A Beginners Guide to UML Part II

Dan Brown, Dunstan Thomas Consulting

Summary

In the first part of this article, I examined the origins and definition of the UML to provide a basic understanding of what it is, and what the UML can offer us. I also explored the architectural views, models available and how these can link together. Now I shall apply the principles discussed to analyse a real business problem and design a solution using the UML.

The problem

Consider a system which relies on acquiring stock valuation data from the financial markets after each day's trading. At present the data is published on a remote web site after close of business each day. Early in the morning a worker comes in to download a set of files from the web site onto the local network. The requirement is to develop an automatic process which will retrieve these files automatically, thus saving an individual from coming into work early and therefore removing any scope of user error. The solution needs to be generic, configurable and re-usable.

Analysis

Remember the UML is a notation and not a process as a whole, therefore I decided to base my initial analysis on a business process model show in Figure 1 below. Then I will then continue my analysis by devising use case models where appropriate and from these a basic class diagram. My choice in the UML models employed will be based on those which I feel will be relevant given the problem and will give me a greater insight into the analysis and design issues that might arise.

Business Process Model

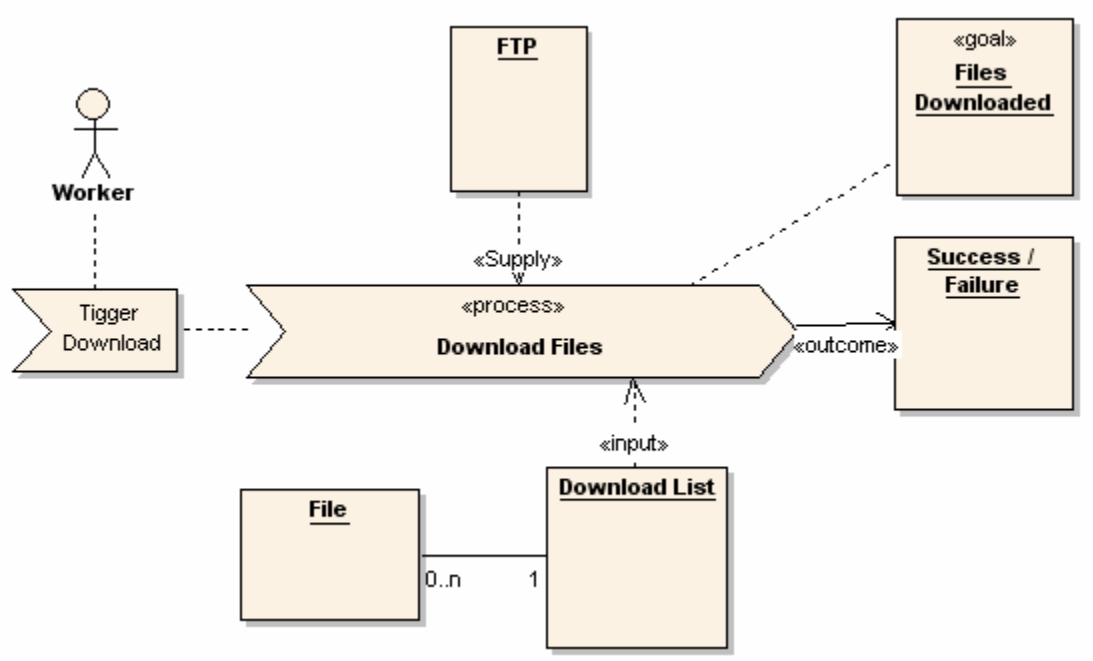

Figure 1: Business Process Model

The business process model allows the procedures and a broad outline of the processes employed by a business to be captured. It also allows the significant inputs, events, resources and outputs associated with a business process to be captured by the analyst.

In my example illustrated in Figure 1 above, the business process model allows the visualization of the process as a whole. The process is to download files from an FTP site which at present is initiated by an employee, is dependent on FTP services and knowledge of the files to be downloaded with either success or failure as the outcome.

Use Case Model

After analysing the business process and the requirement to automate this process, I decided to build several use case models which describe the functionality proposed by the new system. Each component of the use case represents a unit of work and interaction between a human, machine or another process.

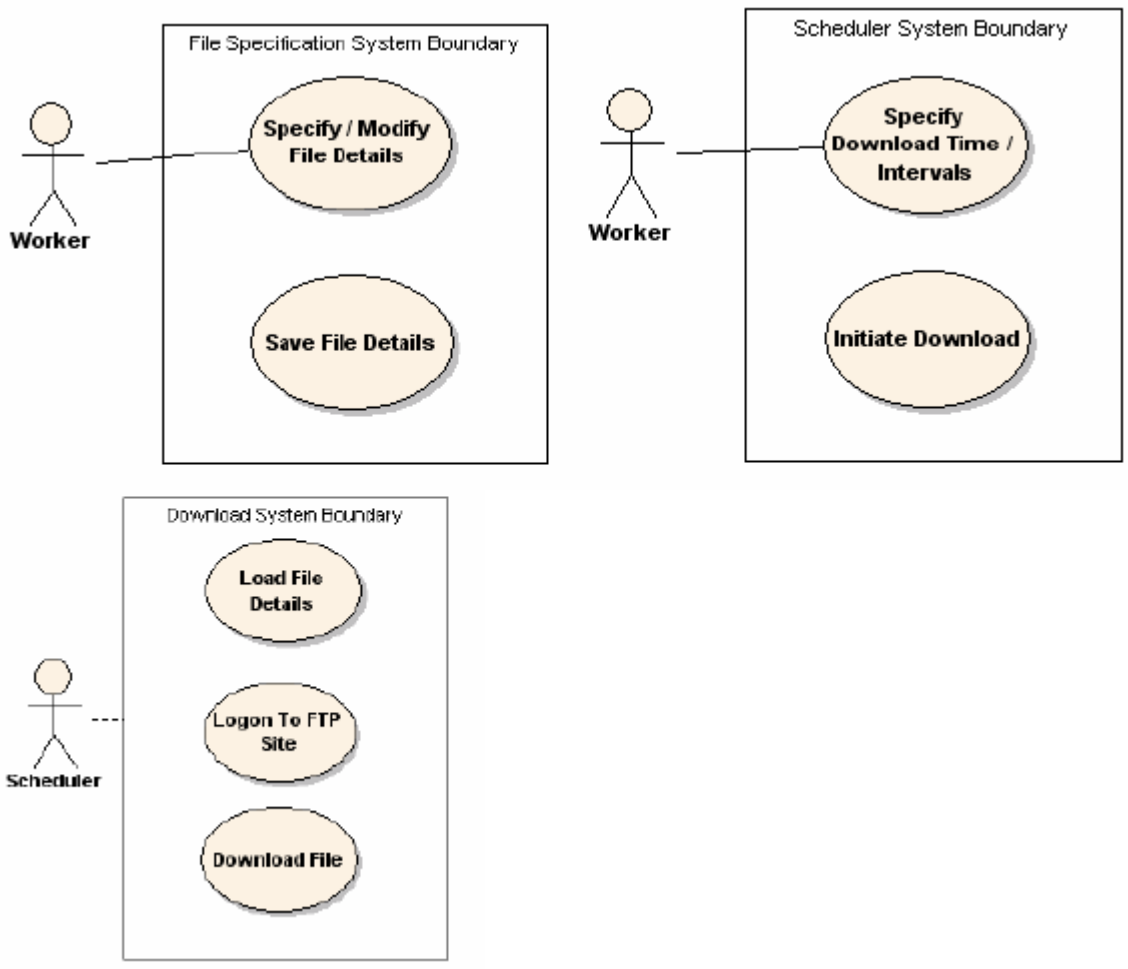

Figure 2: File Specification, Scheduler and File Download Use Case Models

The first use case model shown in Figure 2 above outlines the file specification aspect of our proposed solution. If we are to provide an automated process to download the files, our process needs to be aware of which files need to be downloaded, which FTP site they are situated on and where they need to be downloaded to. This will be initially set up by a worker, known as an actor.

This could be implemented via an ini file or database and therefore will not be focused on during the rest of this example.

The second use case illustrates the automated scheduling aspect of our proposed solution, again a worker will have to specify and maintain a schedule for the FTP file downloads and our scheduler will be responsible for initiating the download at these preset intervals. This use case could be implemented by MS Task Scheduler or other third party package and again will not be considered for the rest of this example.

The third and final use case represents the core of the process we are automating; the actor in this case is the scheduler which will initiate the procedure. For the sake of this example, the rest of my analysis and design will focus on the third download use case. From all of these use case models I can seek approval from the client or stakeholder and produce a formal set of requirements as shown in Figure 3 below.

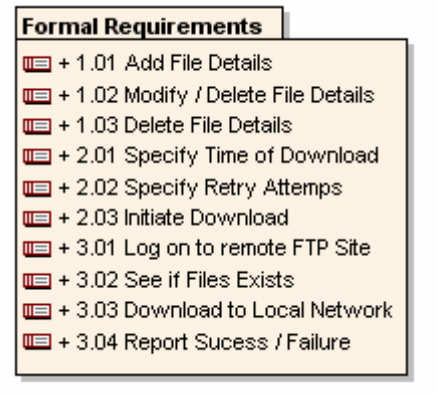

Figure 3: Formal Requirements

Basic Class Diagram

Now I have a formal set of requirements and a good understanding of the process I can now start to identify the static structure of the objects required to address the process.

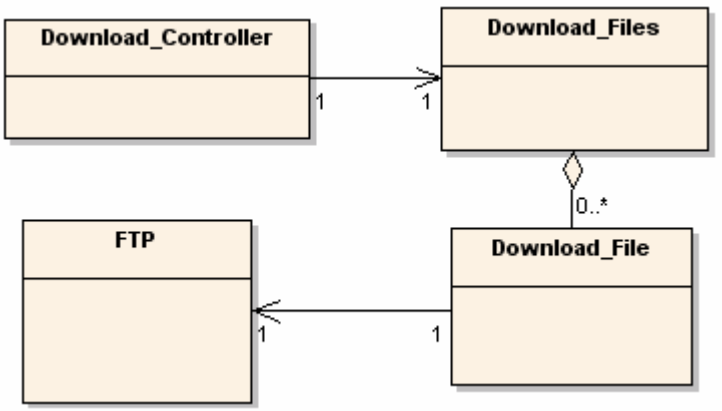

Figure 4: Class Diagram

The class diagram shown in Figure 4 above identifies the need for a Download_Controller class. The download controller would be responsible for issuing the download command and reporting back

the status of the download, it would also act as a container for all our download process functionality and could be implemented eventually as a COM component or an executable.

The Download_Files class would be associated to the Download_Controller and would act as a collection class managing many instances of Download_File classes, each of these would each possesses an associated FTP class.

Design

Now I have identified our basic classes and concluded our analysis for the time being, it is tempting at this point to continue by allocating behaviour to my classes and producing a more detailed class diagram. However before I plan to do so, I shall model more thoroughly the dynamic interactions and messages between our classes.

Therefore I plan to start the design phase with a sequence diagram and continue with a collaboration diagram. I shall then revisit the class model for a more detailed implementation. Furthermore, state and activity diagrams are extremely useful models to employee during the design phase. But since my example is so simple they are not really needed in this case and will be omitted from this example.

Sequence Diagram

The sequence diagram is great for helping us to allocate behaviour to our classes as it allows a graphical means of depicting the interactions and flow of events between my classes over time. One sequence diagram is typically used to show a user or an actor, the object and components they interact with in the execution of a use case.

Figure 5 on page 4 shows a sequence diagram for my file download use case and classes. We can therefore now see the interactions between classes over time and start allocating behaviour and responsibilities accordingly. In the sequence diagram each arrow represents a message between each class or entity in both directions, you can also see from the diagram when a class is instantiated. Sequence diagrams can actually go further than this and illustrate an objects lifetime by depicting when classes are destroyed.

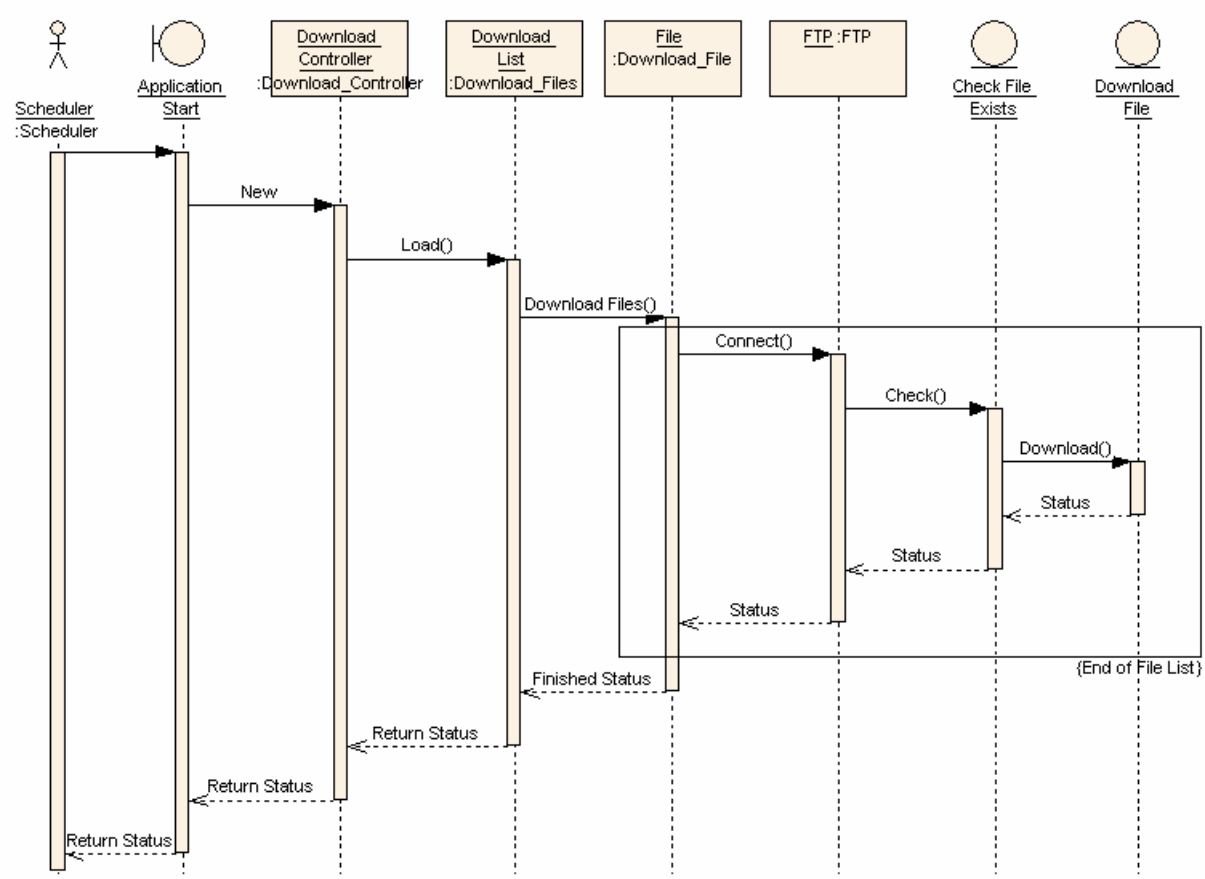

Figure 5: Sequence Diagram

Collaboration Diagram

A collaboration diagram is similar to a sequence diagram; it uses the same semantics, however instead of solely showing an explicit sequence of messages it also shows the relationship between the objects. Collaboration diagrams also provide a bird's eye view of a collection of collaborating objects and facilitate the modelling of the logic of the implementation of a complex operation. They are particularly useful when there is interaction between large numbers of objects.

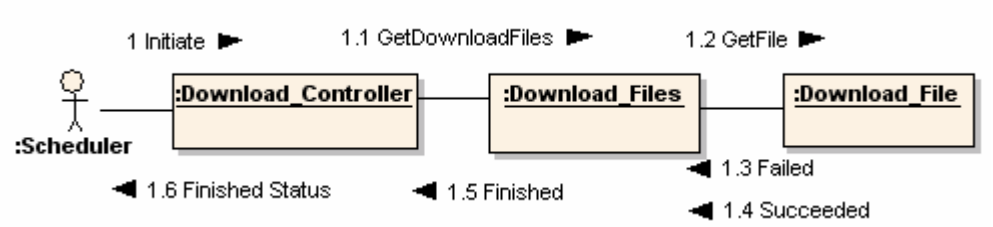

Figure 6: Example Collaboration Diagram

Detailed Class Design

I have now identified the messages and interactions between our classes using both the sequence and collaboration diagrams. Now I can start to add attributes (properties) and operations (methods) to my classes based on the dynamic models generated earlier. In practice I would be looking for

flaws in my design as I realise the implementation of my classes and when found my earlier models would be updated to reflect any changes.

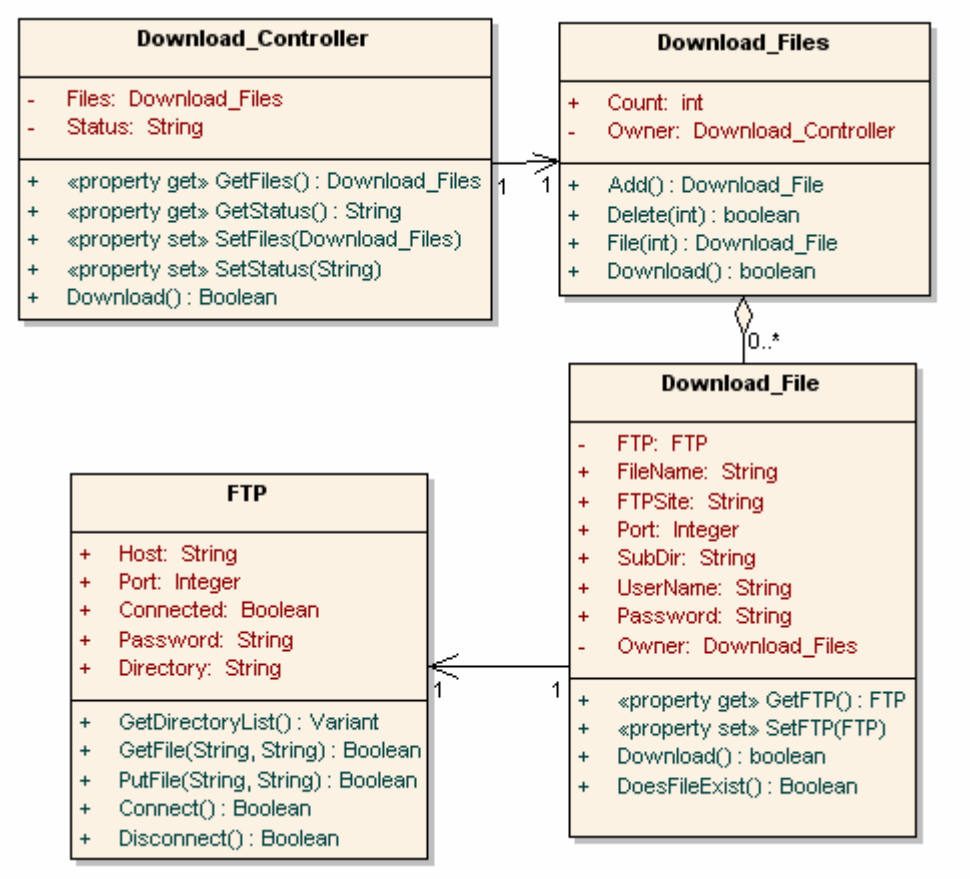

Figure 7: Detailed Class Diagram

From my detailed class diagram shown in Figure 7 above, we can see all the elements that were depicted right back at the analysis stage. With all facets of the analysis and design down on paper I can now craft my implementation without writing a single line of code. In this example I would use a third party FTP component to save the overhead of writing my own, the properties and methods of which I have added to the FTP class shown above.

Finally, since I used an appropriate case tool, such as Sparx Systems Enterprise Architect which was used to generate these examples I can go on and generate my source code, fill in the blanks, plug in my FTP component and I know that my implementation will work and satisfy all of the requirements, while being robust, re-usable and documented.

Conclusion

My aim through these two articles was to provide a gentle introduction into the UML, a rather potentially daunting subject and to provide an example which would allow us to see how we could benefit from its use and allow us to be able to apply it in a real life software engineering example.

I have used a basic set of models to try and illustrate that the UML is a notation that offers traceability and control but with no pre-defined process. The choice of which models to employ is specific to the situation at hand and dependant on what the analyst/developer and client will gain from the use of each model.

In short however, the use of UML allows us to craft our solution and communicate it with others before it is implemented and works in the same way an architect will draw up blue prints for a house before it is built.

Further Reading

UML Class Diagrams Example, sourceforge.net:
http://casetool.sourceforge.net/documents/tutorial/uml_tutorial.html
UML Summary Card, Embarcadero Technologies:
http://www.embarcadero.com/products/describe/UMLposter.pdf
UML Reference Card, Embarcadero Technologies:
http://www.holub.com/class/oo_design/uml.pdf
UML Resource Page, OMG:
http://www.omg.org/uml/