Unit 5

Understanding UML
(Reflection)

This week's consideration focused mainly on expanding my understanding of how to document and analyse Unified Modelling Language (UML) systems. We moved from simple object diagrams to developing the object diagram into a class diagram. Together with feedback from fellow students during unit 3 and unit 4 regarding UML modelling, I gained a complete understanding of the meaning of the different types of UML relationships and other UML concepts in general.

UML Granularity

I considered how closely UML ties to the software engineering world. For example, there is the notion of "public", "private", "abstract", "template set", "operation", "class", and "object" that have real-world representations in most programming languages. Such a close connection implies that when modelling in UML, as information system designers, we impact the final software implementation. Therefore, I believe UML models must be reviewed regularly for correctness and realism before blindly submitting to a development team to begin working. From this perspective, if modellers are not permitted to have significant influence over the software implementation, perhaps they are better off using UML to develop *conceptual* and *logical* models? Such models still look like class diagrams but do not dictate the actual code implementation.

There is a delicate collaborative balance between UML diagrams and the software developers' willingness to leverage them. In my experience, developers want to see simple boxes with connectors and few words, leaving the creative aspect of software implementation to development leads or senior developers. For such teams, UML may provide too much detail, and simple Visio diagrams are sufficient. However, regardless of the readiness of development teams to read, understand or accept UML models, I believe UML presents the opportunity to quickly document the inner workings of information systems and how all the components interact.

UML Associations

Upgrading the object diagram in unit 3 to a class diagram has provided me with more profound confidence in wielding UML as a modelling tool. I feel better equipped to model real-world systems, having spent significant time pondering on feedback from other students, continuing to read up about UML. I have reached the following conclusions about UML associations:

1. **Association.**
   These relationships manifest in code via properties or fields. Next, I consider *how* these references are passed into the implementation code. Such connections can only be supplied into the class from an external source or created by the instance. Making this assertion leads me to conclude then that association is no different than composition or dependency.
2. **Dependency.**
   This relationship tells me that some object *requires* the presence of another object to achieve its outcomes. Therefore, I argue that *any* reference to another object instance is a dependency; otherwise, there is no need for such a reference to exist.
3. **Composition.**
   This relationship denotes that an owning class will both create and destroy another object instance.
4. **Aggregation.**
   This relationship uses another object, but neither destroys it nor creates it. Aggregation is a strange relationship because it is no different to dependency or association.

UML Relationship	**Software Development Constructs**
Association	- Instance Property/Field - Bi-directional associations violate encapsulation.
Dependency	- Dependency Injection - Instance Property/Field
Composition	- Factory Pattern, or - Manually instantiate an object instance
Aggregation	Akin to an association relationship.

Reflection on UML Associated Classes

I enjoy the notion of UML associated classes because, in my experience, they closely resemble join-tables in SQL or relationship members in graph databases.

For example,

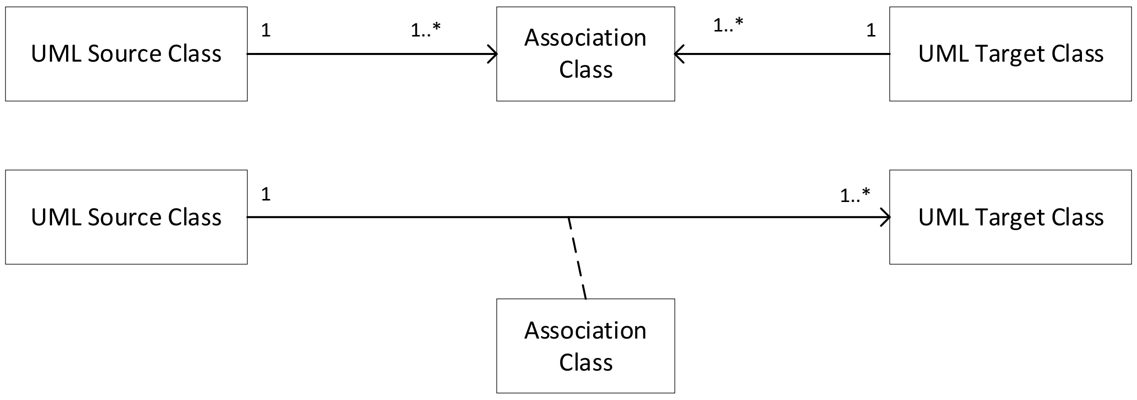

Figure 1 Representation of Association Classes in UML

UML Association classes manifested in technology stack:

SQL SERVER

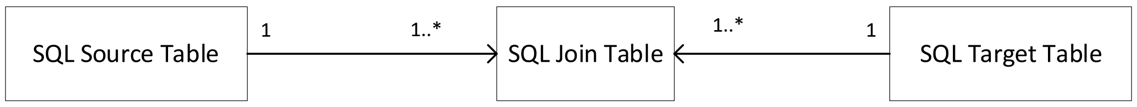

Figure 2 Typical representation of Association Classes in SQL

In the SQL world, the join table exists at the time of schema creation, though inserting data performed when required.

GRAPH DATABASES

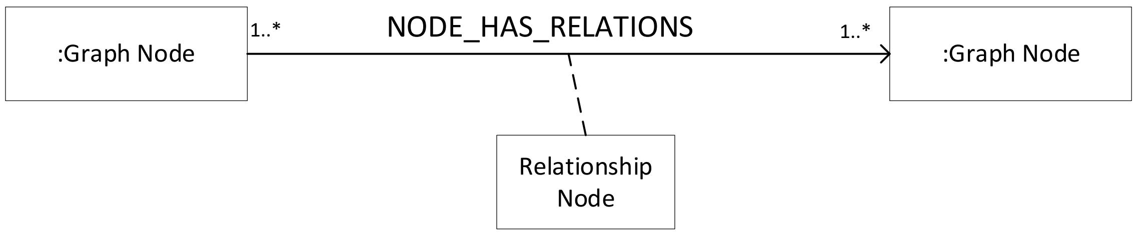

Figure 3 Representation of Association Classes in Graph Databases

In graph databases, the syntax is slightly different (relations are first-class citizens). At a high level, the relationship between two nodes only contains data necessary for *that specific* relationship, as opposed to SQL. For example, every instance of a connected "Association Class" relationship must have data.

Therefore, from my understanding above, I think association classes in UML apply to everyday scenarios when modelling information systems, especially useful when requiring additional data in the presence of two related classes.

Other UML Aspects

Multiple Inheritance

Just like Python, UML supports depicting multiple inheritances through the generalisation association.