# Defining the Classes

In this lecture:

- Basic Ontology Modeling based on Red Riding Hood fairytale
- Identifying the classes
- Class derivation
- The challenge of onomatology
- Class derivation

# Basic Ontology Modeling

The first step in a Semantics exercise is transforming the complex story of your business into a set of short sentences consisting of 3 parts:

SUBJECT - PREDICATE - OBJECT

or s.p.o.

There are 2 ways to go about it:

- From the story: your basis is a “manual” of how the domain you are modeling works.

*We deliver food to grandmothers; to do so, our agents have to walk through the forest. They are instructed to stay strictly on the pa0th. The food we deliver can be cakes, but we also deliver wine. There are wolves living in the forest; sometimes, when our agents reach the grandmothers, they find a wolf pretending to be the grandmother instead. The agents are instructed to examine the grandmothers’ eyes and have them talk to assess their voice, so the agents could conclude whether it is a wolf or a grandmother they are talking to.*

- From the data: this happens when the business narrative is not very clear. As a Semantic Architect, you can always ask for data to analyze to draw the narrative out of it.

| *Agent_id* | *Agent_name* | *Path* | *Basket_id* | *Agent_status* | *Agent_status _comment* |
|---|---|---|---|---|---|
| *1* | *Red Riding Hood* | *ForestPath* | *5* | *5 (dead)* | *Eaten by the wolf* |

*When you talk to your counterpart to understand this table, the following story may emerge:*
*“The girl walks through the woods to deliver food to her sickly grandmother (wine and cake depending on the translation). In the Grimms' version, her mother had ordered her to stay strictly on the path. A Big Bad Wolf wants to eat the girl and the food in the basket. He secretly stalks her behind trees, bushes, shrubs, and patches of little and tall grass. He approaches Little Red Riding Hood, who naively tells him where she is going. He suggests that the girl pick some flowers as a present for her grandmother, which she does. In the meantime, he goes to the grandmother's house and gains entry by pretending to be the girl. He swallows the grandmother whole.*

*When the girl arrives, she notices that her grandmother looks very strange. Little Red then says, "What a deep voice you have!" ("The better to greet you with", responds the wolf), "Goodness, what big eyes you have!" ("The better to see you with", responds the wolf), "And what big hands you have!" ("The better to hug/grab you with", responds the wolf), and lastly, "What a big mouth you have" ("The better to eat you with!", responds the wolf), at which point the wolf jumps out of bed and eats her, too. “*

Now, before we start modeling, it is important to understand WHAT we are modeling. Let’s say, our viewpoint is that of a business that wants to create a risk management system that will predict the risk level for the Agent and perform KYC on the grandmas to find out whether they actually are wolves.

# Grandma food delivery ontology based on Red Riding Hood

## Step 1: What are the Things that exist?

From the story above, we can see some things that are mentioned:

- Agent
- Client
- Food basket
- Food
- Forest
- Path
- Enemy - aka Wolf aka Perpetrator

From the data the client brought us, we also have:

- Flower
- Bush
- Tree
- Shrub
- Grass
- House

These are our **candidate classes**.
Why are they “candidate”? Because, inevitably, we step into **knowledge noise**: our client tells us too many details, and our manual may contain information serving different viewpoints or a generic viewpoint.

---

*Note: start exercising your naming conventions right from the beginning. It is easy to get swayed by the story and start recording your classes as they come - in plural here, in singular there. The best practice is to name all your classes in the singular, whenever common sense allows.*

---

## Step 2: What are the Things that are relevant?

Remember, the goal of the exercise is to create a view of the universe that has all the necessary objects that can be used to assess the grandmother our agent is traveling to and minimize the risk for the agent.
We do not need to model the whole reality; only the part that is relevant for our project.
What are some of the classes that don't have any use for that purpose?

- Agent
- Client
- *Food basket*
- *Food*
- Forest
- Path
- Wolf
- ~~Flower~~
- ~~Bush~~
- ~~Tree~~
- ~~Shrub~~
- ~~Grass~~
- *House*

Some things are easy to exclude.
Others, not so much. The golden rule here is:
**DO. NOT. ASSUME.**


Always ask the client when in doubt.

Is it not an option? Record your assumption, so you can remember why you made this or that decision.

What are some questions you would ask your client now to confirm the relevance of the classes that are in *itallic*?

“Does the kind of food influence the risk of encountering a wolf instead of the grandmother?”
“Does the kind of basket matter to assess the risk?”
“What about the house of the grandma; where does it fit in the story?”
This is the answers you get from your client:
*Food kind does matter; wolves are more likely to be drawn to meat pies than to cookies and almost never attack a wine delivery. We have 3 models of baskets, but it is impossible to tell what is inside by the model; I think the wolves react more on the smell than the brand or looks of the basket.*
*We want to know what kind of house we have, and where it is located, so we can send the rescues (we have a Hunter on payroll) if needed, and make sure the Hunter knows where to look for the bedroom (in a 2-story building, it will be on the 2nd floor). Now we only have the address, but extra detail on the construction would be really helpful.*
*It also seems like the encounter with the wolf along the way increases the chances of attack at the destination.*
*And, yes, we know some of the wolves, we keep a database of them, which also helps the Hunter in case we are dialing with a known perpetrator.*

What does our list of classes look like now?

- Agent
- Client
- Food
- Forest
- Path
- Wolf
- House

Listen to the client. Did we just miss an important class?

- Encounter

## Wolf, Enemy or Perpetrator?

All of these words reflect the essence of the entity we are modeling. Which one is the best name for our class though?
The question can be banal - just listen to how the client talks and use their own language. This is an applied ontology, after all - we don’t care about academic nuances.
But sometimes a Semantic Architect has to become a real **Onomatologist**: the naming specialist (in case you were wondering, it is a real thing - think of people who invent the names for brands, companies, projects etc).

What can cause an onomatology challenge?

- Semantic clash. Two parts of the client's organization refer to the same thing differently, or, worse, the same word has different meanings depending on who you ask.
- Hidden entities. These are the entities we are going to discuss when discussing temporality. To put it simply, they don't have a name because nobody thinks of them as separate entities. They are information ghosts floating around and often causing data quality issues in relational databases; just like proper ghosts, they are utterly nameless.
- Semantic precision requirements. Note how we are referring to the contents of the basket as "Food", but the client keeps mentioning that the Agent may also be delivering wine. Technically, wine is not food - the name "Food" for our class would be confusing and would lead to recurrent waste of time (I guarantee you, every meeting you hold would have at least one of the participants raising the fact that wine is not food, and a discussion will always follow). We are not looking for academic, textbook-worthy precision, but we would be wise to rename Food into, for example, Product.

In the class, we chose to call the entity "Enemy" - so I will stick to that in the notes.

## Step 3: Time for S.P.O

Can you tell the story using the classes we just identified, as a set of short sentences having:

- Subject: the entity that the sentence is about
- Predicate: the description of the subject or its link to another subject
- Object: the value of the description of the subject or the linked entity

The sentences:

- X exists / there is X
- X has some relation to Y
- There is a list of possible X
- We care about some attributes of X
- X happens / occurs / is measured / …
- X  happens / occurs / is measured between A and B / for A / at B...

*(the two last sentences are Derivative classes. We will discuss Derivation and Surrogacy later).*

Let's try it out:
*There is Agent*
*There is Client*
*There is Product*

# Derivation: Catching the Ghosts

## Defining relationships

In the previous chapter, we asserted the existence of some things in our data universe. Semantically speaking, we defined the classes.
Now, it is time to define how these classes relate to each other.
Let's take an example:

*There is Forest*
*There is Path*
*Path is located in Forest*

Now, let's perform a validation of this relationship. So far, we've been treating generic classes: conceptual entities we cannot point a finger at. We need to test whether the relationship we defined for the class still stands for its individuals:

*Path is located in Forest*
*Path X is located in Forest Y*

This works. We can find Path X on the map, and it will be in Forest Y, which we can also find on the map.

## Temporality and persistence

And here comes the funny part.

*Agent delivers to the Client.*
*Agent delivers Food.*

Both these statements are correct….on a generic level. However:
*Riding Red Hood delivers to Grandma Suzie*
*Riding Red Hood delivers pies*
Might be true for this particular case, but… we have a delivery business here, and it would be outrageous to think the same agent will always deliver to the same client, and to that client only, or will deliver the same food and that.

We check with the client and learn that:

*An Agent is licensed to deliver only certain products. Riding Red Hood may deliver pies or cakes, but she is not licensed to handle wine nor meat products. And she delivers to whoever comes up, our agents do not have a fixed relationship with a client and we do not do client portfolios.*
Tomorrow it may be true that:
*Riding Red Hood delivers to Grandma Emma*
*Riding Red Hood delivers cakes*
The relationship we initially defined between the Agent and the Client and the Agent and the Product is not persistent on individual level. Remember, we only handle 3-part sentences; we cannot say:
*Agent delivered to Grandma Suzie and now is delivering to Grandma Emma*

This is a clear indication that there is a ghost entity, something we don't talk about, something that lives in the moment and then ceases to exist - and that something has a relationship to both Product, Client and Agent.

Of course… Assignment! The Agent is on an Assignment to deliver certain Product to certain Client. Assignment has a date, and Agent has a link to the assignments past and present.
*Agent has Assignment*
*Assignment is about Product*
*Assignment is about Client*
*Assignment has a date*

*Riding Red Hood has Assignment "Deliver pies to Grandma Emma"*
*Assignment "Deliver pies to Grandma Emma" is about Pie*
*Assignment "Deliver pies to Grandma Emma" is about Grandma Emma*
*Assignment "Deliver pies to Grandma Emma" has date 1 September 2018.*

*Assignment "Deliver cakes to Grandma Suzie" has date 01 August 2019*.

---

Look at the client debrief above. Do you see a relationship we missed between the Agent and the Product? What do you need to ask the client to verify its temporality? Assume a couple of possible answers; how would you model it in each case?

---

## Event classes

Assignment here is an Event class.

*Note that the official Semantics does not recognize class temporality types; that is something that I had to introduce doing Applied Semantics. However, this might be one of the most important concepts for a Semantic Architect to be familiar with.*

Event classes are there to represent the old and tired concept of Transactional Data in generic Information Architecture.
Anything that happens once, lives in the moment and cannot change its attributes is a **transaction**.
A Sale is a transaction. You buy a cup of coffee on a certain date in a certain place. That's it. There is no update possible tomorrow to the price you paid, nor to the kind of coffee you bought. This is a point-in-time datapoint representing an interaction or occurrence in the real world.
*A call received from a client, good shipment from A to B, stock purchase, death of a patient - these are all Events and need to be modeled as Event classes.*

**Event classes:**
**-always have some kind of timestamp on them**
**-never change the value of their attributes**
**-need classes of other types to make any sense at all:** *a call needs to be linked to the Client, shipment - to Location, stock purchase - to Company and Price, death of a patient - to Patient.*
We will talk about the historization of different class types later in this course.

---

Can you identify more event classes in the Riding Red Hood ontology?

---

## Managed Taxonomies

As we see from the table the client provided us, an assignment has a status.

Note: initially, you might be confused into thinking the Agent has the status. This is what happens very often in relational models - they are built for application consumption and do not always align to the real world (and, expectedly, when the business grows, all kinds of problems start emerging with this approach as application needs change). We've done a good exercise in Semantics and now we have the correct classes to attribute the values to; in this case, it's not an Agent, it's the Assignment that has the status - and possibly, the status reflected in the table should actually mean "Agent dead".

---

In fact, there are multiple semantic issues in this table - can you identify them?

---

| *Agent_id* | *Agent_name* | *Path* | *Basket_id* | *Agent_status* | *Agent_status_comment* |
|---|---|---|---|---|---|
| *1* | *Red Riding* | *ForestPath* | *5* | *5 (dead)* | *Eaten by the* |

| | | | | | |
|---|---|---|---|---|---|
| | *Hood* | | | | *wolf* |
| *2* | *Fairy Godmother* | *ForestPath* | *7* | *1 (successful)* | *Delivered pies to Grandma Suzie* |

From the previous lesson, you might remember that whatever HAS a value is a class; whatever IS a value, is an attribute of the class.
Status here certainly seems to be an attribute of a class: it is Status OF something.
However, you don't want your client to go cowboy on that attribute either. Your application needs Status to have only a number of values that we know and that can be used by the application to take decisions.
You really don't need statuses like "Unsuccessful", " Agent down", "Agent dead" to be there to reflect the same exact thing.
You also really don't want to rely on status codes. The issue with these is that they are not a real world thing; they have been invented by a person populating the table. The next person migrating the application may decide to change them, make them a string (so you may get "05" where you used to have 5), or get rid of them altogether.

## *Never trust the database primary key*

*In a relational database, every table would have a primary key - a unique identifier, that is often generated by the system. You cannot change it, and when you delete a record, there will be a gap in the sequence.*
*No matter whether you work with Semantic or traditional paradigm, do not rely on the primary key!*
*Let's imagine a table like this:*

| *Status_id (primary key, auto-increment)* | *Status* |
|---|---|
| *1* | *Success* |
| *2* | *In progress* |
| *3* | *Failed* |
| *4* | *Queued* |

*Now, somebody decides we are no longer going to need the In Progress one, and deletes the record:*

| *Status_id (primary key, auto-increment)* | *Status* |
|---|---|
| *1* | *Success* |
| *3* | *Failed* |
| *4* | *Queued* |

*Imagine you have an application that calls services depending on the status. It has a code like:*
*If status_id ==3, call ambulance*
*Else if status_id ==4, send to next free agent*
*Else do nothing*
*Now imagine you migrated the database, automatically transferring all the data into the new tables. Your ID column is auto-generated, so now you have:*

| *Status_id (primary key, auto-increment)* | *Status* |
|---|---|
| *1* | *Success* |
| *2* | *Failed* |
| *3* | *Queued* |

*And now every time a new assignment is queued, the ambulance gets called (hopefully, you catch this during testing).*
*My point is, never rely on machine IDs for any logic, and certainly not for any semantic logic. Whenever possible, use something that identifies the data by a real-world attribute. In Semantic world, not only can you use URIs for that (which we will discuss later), but you can also embed the logic right on top of your data.*

Anyway, my point is: you want to manage the list of possible statuses. You also want to be able to annotate each status with, for example, the service to call when this status is used.
What you do is creating a **managed taxonomy** of statuses, where Status becomes a class - although it is not in any way a real Thing that exists and is tangible - and all the possible statuses are defined as Individuals of this class.

*Assignment has_status Status*
*Assignment "Deliver pies to Grandma Suzie" has_status Success.*

Managed taxonomies represent the good old **Reference Data**: almost immutable lists of possible individuals in a certain class.
Managed taxonomies do not have to be derivative classes at all: they can also be real Things that exists. Countries, seas, chemical elements, months of the year - these are all reference data that can be reflected as managed taxonomies.

Can you think of more managed taxonomies in the Red Riding Hood use case?