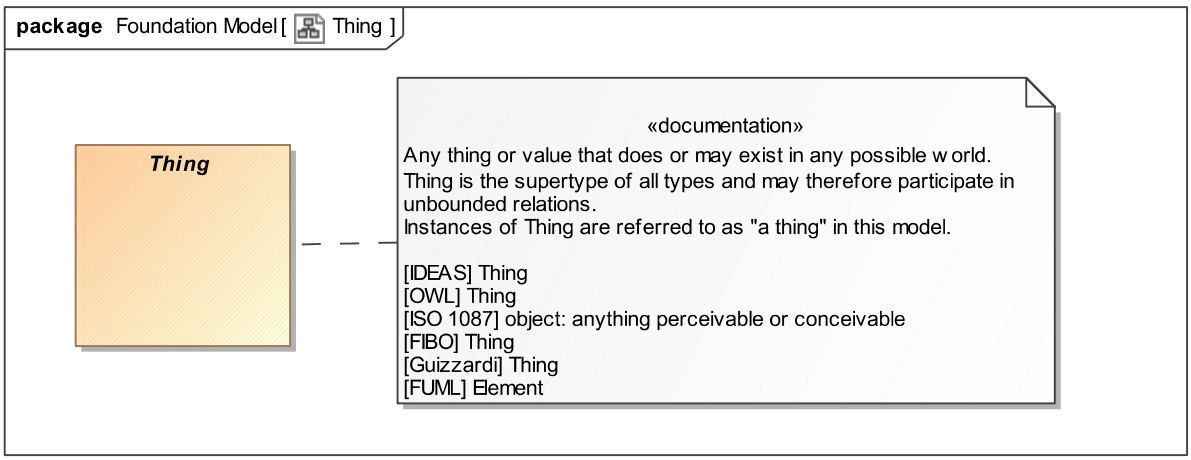

Figure 1.1: Thing

In many models it is convenient to have a sign for anything that could possible be in a model – the most general concept and therefor a "super-type" of everything else. We call this concept "Thing". As a concept for anything, "thing" may be considered somewhat meaningless – but it is a convenient concept, and one that is very common in models and data structures. Our more interesting concepts will all be sub-types of "thing".

Examples of things are "George Washington", "The song – Rock of ages", "Unicorns" and the number "5". Other examples includes a DBMS record about George Washington or a recording of the "Rock of ages". Note that things include "real worlds" things as well as made-up things and data about things we find in computers or filing cabinets.

Semantics

Everything that is in any world, domain, model or data structure is, directly or indirectly, an instance of "Thing".

For all X, Thing(X)

0.1.2 Type

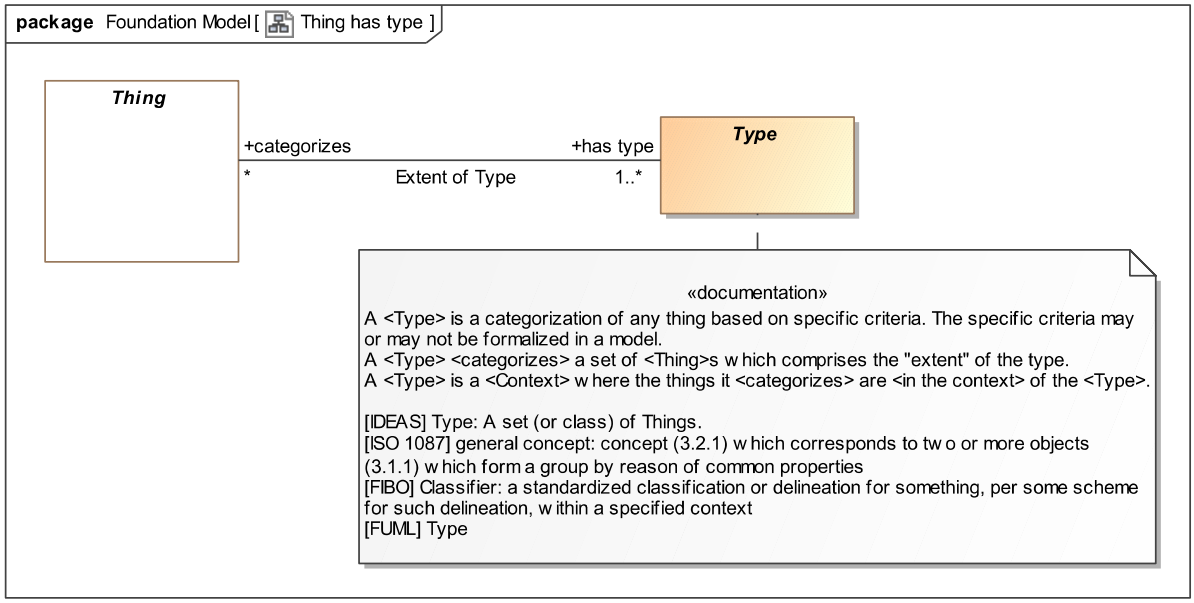

Figure 1.2: Thing has type

A primary way we understand things is by categorizing them as types of things. The concept of "Type" is common across most human and modeling languages. The concept of the type of a thing is also common in domain models, such as product types or kinds of fish. A type <categorizes> a set of things of that type, all of these things <has type> of one or more types. The set of things a type categorizes is called the "Extent of Type".

Things and how they are categorized as types is one of the primary conceptual mechanisms used in SMIF and most other languages – it is part of how we as humans understand the world. Also note that we expect things may have any number of types, and those types could even change over time or be different in various context.

Remember that we said everything is a "Thing", well, types are things as well – we will see how this works later.

As an aside, note a convention we use: that the primary things we are discussing are shaded where as other related things are not.

Examples

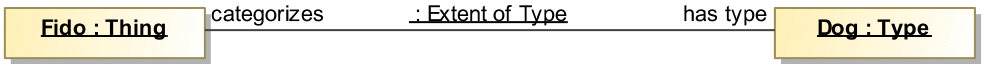

Figure 1.3: Fido is a dog example

In this example we are saying "Fido" is a dog. In terms of the model, there is an "Extent of Type" relationship between "Fido" and the type "Dog" where "Dog" <categorizes> "Fido" and "Fido" <has type> dog. This relationship is one "fact" in our model that can be read either way, from dog to Fido or Fido to dog.

We are also introducing the use of UML "Instance Diagrams" to illustrate our examples.

We would probably never just use "Thing" to categorize "Fido", we would categorize Fido as something more specific - "down the hierarchy" of types – here we see that Fido is a Dog and that Dog is a kind of animal.. As a

shortcut well will usually not show the “Extent of Type” relationship in examples, we will just show the types of something after the name – as is provided for in UML instance diagrams. So the shorthand for the above is just

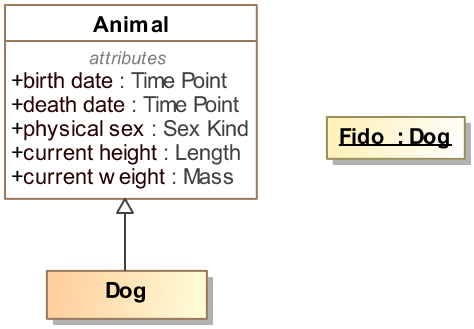

Figure 1.4: UML Type and Instance Example

We should understand that whenever we see an “instance with a name followed by “:” and a type, it means that this instance <has type> of a type with that name. There could be multiple types listed, separated by commas (Fido could also be a Pet).

Semantics

For all things X, where X <has type> T, X shall be conform with the propositions that hold within T.

The set Extent of Type(T) = For all things X, where X <has type> T

In logic, type may also be considered a function, which also implies:

For all things X, where X <has type> T, T(x)

Note: The constructs for determining the propositions that hold within T as well as the semantics of relationships are described below.

0.1.3 Identifiable Entities and Values

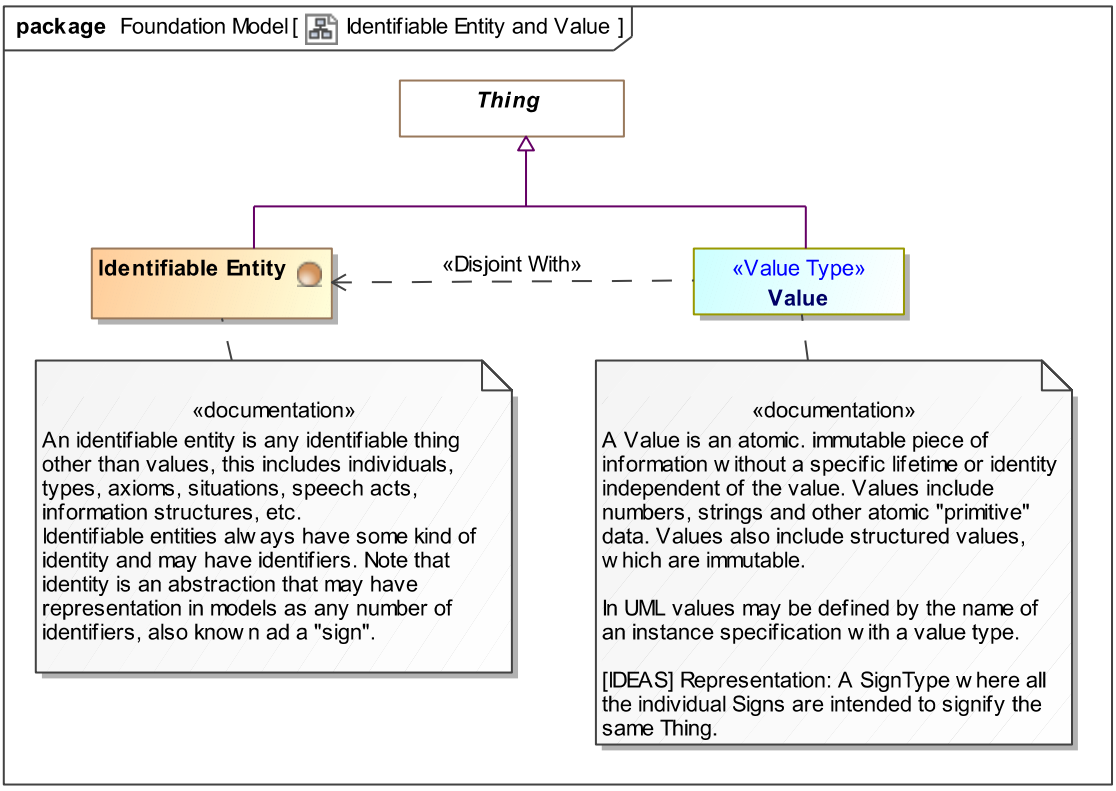

Figure 1.5: Identifiable Entities and Values

We are presenting the concepts “Identifiable Entity” and “Value” together as they are best understood as complementing each other. Identifiable entities and values are, of course, both kinds of things – but of a very different nature. Identifiable Entities are what we mostly talk about – things we give names to, things that have some kind of independent “identity” - everything we can see, touch are identifiable like people, rocks and dogs. Intangibles can also be identifiable, such as purchases or processes. Many, but not all, identifiable things have some kind of “lifetime” where that may change over that lifetime yet retain their individuality.

Values, on the other hand, “just are”. One way this is explained is that values have no identity other than the value it’s self – which can never change and are the same everywhere. All numbers are values as are quantities like “5 Meters” or “pure data” like the text string “abc”. The number “5” is the same number five everywhere (even if it has different representations). The text string “abc” is indistinguishable from the text string “abc” in any other document or database. Values are typically used to describe characteristics of things, such as the weight of rock “R555” is 5 kilograms.

In the SMIF foundation model we partition things as being values or identifiable entities. Something can’t be a value and identifiable entity – these classifications are “disjoint”.This partitioning, like most of our concepts, is found in many other languages – both modeling languages and human languages. Domain models typically use the same kind of partitioning.

Examples

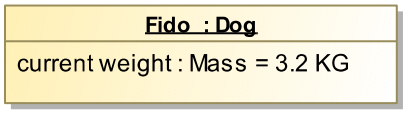

Figure 1.6: Identifiable Entity with Value Characteristic

Returning to Fido for a moment, Fido is clearly an "Identifiable Entity" with a lifetime. We use values, like quantities, to define characteristics of identifiable entities – like their weight.

The above "Characteristic" for Fido, shown as the value of a UML property, states that the current weight of Fido is 3.2 KG – a nice lap dog. This is how values are typically used with identifiable entities (like dogs). We will see in more detail how characteristics are represented in the SMIF model later. We will also see how we can understand how the weight of Fido may change over time (what we see above is just a "snapshot" of Fido, (perhaps when he was a puppy).

The rule we use is that Characteristics are always values as this clearly distinguishes characteristics from relationships between identifiable entities. This is a recommended convention but is not a SMIF constraint to allow for various methodologies.

0.1.4 Identifiers

0.1.4.1 Basic Identifiers

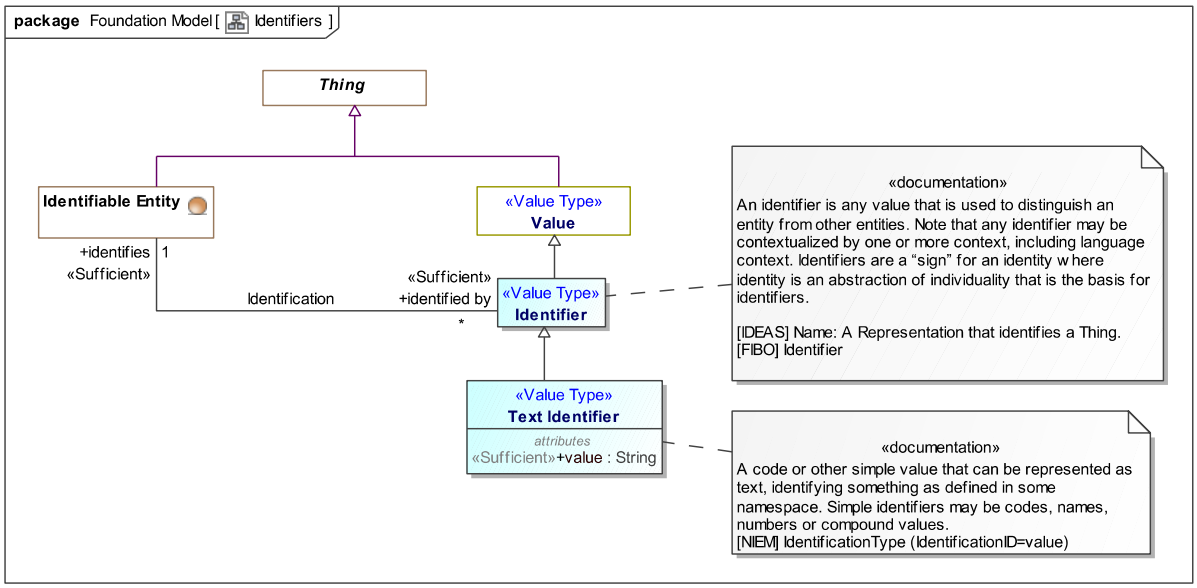

Figure 1.7: Basic Identifiers

Even in our simple examples we have been naming things – giving them "Signs", like "Dog" and "Fido". Most models and data structures have ways to name things. SMIF defines the basic concept of an "Identifier" that <identifies> some identifiable entity (is what makes something identifiable starting to make sense?). There is an "Identification" relationship between an identifiable thing and what it is <identified by>. Note that something may be identified by any number of identifiers, or none at all. *The "thing" that is identified is different from the*

values that identify it – one of its signs. It is also different from a data record that provides information about something. Modelers need to be clear about what the elements in their model really represent.

One of the design philosophies we have used in SMIF is that we should not "commit" to anything unless it is *necessarily true* for the concept we are defining – but when something is necessarily true, we should state it. In this case we don't want to commit to an identifier being text (it could be a picture or a sound). We do want to commit to identifiers being a kind of value as identifiers should not change. In a sub-type we make a stronger commitment - "Text Identifier" has a string value. Text identifier is a sub-type of "Identifier that makes a *commitment* to the value being a string.

Examples

Returning to Fido again; "Fido:Dog" is really a double shortcut, we are asserting two "facts" - that Fido is a dog (which we saw above) and that Fido has the identifier "Fido". A full instance model would look like this:

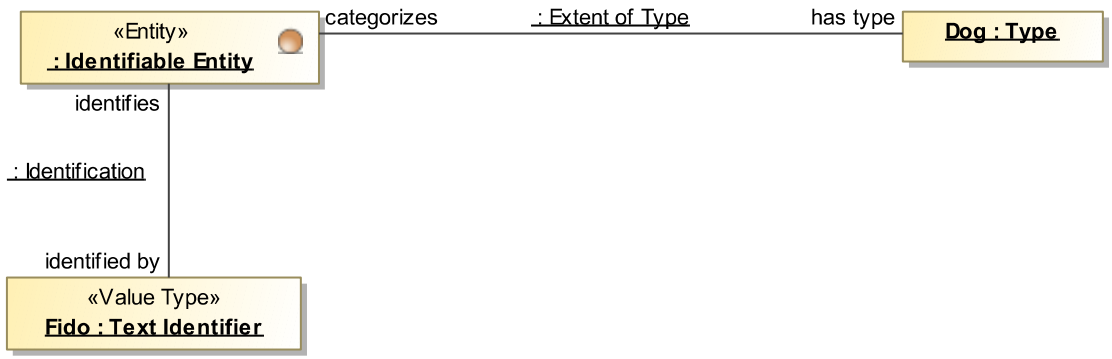

Figure 1.8: Fully expanded type and identifier instance model

Here we see that there is "some identifiable entity" that <has type> Dog and is <identified by> the text identifier "Fido" (noting that there could be other identifiers as well). Also note that the Identifier identifies the entity and that identifier has some *value*. The same value could be used in other identifiers – so at this level we are not saying anything about the identifiers string value "Fido" being unique.

Relating this to some DBMS, we could store a "Record" that represents Fido and has a column representing names. In thinking about the DBMS, we want to distinguish the "real Fido" from records about Fido. The intent of the Fido element above, it is intended as a sign for the real Fido – not data about Fido. Likewise, the "Dog" type is intended to represent "real dogs", not dog records. Of course DBMS systems are real things also, but they contain data *representing* Fido – so *we distinguish a model representing the real things and real relationships between them from records (data) about those things*. This separation of concerns is the foundation of information federation. We will explore this separation of concerns in more depth below.

There are also other types and relationships in SMIF to be able to distinguish names, like "Fido" from controlled identifiers, like a dog-tag number - we will see more about this below.

0.1.4.2 Unique and Preferred Identifiers

Fido may have many names and identifiers, such as his dog tag number and the ID of the "Chip" that can be used to find Fido if he is lost. The dog tag and chip ID are expected to be unique. His name, Fido, could be used for many dogs – it isn't unique but it may be the name we would prefer to use when talking about him.

Since SMIF is intended to work with data from multiple sources that will identify the same things in different ways it is important to be able to hold many forms of identifiers and relate them to the same entity. It is also

important, where identifiers are unique, to be able to understand the scope of that uniqueness – there needs to be some authority or convention that makes them unique. This same “multiple identity” problem exists when any application is “fusing” data from multiple sources – so a foundation model for identifiers has broad applicability.

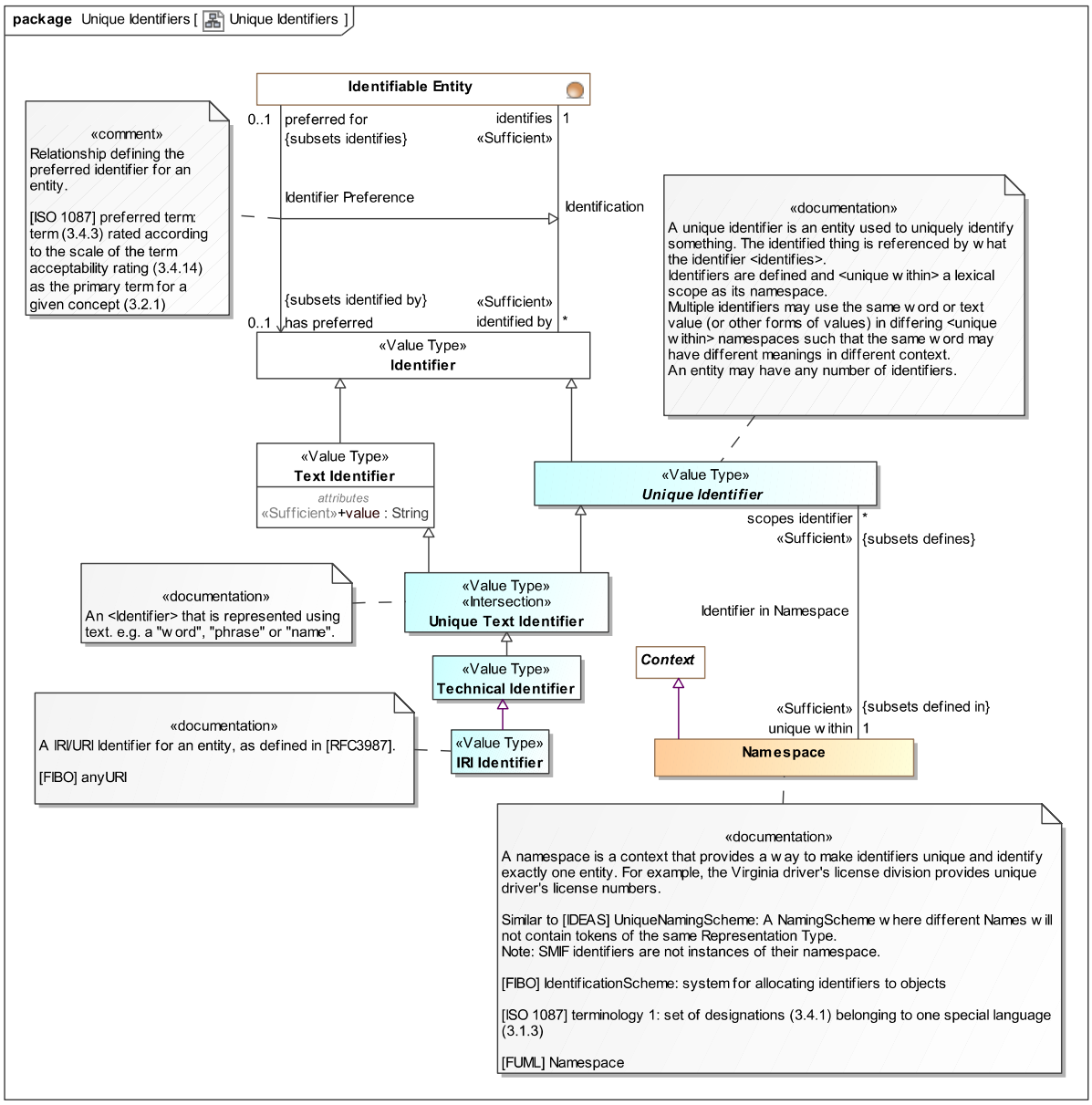

Figure 1.9: Unique Identifiers

In figure 1.9 we have some additional concepts to handle uniqueness and preference.

Note the “Identifier Preference” relationship that specializes the “Identification” relationship we have already seen. Also note the “ends” of this relationship “subset” the corresponding ends of “Identification”. Relationships, as well as the ends of relationships, form a generalization hierarchy from more general to more specific (we don’t always show this hierarchy in summary diagrams). The “Identifier Preference” relationship adds something to Identification, that the <has preferred> identifier has more priority for communication with people. This is a “soft” semantic, intended to assist in human understanding. When we show something we may not want to see all

the identifiers, just the preferred one. Also note that what is preferred in one context may not be preferred in another, such as in another domain or language. We will see how context is handled later.

The other concept we are introducing is that of uniqueness. For some identification value to be unique it really needs to be unique within something – some authority or convention that provides that uniqueness. So a “Unique Identifier” is <unique within> some “Namespace”. A namespace could be technical, like a block of code, or social and based on an authority like names of streets within a town, in which case the town defines the namespace. The URL is a well known kind of unique identifier, based on an IETF standard: 3987. Providing uniqueness in this way is a form of contextualization, we will explore context more below.

0.1.4.3 Names and Terms

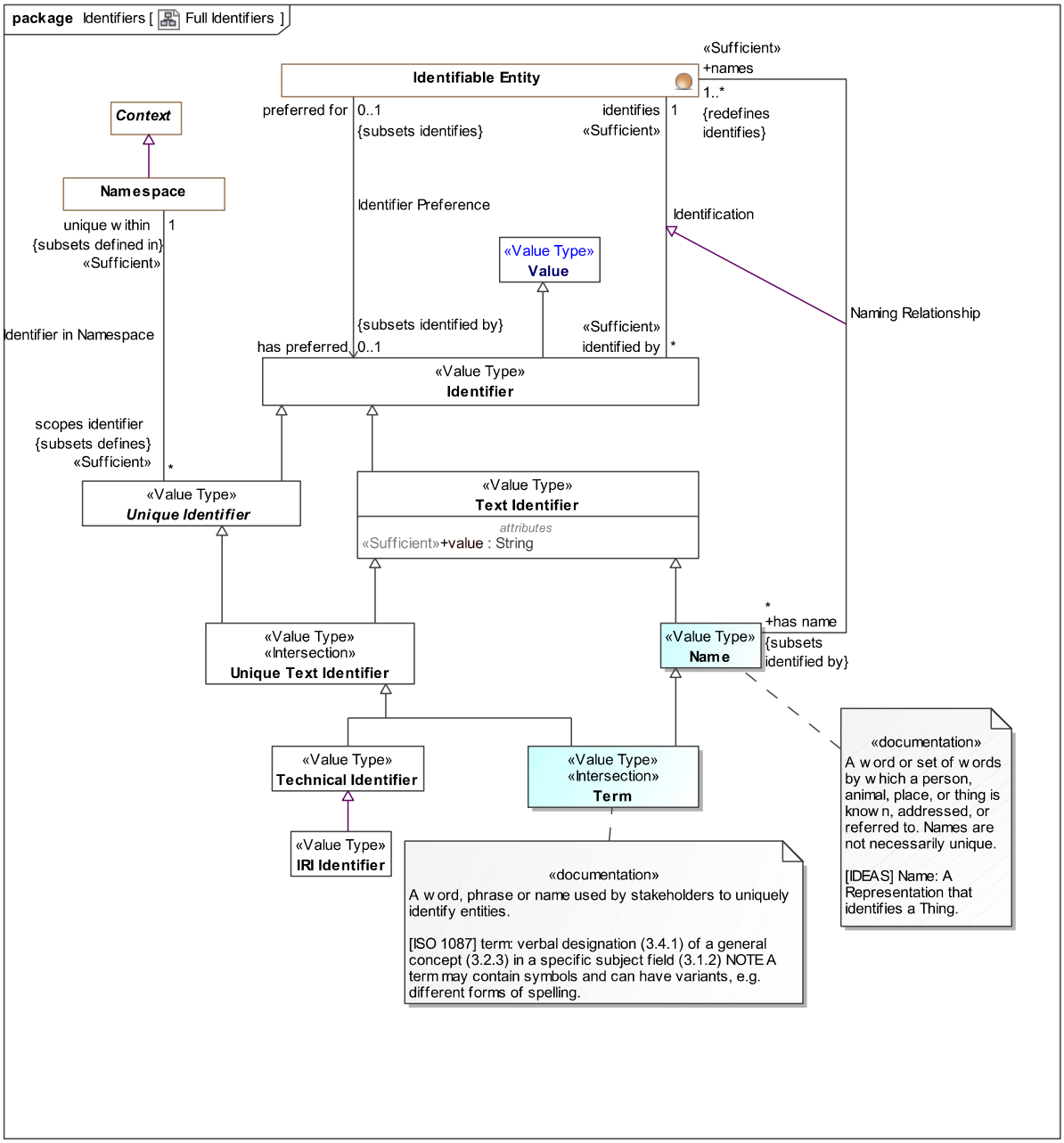

Figure 1.10: Full Identifiers Package

We complete our tour of identifiers by showing the complete identifiers package that includes “Name” Term” and the “Naming Relationship”. Names are identifiers intended to be meaningful to people – must often derived from natural language vocabulary or proper nouns. By typing an identifier as a name (of the concept) we expect that people will be able to relate the name to their intuitive understanding. Compare this with technical identifiers, which may be meaningless symbols. Combining the idea of a name with a unique identifier we get the concept of a “Term”. A term is a name that is unique within some namespace.

Considering the refinements of identifiers we may want to make our Fido example a bit more precise by defining "Fido" as a name and also including a unique identifier, like a Virginia dog tag number.

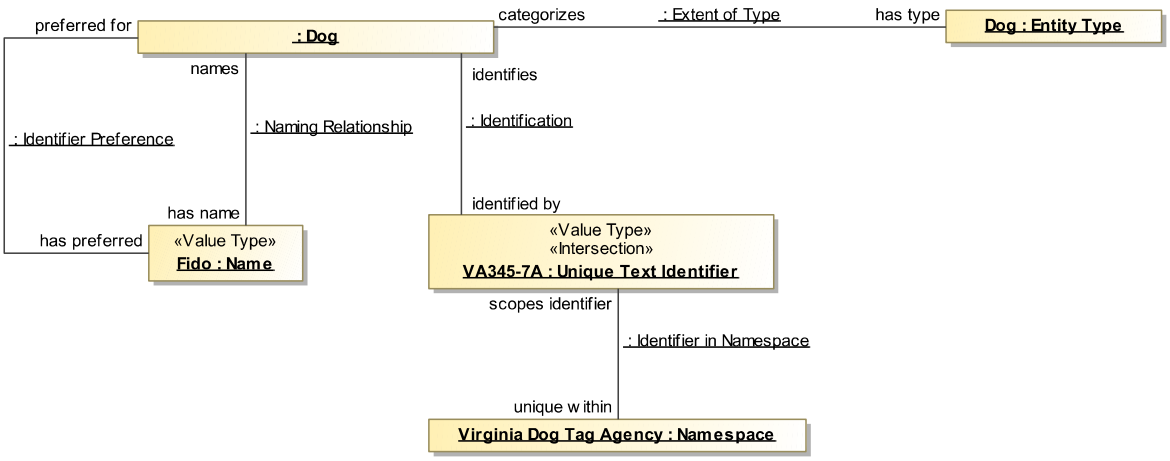

Figure 1.11: Full Identifier Example

From this example we can see that Fido, an identifiable entity that <has type> Dog can have any number of identifiers and that some may be unique within specific namespaces such as the "Virginia Dog Tag Agency". We can also see the "Fido" is a human meaningful name that is the preferred identifier for Fido.

As noted above – this model for identifiers is used by the SMIF language and *may* also be used by domain reference models that deal with names and identifiers.