Some model driven software development approaches

MSc. Shaqir SULAJ[1]
Advisor: Prof. Dr. Petraq Papajorgji

Abstract

A model is a simplification of a reality, an abstraction, which neglects all of the irrelevant aspects of a software and focuses on the ones which define the software. Through modeling, we can visualize the software in a way which is understandable from both computers and people. Unified Modeling Language (UML) is a language, which stands in a higher level of abstraction than any programming language and through it, we can build software models from the simplest to the most complicated ones. However, even though through UML we are able to design software's and to generate a considerable amount of code, still we need to write tons of lines of code in a certain object – oriented language. In order to be totally independent from the execution environment and to focus entirely in business logic as what really defines a software we need to go further. To achieve this goal, the executable UML and state machines concept is introduced. "An executable UML model is one with a behavioral specification detailed enough that it can effectively be run as a program" (Seidewitz, 2011). Through executable model a higher programmer productivity could be achieved and the system is not affected by the year after year change in development technologies, which require that the system to be upgraded each time there are changes in run-time environments. On the other hand, through state machines we are able to simulate the execution of a program through the model. Thus we can develop information systems using

[1] Shaqir Sulaj is graduated in "Business Informatics" at the European University of Tirana in 2020. In 2022, he is graduated from European University of Tirana in 'Software Engineering", profile "Data management". During his studies, Shaqir had his first working experience as Quality Assurance Specialist in the private sector. From March 2022, he is part of the internal staff of Raiffeisen Bank Albania as Software Developer at the IT Division.

techniques which stand in a higher level of abstraction and platform independent that can be translated in whichever specific platform of implementation.

***Key words**: model driven development, software, state machines, executable UML*

The introduction

In the brief history of information systems, besides different approaches regarding the analysis and development of these systems, we can all agree that the progress in the field has been tremendous. All this innovation has been driven by the continuous growing demand for systems which manage, save and handle the massive data which are generated from our daily life activities. In order to develop systems which, meet the user requirements effectively it is important for the system to be specified correctly. A way to achieve this goal has been proven to be software modeling. The model is a simplification of reality, which helps us have a better understanding of our software. It serves as a blueprint for the further stages of the software development. The most accurate tool regarding modeling is the Unified Modeling Language. UML provides a modeling language which is understandable from both humans and machines. Because the model it is organized visually in diagrams and a considerable amount of code is generated directly from the model.

However, the model specifies the software from a static point of view and it is still necessary to translate the model in a specific implementation platform to represent the detailed behavior of our system. In order to escape this bottle-neck the new executable UML it's introduced. "An executable UML model is one with a behavioral specification detailed enough that it can effectively be run as a program" (Seidewitz, 2011). Another solution which supports expressing detailed behavior in a model is the concept of state machines. In this paper, through the theoretical analysis, but also through practical examples, the standard of the executable UML and state machines will be explored.

Aims

This paper aims to represent the advantages that model driven development offers in the software development industry. Thesis aims to:

a) To argue the way which executable model improves software development processes.
b) To represent through concrete examples the benefits of software development in a higher level of abstraction.

c) To argue how the model-based development increases programmers' productivity in a dynamic software development market.
d) To reason how model driven development offers a link between business requirements and programming languages.

Objectives

This paper objectives are:

1. Building two software prototypes based on a visual model which is easily readable, modifiable and reusable using executable UML and state machines technologies.
2. To emphasize the aspects in which developer's productivity is improved through all the stages of software development life cycle.
3. Developing an information system which is totally independent from programming languages and translatable in each object – oriented programming language.
4. To provide a documentation for the system which will be developed that resists throughout time.

Analysis

Model Driven Architecture

As the name suggests, in the model driven architecture in the main scope of development process; it is the model. On the contrary from model driven architecture, the traditional software development passes through several stages such as functional requirements analysis, feasibility analysis, design, coding, testing and development. This process besides long, can be stressful and requires a lot of working hours. The reason for this it is because it is necessary a journey from the high level of functional requirements analysis to the technical low level of abstraction, which is coding. "Models are the stepping stones on the path between a description of a business need and the deployable runtime components of its solution." (Brown, Iyengar, & Johnston, 2006). "Within MDA the software development process is driven by the activity of modeling your software system" (Kleppe, Warmer, & Bast, 2003).

The benefits of model driven architecture

Taking into consideration the commercial success of programming languages and frameworks based on them, it is fair for one to think, which are the benefits of model driven architecture? In this type of architecture, we have two main benefits, which are productivity and portability. This approach is more productive since a model which is independent form the implementation platform avoids the development technicalities embeded in a specific platform and it is focused on developing solutions for business requirements. The other benefit, portability makes it possible that a system based on the platform independent model to be translated in every platform specific model. Furthermore, the translation into a PSM, it is much faster and feasible, since a considerable amount of code it is generated automatically from the platform independent model. "The main idea in MDA it is building detailed enough models and to use transformations to generate the most part of the code" (Starr, 2002).

Model Driven Development

Model driven development includes several processes, which start from creating a model to delivering an executable code. "MDD is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations." (France & Rumpe, 2007). According to this approach, model driven development is a series of transformations, which have as an input a model and as an output another more detailed model, until the software implementation is achieved. For example, if we want to design a software, firstly we have to define the objects which will be part of the model and also, we have to define the state and behavior of these objects. Furthermore, model after model, the state and behavior of the object becomes more and more detailed, to the point which the information system it is completely developed based on the model. "The benefits of adopting MDD include reduced software development time, enhanced code quality, and improved code maintenance." (Chao, et al., 2006). On the other hand, another important aspect of the model driven software development approach, it is the process of model driven software testing. "Model Driven Testing is an approach based on MDD in which tests can be generated from development models in an automated way through the use of transformations" (Baker, Rai, Grabowski, Haugen, & Williams, 2008). Imagine a software development process, which goes through all the stages of software development life cycle. The modeling process it is right after functional requirements analysis, meanwhile testing according to the traditional method of testing is a step before deployment. If it is the case that in testing serious issues are identified regarding business logic of the information system, there would be unimaginable consequences of wasted working hours and resources. MDT approach mitigates this problem by testing the software based on the model, which is developed.

Executable UML

In the model driven architecture there are two main behavior modeling approaches. One of these is elaborationist approach, which uses formal languages to manage the definition of object's behavior.

"The second approach is referred to as translationist; the complete system should be defined within the PIM." (Papajorgji & Pardalos, 2016). "In the elaborationist approach, the definition of the application is built up gradually as you progress through from PIM to PSM to Code." (McNeile, 2003). In order to make this work "in 2008, this led to the adoption of the Foundational UML (fUML) specification, providing the first precise operational and base semantics for a subset of UML encompassing most object-oriented and activity modeling" (Seidewitz, 2011). However, this was still not enough to support full specification of object's behavior in an UML diagram. To properly define the methods through which objects would operate and implement their behavior, you would have to draw detailed activity diagrams, which take a lot of time and effort and they are not effective. So, it became necessary to apply a language for objects interaction and this language was Action Language for fUML or Alf.

> "Alf is basically a textual notation for UML behaviors that can be attached to a UML model any place that a UML behavior can be. For example, Alf text can be used directly to specify the behaviors of a state on a state machine, the method of an operation or the classifier behavior of an active class. Further, the "extended" Alf notation actually includes some basic structural modeling constructs, so it is also possible to do entire models textually in Alf" (Seidewitz, 2011).

Since, semantically Alf marks a subset of fUML, the last one could me interpreted as an execution environment for Alf. "The goal is to make UML modeling executable modeling, to allow designers to test and verify early and to generate 100% of the code if desired." (Sunye, Pennaneac'h, Ho, Guennec, & Jezequel, 2001). "Indeed, the real power of executable modeling going forward relies on keeping the entire behavioral specification at such a higher level of abstraction." (Seidewitz, 2011).

State machines

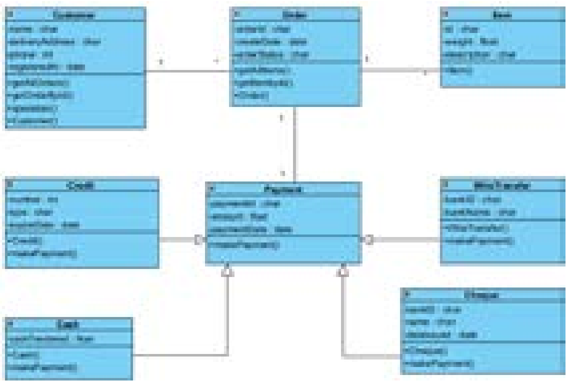
According to object-oriented paradigm, an object it is characterized from two significant components, which are state and behavior. To bring in memory, state it is one of the many situations in which an object might find itself, as for behavior it represents how an object behaves and reacts to other objects requests. So, we could say that the state of an object it is a static view of an object defined at a certain moment, meanwhile the behavior includes all the dynamics, which affect the state of the object or which the object affects other object's state. But how the change of these states happens, which are the situations which trigger the process of changing the state of an object? To get an answer we need to include the concept of state machines. "A state machine is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events." (Booch, Rumbaugh, & Jacobson, 1999). Events which trigger the change of a state might include signals, method calls or time passage. Depending from the happening of the events, an activity will be triggered, which will affect the actual state of the object. Activity represents an action, which impacts the state of an object.

Simulation

Executable UML prototype

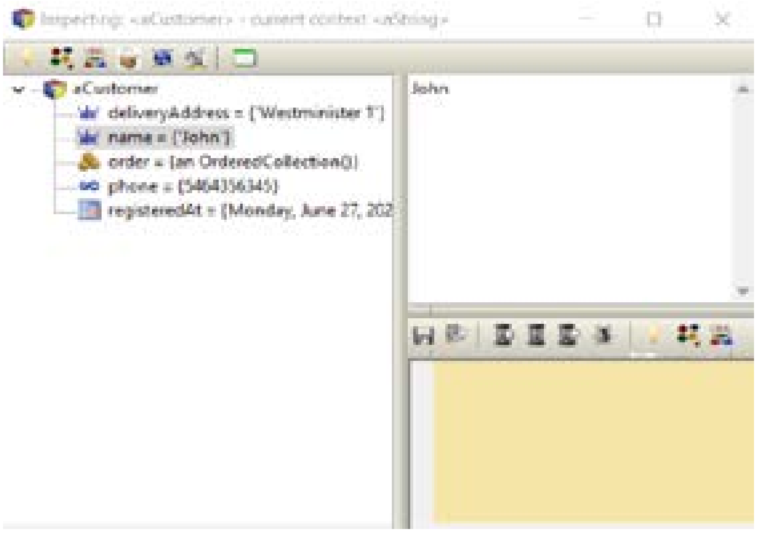
In order to create a better understanding of the executable UML standard, I will be demonstrating and executable UML diagram. This example will represent an e-commerce executable class diagram as it is shown below:

FIGURE 1. Class Diagram of E-commerce system



After the design of the class diagram for the e-commerce system, next we will execute the model in the UML Almighty environment. UML Almighty is a runtime environment specialized for the UML executable standard. In order to run the program, we have to define a class which will serve as a reference for the executable UML prototype. In this study case, since the center of activity is the Customer class, will serve as a reference to execute the application. In the picture below, it is displayed the initialization of Customer class:

FIGURE 2. Customer class instance



Inspecting: «aCustomer» - current context: «aString»

aCustomer
deliveryAddress = ['Westminister 1']
name = ['John']
order = [an OrderedCollection()]
phone = [5464356345]
registeredAt = [Monday, June 27, 202

John

Using the powerful abilities of simulation in the UML Almighty, it is possible that right after we design the model, to be able to create class instances. All this, without having to select as PSM environment and without entering in technical details of a specific development platform. The standard of executable UML provides us with a prototype if the e-commerce system, in which we can test the dynamic aspect of UML model. This ability is quite beneficial because "studies show that modelers, often create models that do not quite reflect how the system will behave" (Farah & Lethbridge, 2007).

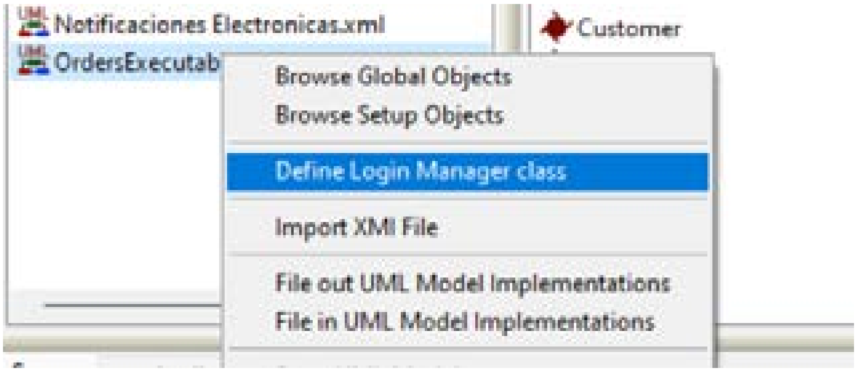
However, as it is known, the class diagram's goal is to address the static point of view of a system. So, how can we address the dynamic aspect of our e-commerce system, keeping in mind that UML diagrams are not detailed enough to define behaviors and operations which characterize objects. In order to make this happen, we will be using an action language which does the detailed specification of operations.

```
Customer : = Customer newInstance.
Customer initialize.
Customer name : 'John'.
Customer deliveryAddress : 'Westminister 1'.
Customer phone : '464356345'.
Customer registeredAt : '27/06/2022'.
```

Action Language for UML Almighty it is similar to pseudocode and it is high level language close to the human logic.

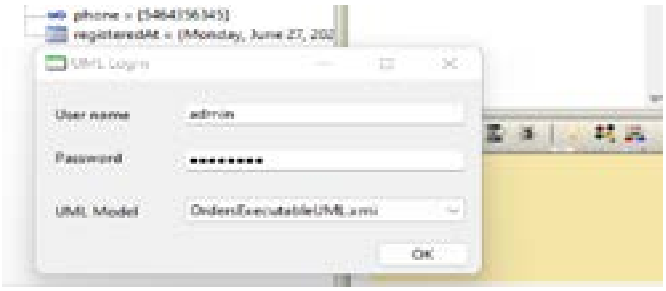
After the creation of a client, the next step into the execution of the e-commerce system prototype it is the login. In order to login, we have to define a class, which will serve as a login manager, much or less like the class which contains the public static void main() method in object-oriented languages such as Java or C++.

FIGURE 3. Defining login manager class


Notificaciones Electronicas.xml
OrdersExecutab
Customer
Browse Global Objects
Browse Setup Objects
Define Login Manager class
Import XMI File
File out UML Model Implementations
File in UML Model Implementations

Customer class will serve as a main class to our system and now we are ready to test the client's account created earlier. UML Almighty provides two options for the executable UML prototype, one is web based and the other desktop based. In this study, it is used the desktop prototype, in which the user through his account will be able to login, choose products, create orders and make payments.

FIGURE 4. Login

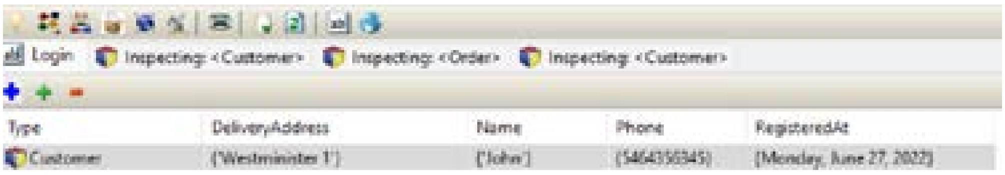

phone = [5464356345]
registeredAt = [Monday, June 27, 202

UML Login

User name	admin
Password	••••••••
UML Model	OrdersExecutableUML.xmi

After the login, the customer data will appear in the profile:

FIGURE 5. Client profile



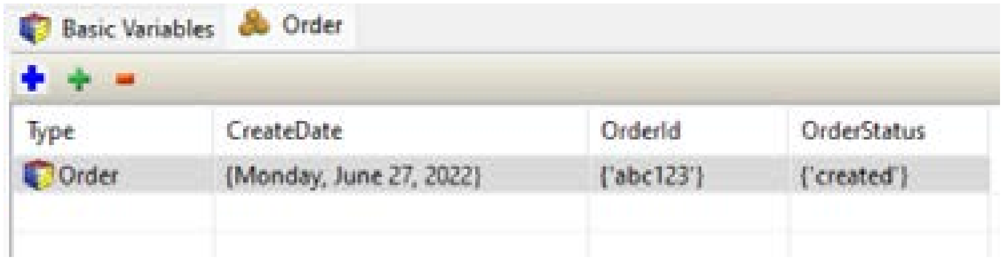
Login Inspecting: <Customer> Inspecting: <Order> Inspecting: <Customer>

Type	DeliveryAddress	Name	Phone	RegisteredAt
Customer	('Westminister 1')	('John')	(5464356345)	(Monday, June 27, 2022)

The client,, in this menu has the possibility to revise his data like name, address, phone number and registered date. The number of data attributes which could be saved for a client is limitless, however for the sake of this simulation only the most esential data are saved. Also, the client can change his data at anytime.

Another functionality of the e-commerce system is adding orders. This feature is set in the Order menu as shown below:

FIGURA 6. Orders



Basic Variables Order

Type	CreateDate	OrderId	OrderStatus
Order	(Monday, June 27, 2022)	('abc123')	('created')

The Order menu displays the data like id, status, creation date, to create orders the script below it is used:

Order : = Order newInstance.

Order initialize.

Order OrderId : 'abc123'.

Order OrderStatus : 'created'.

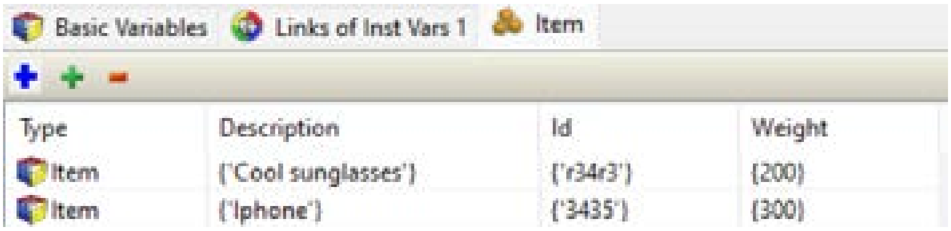
Order CreateDate : '27/06/2022'.

Customer class has a 1 x N relationship with Order class, thus an instance of Customer class, a client has a collection of objects or instances of Order class. In order to create the relationship between the client and his order we have to use the action language as below:

Customer add : Order.

After the order is created, we add products into the order. Products are instances of Item class and hold the necessary information for each product in the e-commerce system. Products are related with orders in a 1 x N relationship, where in an order we might have many products, while a unique product it is only in an order. In the example, we have the menu which displays the product information:

FIGURE 7. Products



Basic Variables | Links of Inst Vars 1 | Item

Type	Description	Id	Weight
Item	('Cool sunglasses')	('r34r3')	(200)
Item	('Iphone')	('3435')	(300)

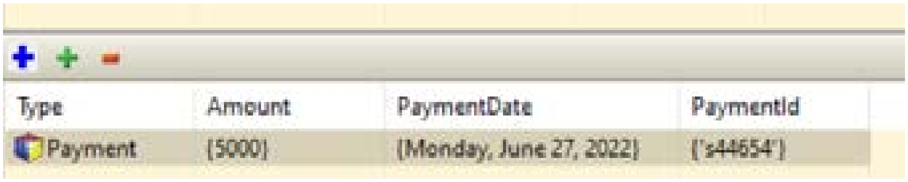
This order products store the data for the product id, description, weight. We shall use the script below for adding these products:

Item : = Item newInstance.
Item initialize.
Item Id : 'r34r3'.
Item Description : 'Cool Sunglasses'.
Item Weight : '200'.

Item : = Item newInstance.
Item initialize.
Item Id : '3435'.
Item Description : 'Iphone'.
Item Weight : '300'.

These product instances shall be added to the order:
Order add : Item.
The other use case to be specified it is the payment process in the e-commerce system. Payments will be executed in the related menu where information regarding payment method, payment id, payment date and amount will be stored. Each payment it is made for a certain order since we specified in the model that the relationship between payment and order it is 1 x 1 and different payment methods are implemented using the polymorphism principle with an interface.

FIGURE 8. Payment



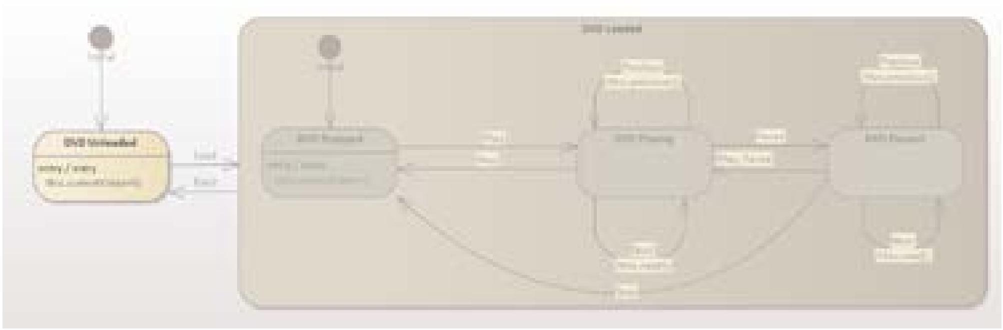
Type	Amount	PaymentDate	PaymentId
Payment	(5000)	(Monday, June 27, 2022)	('s44654')

From the UML model execution, we can realize that the model is designed correctly and the information system behavior is according our expectations. E-commerce system simulation was achieved in less time using a low code approach. The simulation through the executable model has the advantage of discovering errors of the solution design since in the beginning and this helps reducing costs which come from an incorrect modeling of the system. In addition, a model driven developed system lets us focus more in the business logic rather than in the specific implementation platform. Coding itself, in whichever programming language it is just a tool which helps us to build software. The key in offering a reliable solution, universal and promising to resist in time it is to focus in the business logic. That is the aim of the executable model, to shift the focus in problem solving and not in the thousand different ways there are to solve them.

State machines prototype

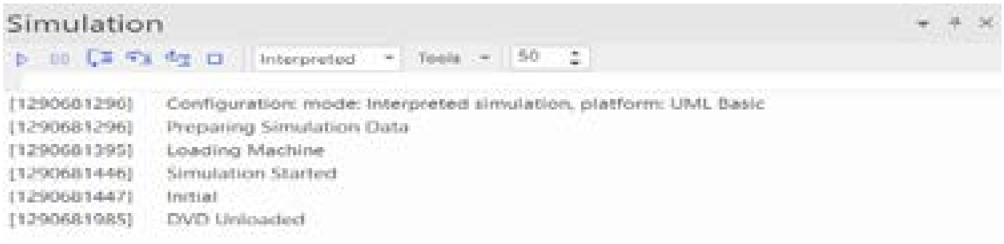
For the state machine prototype, a solution design it is implemented, which will serve as a model for Video Player embedded system.

FIGURE 9. State machines simulation



After the design of the state machines model for the Video Player, the model will be transformed into an executable state machine. In the Enterprise Architect's menu, in the Develop tab, button Generate allows us to generate the code for the executable state machine in several object-oriented languages such as C#, C++ or Java. After the code is generated, we select the option start simulation. In the moment that the DVD is Unloaded we could choose the event DVD Load to put the DVD inside the Video Player.

FIGURE 10. Begin Simulation



```
[1290681296]    Configuration: mode: Interpreted simulation, platform: UML Basic
[1290681296]    Preparing Simulation Data
[1290681395]    Loading Machine
[1290681446]    Simulation Started
[1290681447]    Initial
[1290681985]    DVD Unloaded
```

As we can see in the simulation menu at the moment in which we begin the simulation of the state machines, we can track the execution flow and as soon as we start the simulation we transit to the initial state DVD Unloaded. When we click Load, the code generated in Java is responsible for the transit between states process. The logic in the code uses three main concepts in state machines which are states, transitions and signals. The Java code stores an enumeration of all states and transitions which are specified in the diagram and generates a respective method for each transition between the states.

```
    private void DVDUnLoaded__TO__DVDLoaded_2221 ( Signal signal,
StateData submachineState ) {
                if ( m_StateMachineImpl == null )
                        return;
                if(                        !m_StateMachineImpl.GetStateObject(
submachineState,StateEnum.DVDPlayer_ENUM_STATEMACHINE_
DVDUNLOADED.ordinal() ).IsActiveState()){
    return;
    }
    StateProc(StateEnum.DVDPlayer_ENUM_STATEMACHINE_
DVDUNLOADED, submachineState, StateBehaviorEnum.EXIT, null);
    DVDUnLoaded__TO__DVDLoaded_2221_effect(signal);
    m_StateMachineImpl.currentTransition.SetActive(m_StateMachineImpl);
         StateProc(StateEnum.DVDPlayer_ENUM_STATEMACHINE_
DVDLOADED,    submachineState,    StateBehaviorEnum.ENTRY,    signal,
EntryTypeEnum.DefaultEntry);
    }
```

The method above takes as parameters the signal or the event which initiates the transition process and the data of the actual state. If the implementation of the state in a state machine is not initialized, the method does not continue any further with the execution of the transition process. If the actual state of the state machine, it is not active again the execution comes to a halt. If the simulation process passes those conditions the method calls the actual state and the method which executes the transition process using the signal as a parameter which triggers this process, which in our case it is the Load button. This way the state machine completes a full transition of state from DVD Unloaded to DVD Loaded. So on and so forth, other transitions between states and sub-states continue to happen over and over again through the life cycle of an object.

Conclusions

Conclusions

Through this paper, secondary sources analysis and simulation through the prototypes of executable UML class diagrams and state machines, it was concluded that:

- Model driven development supports software development in a manner which is fast, accurate and independent from the implementation platform. This development approach shifts the focus from the specific platforms of implementation and emphasizes the business logic as the backbone to provide an efficient software solution.
- Model driven development approach increases the developer's productivity since the model creates a better understanding of the system which is being developed and a considerable amount of code it is generated quickly and accurately directly from the model. Furthermore, information systems which are built according to PIM methodology are universal and portable from the platform point of view. This means that a model driven developed system can be implemented in whatever specific platform which is required.
- The executable UML model supports software testing right after the model is designed, giving the opportunity to verify if the information system behaves as expected since in the early stage of development. This advantage reduces costs, mitigates errors in development and increases the development process efficiency.
- Model driven development creates a precise, whole and understandable documentation of the information system. Because this approach provides us with a model which visualizes in details each functionality of the system, hence offers a valuable documentation.

Suggestions

Taking into consideration the potential and benefits which derive from model driven development, which were evidenced throughout this study and the simulations of the prototypes of executable UML and state machines, some of the suggestions recommended are:

- Applying model driven development approach as a standard practice in every project which includes software development.
- Researching the usage of editors, which support executable UML and the expansion of their capacities to build and execute models with detailed enough behavior to be ran as a program.
- Developing software which are independent from the implementation platform in a higher level of abstraction from the programming languages of our present. As Booch said, the idea of developing software in C++ or Java, in a near future will sound as absurd as it is nowadays to write an application in assembly.

Bibliography

1. Baker, P., Rai, Z. D., Grabowski, J., Haugen, O., & Williams, C. (2008). *Model-Driven Testing Using the UML Testing Profile.* New York: Springer.
2. Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language, User Guide.* Addison Wesley.
3. Brown, A. W., Iyengar, S., & Johnston, S. (2006). A Rational approach to model driven development. *IBM SYSTEMS JOURNAL*, 463-480.
4. Chao, T., Chen, S.-k., Dikun, M., Lei, H., Jeng, J.-J. (., Kapoor, S., . . . Zeng, L. (2006). Model Driven Development for Business Performance Business Performance. *IBM SYSTEMS JOURNAL*, 587-605.
5. Farah, H., & Lethbridge, T. (2007). *Temporal Exploration of Software Models: A Tool Feature to Enhance Software Understanding.* WCRE 14th Working Conference on.
6. France, R., & Rumpe, B. (2007). *Model-driven Development of Complex Software: A Research Roadmap.* IEEE.
7. Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture : Practice and Promise.* Addison Wesley.
8. McNeile, A. (2003). *MDA: The Vision with the Hole?* . Retrieved from www.metamaxim.com: http://www.lcc.uma.es/~av/MDD-MDA/publicaciones/P_7MDA-the%20vision%20with%20the%20hole.pdf
9. Papajorgji, P. J., & Pardalos, P. M. (2016). *Software Engineering Techniques Applied to Agricultural Systems, An Object-Oriented and UML Approach, Second Edition.* Springer.
10. Seidewitz, E. (2011, January 19). *https://modeling-languages.com/new-executable-uml-standards-fuml-and-alf/.* Retrieved from https://modeling-languages.com/: https://modeling-languages.com/new-executable-uml-standards-fuml-and-alf/
11. Starr, L. (2002). *Executable UML: How to Build Class Models.* Prentice Hall.
12. Sunye, G., Pennaneac'h, F., Ho, W.-M., Guennec, A. L., & Jezequel, J.-M. (2001). *Using UML Action Semantics for Executable Modeling and Beyond.* Springer.