# CS 255 Model Application Short Paper

Paul Kenaga

Paul.kenaga@snhu.edu

Southern New Hampshire University

## Process Model Application

During requirements structuring, a process model in the form of a data flow diagram (DFD) can be applied to a design for the DriverPass scenario to depict the movement and transformation of data between components within a system and between a system and its setting. In the DFD, data sources would represent the origins and destinations of data that flow within the to-be system, processes would represent the functions that perform work on data, and data stores would represent data at rest. Labeled arrows would be used to represent data in transit and its movement throughout the system between data sources, processes, and data stores (Valacich & George, 2019).

The data sources for the to-be DriverPass system would be Customer, Driver, Secretary, IT Officer, Admin, DMV, Cloud Provider, and Microsoft. The data stores would be Account Information, Reservation, Online Class Content, and Packages. The Customer, Driver, Secretary, IT Officer, and Admin data sources send login data to the login process to access the system and can send their email to the reset password function that would send a reset password link back. The Customer data source sends personal information as data to the registration process to create an account or to the modify process to change account details. These functions would access and update the Account Information data store in order to execute.

After the login process is successfully completed, the Customer data source would send order data to the purchase process to buy a driving package offered by DriverPass. The order data originates from the browse packages process that accesses the Packages data store. After purchasing a driving package, the Customer data source would send requests to execute functions like take practice test which is linked to the Online Class Content data store or make, modify, and cancel a reservation that access the Reservation data store. The Customer data source would be able to send queries to the contact DriverPass function that are then sent to the Secretary data source. The Secretary data source would be able to send responses back to the Customer data source through the contact DriverPass function. The Secretary data source would send customer information to the registration process on behalf of new customers and perform the make, modify, and cancel a reservation function upon request.

The Driver data source sends driver notes to the Account Information data store through a process to update the comments for a customer. The Driver, Secretary, Admin, and IT Officer data sources would all be able to send requests to the display schedule function to query the Reservation data store for a schedule of reservations. The Admin data source would send requests to the download and print reports function to retrieve activity reports from the Reservation and (possibly) Account Information data stores. The Admin and IT Officer data sources would also be able to perform functions like block access to a user and reset an account through requests. The DMV, Cloud Provider, and Microsoft are data sources that send updates to processes that notify the IT Officer data source. The IT Officer Data source would send content updates to a function that would access and update the Online Class Content data store. The IT Officer data source would also send requests to the disable package function to access and update the Packages data store as needed.

The implementation of a DFD in the DriverPas scenario would provide a high-level overview of how external entities will interact with the to-be system and a conceptual understanding of the flow of data in it. The DFD would be decomposed from a level-0 diagram to a primitive level to ensure that all conditions necessary to satisfy the completion of the project are met.

## Object Model Application

An object model would be applied to a design for the DriverPass scenario through the use a class diagram to illustrate the internal structure of the to-be system and relationship between objects within it. The class diagram would consist of User, Customer, Driver, Secretary, IT Officer, Admin, Test, Package, and Reservation classes. Each class would be made up of attributes and methods that describe an objects state and behavior. Relationships between objects would be established through connecting lines and arrows that indicate associations and numerical representations that indicate multiplicity (Valacich & George, 2019).

The User class would be an abstract class that cannot be instantiated. The User class has username, password, name, photo, and contact information attributes. The methods in the User class are login, reset password, register, and modify account. The Customer, Driver, Secretary, IT Officer, and Admin classes are concrete classes that extend from the User class. The Customer class has an additional attribute for payment information. The Customer class can perform methods like purchase a package, take online test, make a reservation, modify a reservation, cancel a reservation, and contact DriverPass. The Driver class would have a method for adding driver notes. The Secretary class would have methods for making, modifying, or canceling reservations and to display the reservation schedule. The IT Officer class would have methods for updating the system, disabling packages, blocking account access, resetting accounts, displaying the reservation schedule, and updating online course material. The Admin class would have methods for retrieving a report, printing a report, modifying a report, blocking account access, resetting accounts, and displaying the reservation schedule. An abstract internal user class could be used to avoid duplicity of functions and simplify the design.

The Reservation class would have attributes for the customer’s name, driver’s name, time, car, and drop-off/pick-up location. Multiplicity would show that a customer and driver object can have zero to many reservations, but a reservation object will only have one customer and one driver. The Test class would have attributes for name, status, time taken, and score. The Test class would have a method for updating the status. A customer object may have zero to many tests but a test would belong to one customer. The Package class would have attributes for behind the wheel lesson hours, in-person lesson hours, online classes, and tests. The Package class will consist of zero to many test objects and the Test class will belong to one to many package objects. A customer object may have zero to many packages, but a package would belong to one customer.

The application of an object model in the DriverPass scenario would focus on objects that exist in the to-be system through classes and provide a detailed view of their attributes and the operations they perform. It would be important for the object model to show relationships between classes through associations and multiplicity. Aggregation and composition would be used to show stronger forms of association between classes.

## Process and Object Model Comparison

The advantages of the process model for the DriverPass scenario are that it would show where the data in the to-be system originated from, how data flows through the system, where data is transformed, which function transforms data, where data is stored or sent, and who or what interacts with the system. Another advantage for the scenario is that a process model would help capture some of the functional requirements from DriverPass and facilitate the system’s development. A process model would also provide analysts with a high-level outline of the to-be system’s business processes that can later be used to identify use cases and actors for development of behavioral models. A process model is advantageous for the external view of system behavior that is not present in an object model. The disadvantages of using a process model for the DriverPass scenario are that it does not give insight to the definitions, behavior, or interrelationships of data in the system. It would also not represent multiplicity or associations between entities that interact with the system. System analysts would not fully understand process logic from the model without decision tables (Valacich & George, 2019).

The advantages of an object model for the DriverPass scenario are that it would provide a graphical representation of the system’s structure, show the characteristics of data in the system, and illustrate the relationships between internal entities that interact within the system. An object model would allow system analysts to articulate the components of the system that support the business processes through objects which can facilitate the development of application logic. An object model for the DriverPass scenario would also be used to create a behavioral model to show how the state of an object changes. The disadvantages of an object model for the DriverPass scenario are a lack of detail into the storage of data, representation for external entities that interact with the system, and processing logic. Unlike a process model, an object model would not provide representation for data stores where data like account information is kept and it is not that clear how exactly data it is used by the functions in the classes that encapsulates it (Valacich & George, 2019). External entities like the DMV and Cloud Provider would also not be represented in an object model for the DriverPass scenario.

It is recommended that an object model be used in conjunction with a process model for the DriverPass scenario. Together they can be used to help create a behavioral model, like a Sequence diagram, that represents the different behaviors of the system, displays the interactions actors and objects, and show changes in the state of the system.

## References

Valacich, J. S., George, J. F. (2019, January 3). *Modern Systems Analysis and Design*. Hoboken, NJ: Pearson.