**Guide for conceptualizing domain ontologies**

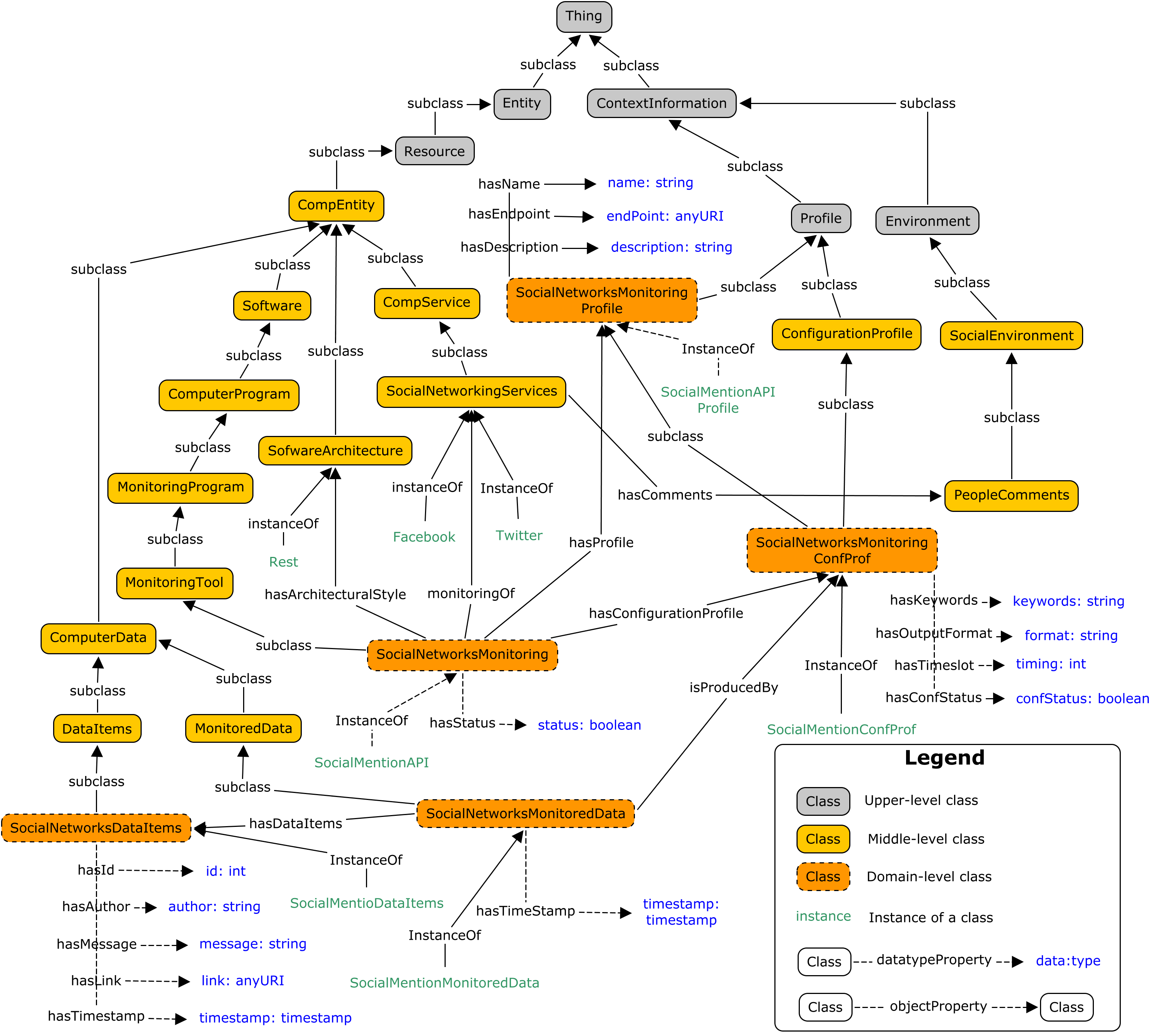
In Fig. 1 we depicted an ontology that has the aim to conceptualize the input, output and general capabilities of a monitoring tool. The ontology defines several classes organized in two levels of abstraction to facilitate its extension: the upper level and the domain level.

* The upper level contains generic classes that have the aim to provide the semantic needed for lower level classes.
* The domain level contains domain classes that have the aim of conceptualizing domain specific concepts (e.g. a specific type of a monitoring tool).

In the ontology, we conceptualize a monitoring tool in the following form: “a *monitoring tool* is a *monitoring program* which is in turn, following the hierarchy of concepts, a *computer program*, a *software* and a *computational entity*”. A *feedback gathering tool* *(or mechanism)* could be modelled analogously following the same hierarchy.

To model the concepts, we suggest to follow the following steps:

1. Create the required domain classes (e.g. *FeedBackMechanism, FeedBackMechanismProfile, FeedBackMechanismConfProf, FeedBackMechanismData).* You can create the classes that you need and name them as you would like for modelling the domain ontology for feedback mechanisms.
2. Make them subclasses of generic classes already defined in the context ontology. For instance, the *FeedBackMechanism* class can extend directly the *software* class or computer program class, depending on the semantic that the modeler wants to represent. If you need intermediate classes that are not yet represented, you should create all of them and trying to connect them in the hierarchy, considering that CompEntity is the most abstract class to represent computational entities.
3. Relate the domain classes created among them to identify dependencies, roles and the data flow for configuring feedback mechanisms and for obtaining their results. For this tasks, you can create or use object properties such as hasProfile, isProducedBy, hasConfigurationProfile, etc.
4. Specify datatype properties for each domain class created. They represent the schema and structure of data for inputs and outputs of the feedback mechanisms. For this task, you can create or use datatype properties such as hasName, hasDescription, hasValue, etc. It is worth noting that each of these datatype properties should be related to an attribute:type (e.g. name:string, ID:bigInt, etc.).



**Figure 1:** Conceptualization of a monitoring tool