This far we have only considered UML classes where the attributes are primitive types rather than classes. Here we will consider UML classes that have classes as attributes. Assume we want to model projects. Assume a project must have one name, one sponsor that must be a manager and it must have a team of between 3 and 10 employees. In UML this can be stated using attributes (see Fig.1(a)) or associations (see Fig. 1(b)). For interest sake Wazlawick [1] suggests using attribute notation for data types and associations for classes. His motivation is that associations makes dependencies between classes more apparent. I usually follow this guideline myself.

The OWL representation for these 2 class diagrams is given in Fig. 2. The first thing to notice is that we use `ObjectProperty` instead of `DataProperty` to represent `sponsor` attribute/association. Similar for the `team` attribute/association. Our property definitions also now have `Domain` and `Range` restrictions. When we say that Susan is the sponsor for ABC, we can infer that Susan is a manager and ABC is project. This information can be capture through `Domain` and `Range` restrictions. For the purpose of finding modeling errors in it is preferable to add texttttDomain and `Range` restrictions.

To limit the number of employees on a team to between 3 and 10 employees we use the property cardinality restrictions `team min 3 owl:Thing` and `team max 10 owl:Thing`. It may seem strange that we use `team max 10 owl:Thing` rather than `team max 10 Employee`. Surely we want to restrict team members to employees? Well true, but that is achieved through our range restriction on the `team` object property. Here we restricting our team to 10 whatever classes and the range restriction will infer that the team must be of type `Employee`.

References

1. R. S. Wazlawick, *Object-oriented analysis and design for information systems: modeling with UML, OCL and IFML*, Morgan Kaufmann, 2014.