TEMANIS: Towards an Environment for Modeling Adaptation Needs for Information Systems

[1]Magagi Ali Bachir, [1]Ismail Jellouli, [2]Said El Garouani, [1]Souad Amjad
[1]Department of computer Science, Abdelmalek Essadi University,Tetuan,Maroc
[2]Department of computer Science, Sidi Mohamed Ben Abdellah University, Fez
*Corresponding Author: bmagagi@uae.ac.ma

***Abstract*— In this paper we present a tool for expressing the adaptation needs of information systems. The main objective is to provide a modeling environment allowing the integration of all the elements required for adaptation. We first make a presentation of the state of the art in the field. We define a UML profile applicable to the activity diagram. This activity diagram expresses adaptation needs. Following, we outline the structure of our solution and the way it works. Our solution is composed of three parts, each one representing a phase of the adaptation process. At the end, an example scenario shows a possible application of our solution.**

Keywords—Model-Driven-Engineering, Self-Adaptive System, Model Transformation, Automatic Code Generation, Model-Driven-Approach

I. INTRODUCTION

Adaptation is a crucial need in almost all areas of life. Any serious system must include aspects of adaptation taking account of its environment. This aspect can be related to the technical structure of the system or related to its internal behavior. Adaptation guarantees a high level of autonomy in terms of maintenance and error management. Human intervention on the system therefore becomes rare or absent. Nowadays, there is no universally recognized method for designing and building this type of systems. Several researchers have been interested in this problem and have proposed new solutions. Although very interesting, these solutions do not seem to cover all aspects raised by this problem. Some limit themselves to solving a specific point of the problem, others set limits on the application domain to optimize the management of a specific domain, others provide tools specific to a single technology. This leaves the door open to all other possible solutions. These solutions are generally inspired by the behavior of existing systems, whether natural like the biological system, or artificial like some sophisticated technical systems.

On the other hand, our study of the state of the art shows that the standard language UML [1] does not allow a precise design that supports adaptation; all technical specifications related to adaptation must be considered and formalized by the designer, which is costly in terms of time. Thus most of the approaches we have explored aim to provide tools specializing the UML language [1] so as to make it more compliant with the domain or with the technology. That is why we formalize our approach in the form of a domain-specific modeling environment to facilitate the design of adaptive information systems.

Following the previous consideration, the solution we propose is based on the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) [2] control loop which is a well-established functional architecture for adaptive systems. Structurally, the system implementing the MAPE-K, adopts the Model-View-Controller (MVC) architectural pattern. The dynamic response of the proposed system is insured by the successive replacements of components. This principle consists of replacing degraded components by newly generated ones.

This methodology aims to support the "Generic Model for Adaptive Systems Realization"[3] and "Domain Specific Model Driven Approach for Adaptive Systems"[4], in terms of adaptation needs expression. Other methodologies can also take advantage of it to help express adaptation needs of system they are designing.

The remainder of this document is structured as follows:

In section 2 we present a state of the art in which we outline the field of study and its techniques. In section 3 we present an application case scenario and conduct a comparison between its resolution with the coding method and its resolution with model-driven engineering. We present the definition of the tools to configure the environment, in section 4. Then in section 5, we present the approach and how it works. Then in section 6 we present an application of the approach on the example of the field of application presented in section 3. At the end section 7, in which we present the synthesis of the results and the future work envisaged, comes to conclude the document.

II. STATE OF THE ART

We call an adaptive system a system capable of acting on itself to provide responses to environmental malfunctions. These modifications may consist of changing its configuration or structure.

The adaptive system is a system capable of evaluating its behavior, by itself, and of modifying it when the desired performances are not reached[5].

Depending on their perception of the environment, adaptive systems are able to change their behavior or their structures to respond to the hazards that have arisen[6].

It is pointed out by[7] that what defines an adaptive system is its ability to react to environmental changes and to its internal logic by itself.

Adaptation therefore amounts to making any system capable of responding to the operating problems (internal or related to its functional environment) that it encounters.

However, realizing this type of system in an efficient and predictable manner is a great engineering challenge. The prefix "auto" indicates that systems autonomously decide how to adapt or organize themselves to adapt to changes in their environments. While a self-adapting system may be able to operate without any human intervention, guidance in the form of higher level targets is useful and achieved in many systems[8].

This field is attracting the attention of many researchers, each trying to make their own contribution to advancing the field. As a result, we have a large number of approaches, each with its strengths and weaknesses. We propose, for a better reading, to explore some of the approaches in this part.

Originally the majority of adaptive systems are based on the operation of the control loop. The latter is the central element in the self-adaptation of a system. It gives the system the ability to constantly observe its environment in order to be aware of the change in the latter. Based on the control loop the system analyzes the situation of its environment after acquisition and takes a decision to provide the appropriate response to the environmental changes that have occurred. It is indicated by that the feedback loops (control loops) provide the generic self-adaptation mechanism [8].

But with the evolution of this kind of systems several sources of behavioral inspiration and various technical bases have emerged. However, they are not always used separately, they can sometimes be combined together to define approaches to adapting systems. The biological system because of its complex functioning has inspired many researchers in their work when it comes to the realization of self-adaptive systems[8].

- Architecture-based engineering, in this case all aspects of the approach are defined by extensions of a well-defined technical architecture[9].
- Component-based engineering, it ensures that the system gradually replaces degraded software components with new ones in response to environmental hazards [10].
- Service-oriented engineering, applicable to the service-based system, it leverages the services to ensure the adaptation of the system [11].
- Model-driven engineering, which puts the model at the heart of the entire software process. That is, the adaptation rests entirely on the model in the whole process of realizing the system. In this technique the initial models will be gradually modified using the transformation of the models to produce a system.

A. Model transformation

This is a crucial element in model-driven engineering because it is through which the process of building the system from model takes place. A source model will have to go through several levels of transformation[12] in order to produce a system.

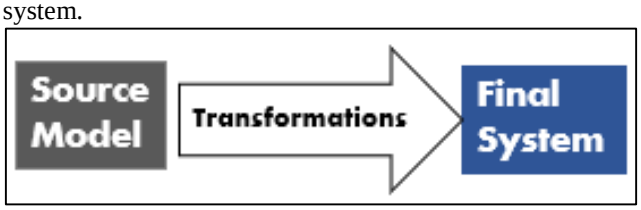

Figure 1:Tranformation concept view

Model transformation is the process of converting a source model into a desired destination model. It can relate to models of the same types or different types.

Transformations are indispensable techniques in software engineering since the beginning of high-level programming languages which are compiled into assembler[13]. Similarly, model transformations are therefore crucial in Model-Driven-Engineering(MDE)[14] and come in different forms to perform different tasks [15]. Transformation is a program that receives a minimum of one input model to provide an output model.

The transformation of models can be of several types so we can distinguish the Model-To-Model transformation from the transformation Model-To-Text, as widely discussed in [16].

Model-To-Model transformation: this is the type of transformation that encompasses the generality of model transformation because it allows to transform any source model into any destination model.

Model-To-Text transformation: this type of transformation is the most specific because it allows you to switch from a source model to a destination model which must necessarily be of text type, this is the case for code generation.

A transformation is called exogenous when the source and destination models are based on different metamodels. The following figure express that.

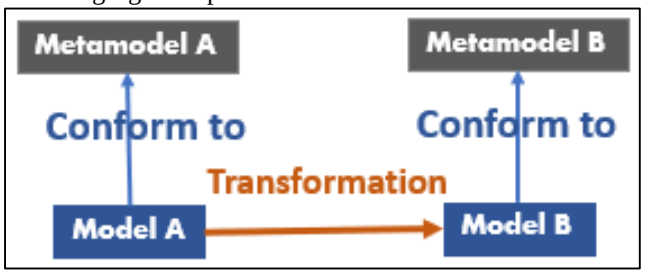

Figure 2:exogenous transformation

A transformation is called endogenous when the source and destination models respond to the same metamodel, like express in the following figure.

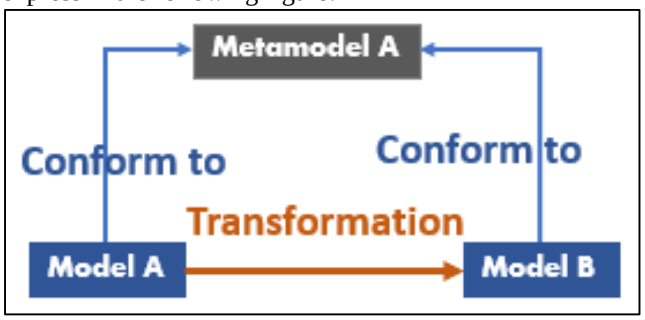

Figure 3:endogenous transformation

To perform model transformations, it is essential to use a model transformation language. There is a variety of model transformation languages, each presenting an advantage over the others depending on the point of view. Some languages are wider because it offers different types of transformation, so it can be used to make a Model-To-Model or Model-To-Text transformation, most languages are of this type. This is notably the case with Atlas Transformation Language [17] or even Query view Transformation[18]. Some are more restricted to performing Model-To-Text type transformations, for example Acceleo transformation language[19].

B. Modeling language and extension of UML

For better modeling specific to a domain, it is essential to define a language which is specific to it. This language thus makes it possible to have precisely all the elements of the field well represented. To do these, the two following solutions are possible:

- ✓ The definition of a new modeling language: this consists of defining new models and a meta-model ensuring the conformity of the models. It will also be necessary to define the abstract and concrete

syntax of the language for its use. The abstract syntax describes the set of valid models that can be used to model any system. It also describes the way in which the models are structured [16] and the different interactions that can exist between them. Concrete syntax then describes how models are presented graphically or verbatim in the modeling editor. It also presents the semantics of models. In other words, it specifies the meaning of each pattern in the design. This solution is very rigorous and time consuming because it consists in redefining all the foundations of a modeling language as a redefinition of UML.

- ✓ The definition of a UML profile: the profile is a domain-specific interpretation of UML, and can therefore be considered as domain-specific languages defined by extending or restricting UML. UML profiles are presented as packages of related and consistent extensibility elements defined based on UML and using elements like stereotypes, tagged values. Stereotypes allow you to refine meta-classes by defining additional semantics to the element represented by the meta-class. The marked values form a tag-value pair attachable to a stereotype as an attribute. They allow the designer to specify additional information useful or necessary to implement, transform or run the model[16].
  We can list the following examples: the EJB profile for describing the basic concepts of Enterprise Java Beans (EJB);the UMLTesting profile allowing to support the design, visualization, specification, analysis, construction and documentation of the elements participating in the tests; the SysML allowing modeling specific to the field of systems; Or SoaML, which is defined as a service-oriented architecture modeling language.

C. Domain specific modeling language

UML is generally the language used for modeling information systems. Nevertheless, it appears to be a very general language for the design of certain specific domains. It requires details to take into account the elements of the domain in the modeling.

Hence the need to implement domain-specific languages depending on the industry or the type of system to be designed. It consists in specifying generic models representing components specific to the domain. In this way we can accurately represent each component of the system relating to a given sector. Thus, the user can have the tools to carry out a coherent design in his field.

A domain-specific modeling language (DSML) is any language designed specifically for a specific domain, context, or company, to make it easier for people who need to describe things in that domain [16].

To define a domain-specific language, one generally relies on existing components in UML that can be reused in the target domain. On top of these UML components, domain-specific components are added through a UML profile. The following figure (figure 4) gives an overview which shows the correspondence between UML language and domain specific language.

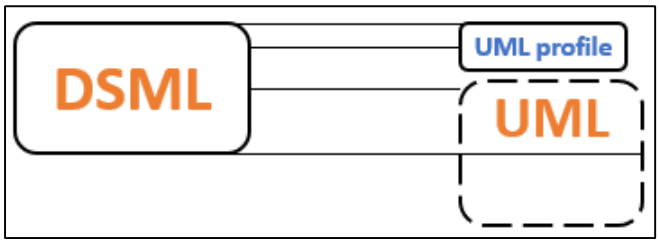

Figure 4:Correspondence between UML, UML profile and DSML

D. Previous approaches explored

We will give a non-exhaustive overview of the different approaches that we had to explore. We limit ourselves to model-driven approaches[20] since they go in the same direction as ours. We can thus present them as follows:

AdpatCase[21] which is a model-driven approach that is based on the structure of the control loop and the behavior of the uml useCase diagram to ensure the adaptation of systems; it allows expressing the adaptation needs of the system to design without taking charge of the implementation part.

ActiveForms[22] is another approach that represents a valuable contribution to the field of systems adaptation. It is a model-based approach that consists on the continuous verification of models. When the service provided by the system is interrupted or is no longer optimal, the models are gradually upgraded in order to stabilize the system.

Model-based Simulation at Runtime for self-adaptive Systems[23] is a structure-to-module approach in adaptation decision making. To ensure efficient decision-making, it offers a simulation of the execution of each model to identify its different states.

FESAS: Towards a Framework for Engineering Self-Adaptive Systems [24] : This is a Framework that provides tools for the design of systems and transforms the source model into a system model, using a reference architecture for the adaptive system and a library with reusable components for the adaptation logic.

In the design phase, requirements, objectives and constraints are captured using a requirements engineering approach suited to adaptive systems development [25]. The design model can then be automatically converted to a runtime system model. Work at the design level is how to help system developers specify requirements and how to translate them into requirements for adaptation logic. These adaptation requirements are first order entities[26] that are present throughout the system life cycle. As with any approach that aims to provide a serious logic of adaptation, this one is based on the MAPE-K loop by opting for a particular level of decentralization [7]. This Framework is completed by its tool which is an integrated development environment[27], which is provided a little later after the approach. It is based on the Eclipse editor and technologies like the JSON format. It offers two tools: the FESAS development tool and the FESAS design tool. These tools represent the separation of responsibilities in the development of Adaptive System using FESAS in roles of developer and designer of system. The system developer uses the FESAS development tool to develop the functional logic code, which is then stored in the FESAS repository.

The following table gives us an overall summary of the approaches.

TABLE I:Table summarizing the descriptions of the approaches

Name	Type	Based on	Application domain
Adapt Case	Modeling language	MAPE-K/ UML Use case	General
Active Forms	Methodology	Component replacement	General
Model-based Simulation at Runtime for self-adaptive Systems	Methodology	MAPE-K	General
FESAS	Framework	MAPE-K/Model-Driven	Pervasive system
FESAS IDE	Tools	MAPE-K/Model-Driven	Pervasive system

All these methods presented seem to have a scope affecting each aspect of system adaptation, starting from the definition of needs to the management of adaptation. They consist in presenting simulations of the behavior of the models in the process of adaptation, which does not offer a real platform for the designer. The methodology that we propose consists in offering more means of communication specific to a domain (that of self-adaptation here) to formalize the expression of the adaptation needs of the system. Thus from these expressions the management of the adaptation of the system can be made independently of our methodology.

III. Scenario

In this section, we present a simple scenario to show how a model-based approach can ease the implementation of adaptation in comparison with a code-based one. Very often, we try to access a page on a website without success. The system must therefore find the most effective alternative answer. The solution to this kind of problem is to consider alternatives explicitly while building the system.

In our case, the aim is to implement a MAPE-K loop. Doing this following the traditional technique of software development leaves the developer with too much freedom therefore all the procedure is carried out on the lines of code. Whenever a change occurs, it will be necessary to return to the code to mirror this change. This approach is costly in terms of time and maintenance.

In contrast, model-driven engineering significantly reduces this task and improves the quality and maintainability of the solution provided.

Figure 5 shows the different phases of the traditional method for building information systems.

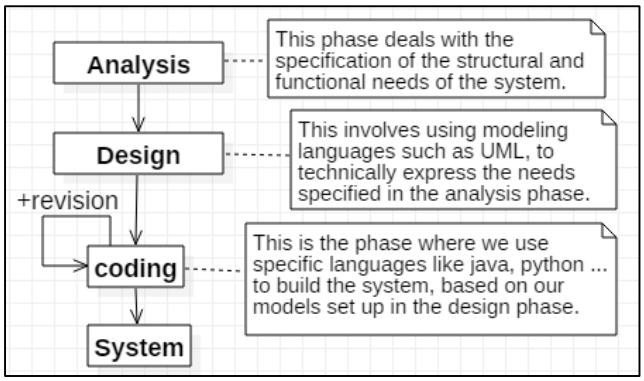

Figure 5:Traditional process

Figure 6 shows the phases followed in model-driven engineering to build information systems.

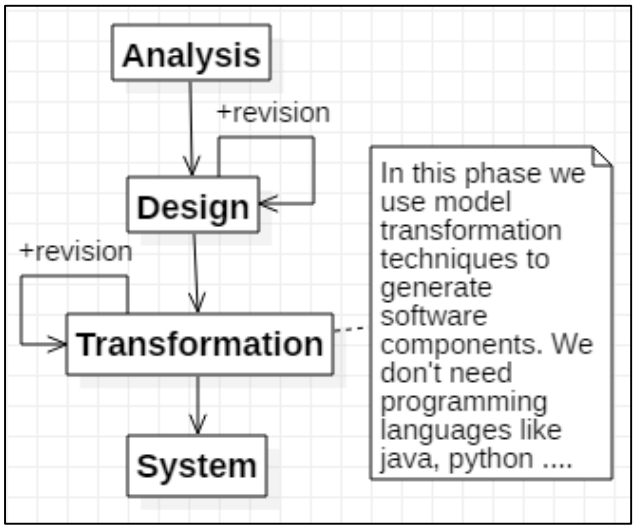

Figure 6:The method of model-driven engineering

In the scenario, we simulate the adaptation mechanism for a product management web site. If the user requests yields to the selection of a non-existing web page, an "ordinary" system will return a kind of 404 error page as shown in figure 7.

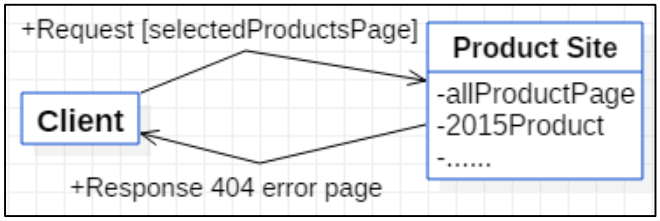

Figure 7:Traditional response

In our case we will try to solve the problem differently.

The system has to return a page similar to the one requested. Figure 8 shows the behavior aimed by our solution.

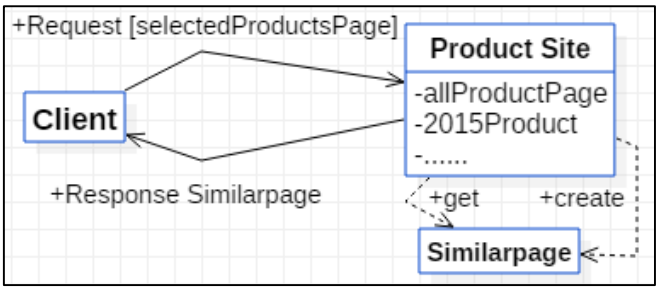

Figure 8:Response according to our vision

IV. ENVIRONMENT SETUP

In this section, we present the implementation of the different technical tools allowing the use of the approach. We thus present the way in which the UML profile, personalizing the components involved in the system adaptation process, is defined. The semantics of each component participating in the UML profile is explained here.

A. Profile definition

To configure our modeling environment, we suggest defining a UML profile. The profile is a mechanism which makes it possible to specialize UML by redefining its syntax. Each element is redefined through a stereotype which makes it possible to enrich its semantics by adding other properties and / or other behaviors.

Our profile allows us to have models to represent components specific to the field of adaptive information systems. Each of the components involved in adaptation management is represented by a stereotype through which we give it a new use.

The profile that we define is applied on the activity diagram to define the adaptation needs of the system to be designed. The needs are first defined using the activity diagram, then we apply the profile to give each component its properties.

The defined stereotype and their descriptions are presented on the following table.

TABLE II:The Stereotype descriptions

Type	Descriptions
Basic	It is used to define a simple element which does not require much details.
Listening	It is used to determine a gateway for listening and superficial information processing. it has a name and an action which indicates its real functioning.
Temporary	These are the models generated during the process when certain conditions are met. They have a limited lifespan.
Archive	It is used to archive all the adaptation responses already proposed.
System	It is the model which allows the representation of the Information System without detailing.
Selector	It is the model which takes care of the selection of the most appropriate component, among those generated, to ensure the adaptation.
Main	it is a model which performs, if necessary, the generation of new components intended to replace the defective ones.
Manager	It is the model used to process the information contained in the event generated following the change in the system environment. It allows you to decide whether there is an old reusable solution or whether to claim a new one.
Alert	It is used to prepare a request in the form of an alert which will be distributed to another model.
Repeat	It represents the repetition of an action according to specific conditions.
Generation	It is used to designate the action for generating a new model.
Generic link	It is used to represent a link that is not explicitly defined between two models.

These are classes that inherit from more general meta-classes to take advantage of their behavior and specialize in an application domain. A meta-class represents a class which instances are also classes. In other words, it represents a class that describes other classes. Figure 9 shows the global profile diagram where we can see all the stereotypes, their categories and their links with the meta-classes.

After having defined the stereotypes, we define a special theme in the profile to distinguish the view of each stereotype.

In the definition of this theme, we used the cascade style sheet (CSS) language. The CSS language has been integrated into the papyrus platform (papyrus is a version of eclipse extended by modeling tools) with a very precise model-oriented syntax. In its basic form, CSS allows to act on an element using its class or identifier, with the profile in papyrus it allows to act on an element through its stereotype; the properties are also customized.

Figure 10 shows a snippet of the theme's code, where it is clear to distinguish the various syntaxes and properties offered by papyrus.

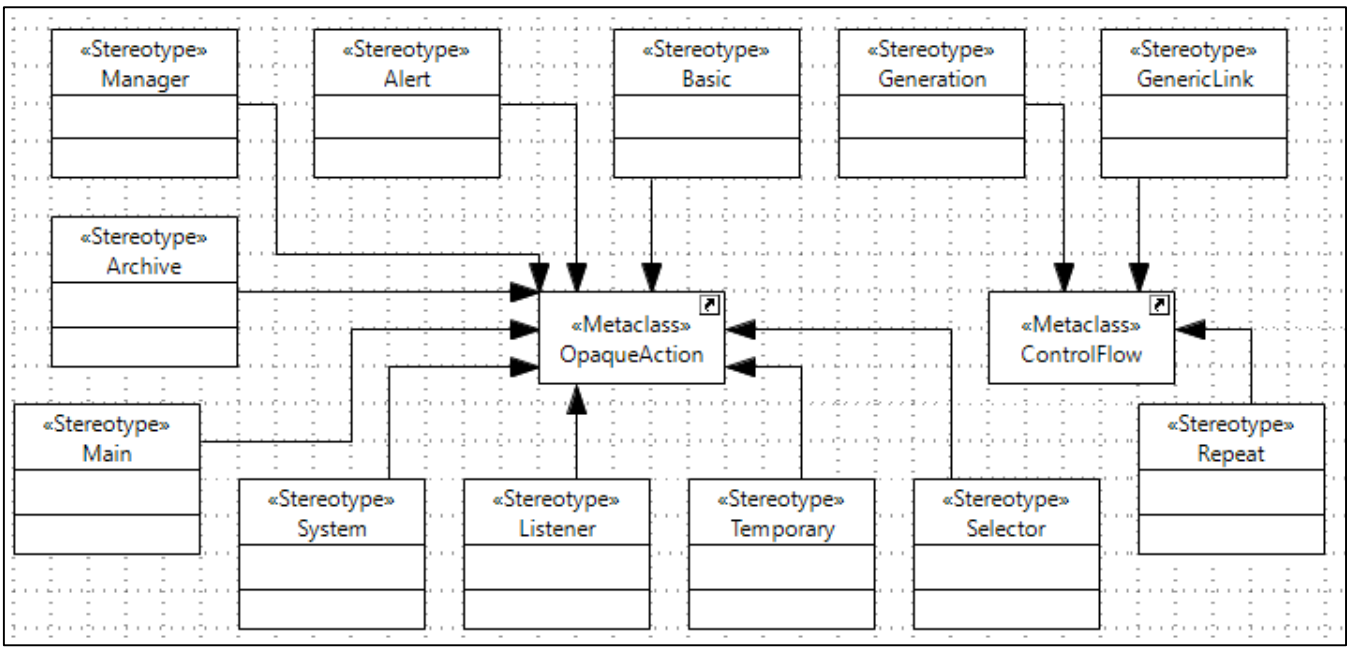

Figure 9:The profile diagram

```
[appliedStereotypes~="Temporary"]{
      bold: true;
      fontColor:white;
      fillColor: skyblue;
      elementIcon: true;
      borderStyle:dash;
      lineWidth:2;
       radiusHeight: 0;
      radiusWidth: 0;
}
[appliedStereotypes~="Manager"]{
      fillColor:#4D8DF0;
      bold: true;
      shadow: true;
      shadowWidth: 8;
      shadowColor:blue;
      fontColor:white;
      elementIcon: true;
      lineWidth:0;
      radiusHeight: 0;
      radiusWidth: 0;
}
```

Figure 10:Snippet of the theme's code

B. Structure of the approach

The good organization of the MVC concept makes it a good source of inspiration for a well-structured approach. Its technical maturity and the conclusive interactions between its different parts constitute a base for a good organization of all techniques that relies on it. Its structure, which is made up of three parts (Model View Controller), allows operations to be grouped into block according to their nature. So, we take advantage of this organization of the MVC to structure our approach. It is therefore divided into three parts (shown in figure 11) as follows: System, Controller, Response process. We will explain these different parts in detail later in this document. Each part of the approach allows the expression of adaptation aspects for the system section it represents.

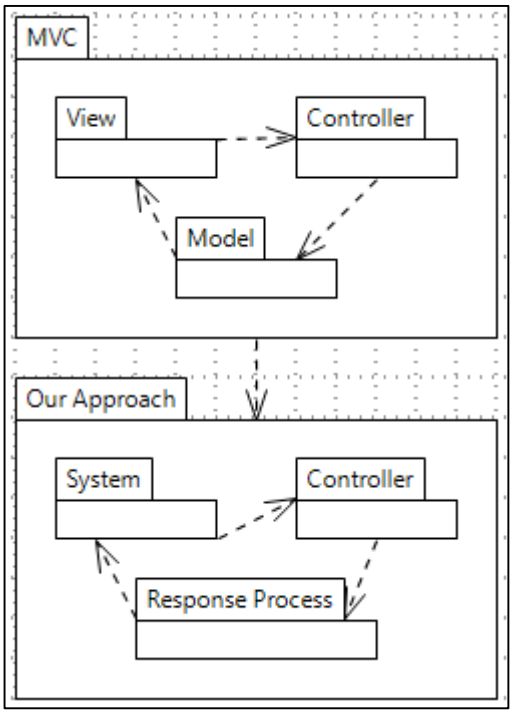

Figure 11:Correspondence between MVC and our approach

The part called system allows to express in detail the components of the system and its environment. The controller performs the analysis of changes occurred in the system environment and decides on the action to be taken. The response process describes how the response to a malfunction will be prepared. All these three parts will be presented in more detail in the following sections.

V. THE CONCRETE APPROACH

In this section we expose the different parts of our approach by explaining the parts responsible for the expression of each

phase of adaptation. These different parts are those inspired from the MVC structure as explained in the previous part (figure 11).

This approach is a method for modeling and specifying the adaptation needs of a given information system. It makes it possible to show clearly the different models, their role and their interactions with others. Each adaptation phase is managed in a well-defined part of the architecture of the approach. It is based on several types of models to provide the tools necessary for modeling the adaptation phenomenon. It allows a clear expression of all the phases of adaptation spread over the three design areas. The system part represents everything that is functional, the response process part is the zone where the adaptation decision is made and the controller is the zone where an analysis of state's change of the system made. The main purpose of this approach is to describe the phenomenon of adaptation for a system under design. It is based on the generation and replacement of system components. When there is an environmental hazard, new solution components are generated to replace the old ones which are affected by the change that has occurred.

A. System

The system part represents the global environment of the self-adapting system. It also presents all the links between the various elements of the environment. Each element of the environment has a well-defined and essential role in the approach. Here we will explain the meaning and the function of each of the model elements.

In this part we are able to represent by models the following elements:

System: A Basic type model to represent the adaptive system without any details.

Data: A Basic type model to represents the mass of data processed by the adaptive system.

Component: A Basic type model used to represent the various structural components that are part of the adaptive system architecture. It is the minimalist element (like an entity in UML)[1] of the adaptive system's internal architecture.

Event: A Temporal type model to express a change occurred in the system environment, so a decision needs to be made.

Operation of the system part:

The system environment representative area works in a very consistent and synchronized manner. A specific model constantly monitors all the components and the system data flows. When a malfunction affecting the system occurs in the environment, an event as Temporal model will be generated. All details relative to the malfunction are contained into the event. As soon as generated, the Event model is read by the devices present in the controller part.

B. Controller

This area triggers the adaptation response process following a malfunction in the system environment. The process begins following the generation of an event by the system part.

In this part, we are able to represent by models the following elements:

Listener: a model that constantly listens to the system environment to monitor its state. It reacts to the generation of an event in the system environment that expresses a malfunction in the adaptive system.

Archive: it represents a container of the old proposed adaptation responses.

Manager: this is a model intended to manage the decision in the controller following the analysis of an event carried out by Listener. It decides if an old solution is enough for the problem or a new one is necessary. If a new solution is necessary, the Manager triggers a new Request.

Operation of the controller part:

The controller is constantly listening to the system environment through the Listener model. And whenever there is no change in the system environment the listening action gets stuck through. When an event is generated, the Listener reads and analyzes it. Once the change's cause has been determined, the controller decides through the Manager model the action to be taken. Either the choice of an old satisfactory answer through the Old model repository or whether a new response must be requested. In the case that a new response is required, it is made through a new request send to the Response process part.

C. Response process

It is in this part that the response for adaptation is determined. In this part we are able to represent by models the following elements:

Main: A model responsible for the generation of the new components that will be used to replace the old defective ones in order to ensure the adaptation of the system.

Selector: A model that allows selecting between the generated components in order to designate the most appropriate to be used.

Component: A Temporal type model to be chosen to serve as a spare part to replace an old defective component.

Operation of the response process part:

It is a crucial part in this approach because this is where are determined the responses to the malfunctions occurred in the system environment. It ensures the continuous verification of a new response request. When a new response request is sensed, the Main model analyzes it and triggers the component generation process. Several components are generated related to the nature of the response request. These generated components must technically be similar to the corrupted components in the system but with additional properties. These components will then be read by the Selector model which will analyze them and make a selection to determine the most suitable to be use.

VI. VALIDATION

In this part we concretely apply the tools defined in this approach. We apply it to our first example of the product management application presented in the section scenario. We explicitly present how all three parts can be expressed to model the scenario.

We carry out this validation by a progressive presentation of three parts explained earlier in this document.

A. Modeling of the application web environment(System)

This part is used to represent the system environment, it represents the web application and its various structural

components (pages), the processed data (data), as well as the structure (state) which processes page requests made by users. When the page request is not satisfied, an event carrying a message will be created to inform the controller that a request has failed (a malfunction has therefore occurred).

Since our profile is based on UML Activity diagram, we will first make the needs expression using an activity diagram and then apply the profile. We will define the remaining parts (controller and Response process) using the same procedure. The following figure (figure 12) shows the initial Activity diagram for the system part.

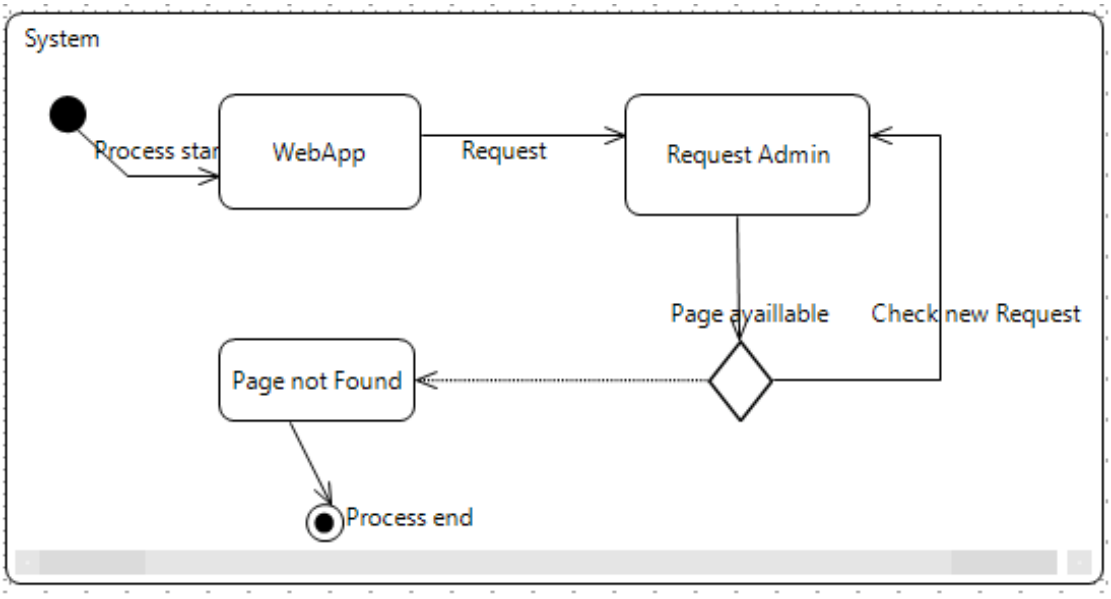

Figure 12:Description of the failure to return a page using Activity Diagramme

The following figure (figure 13) then shows the diagram after we applied the profile to it. We can clearly distinguish all the models and the links between them.

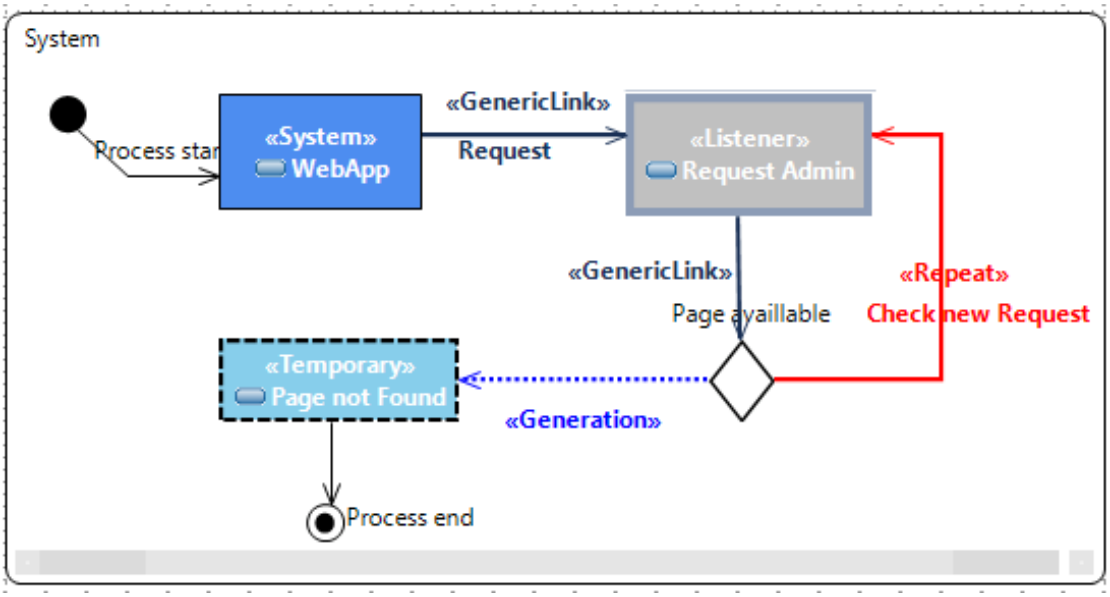

Figure 13:Description of the failure to return a page after applying the UML profile

B. Modeling of the anomaly detection(Controller)

The devices in this part will constantly try to read an event created by the system environment by performing tests. When it detects a new event, it will analyze it using the ad hoc model (Analyzer). Analyze will first explore the archive of existing solutions (previously used) through the component (Archive), if it finds an old page corresponding to the problem it will adopt it otherwise it will trigger the search for a new solution. To request a new response, the controller will produce an alert component to inform the Response process part that a new page needs to be created. All the expressions of the controller part are clearly presented on the following figure (figure 14).

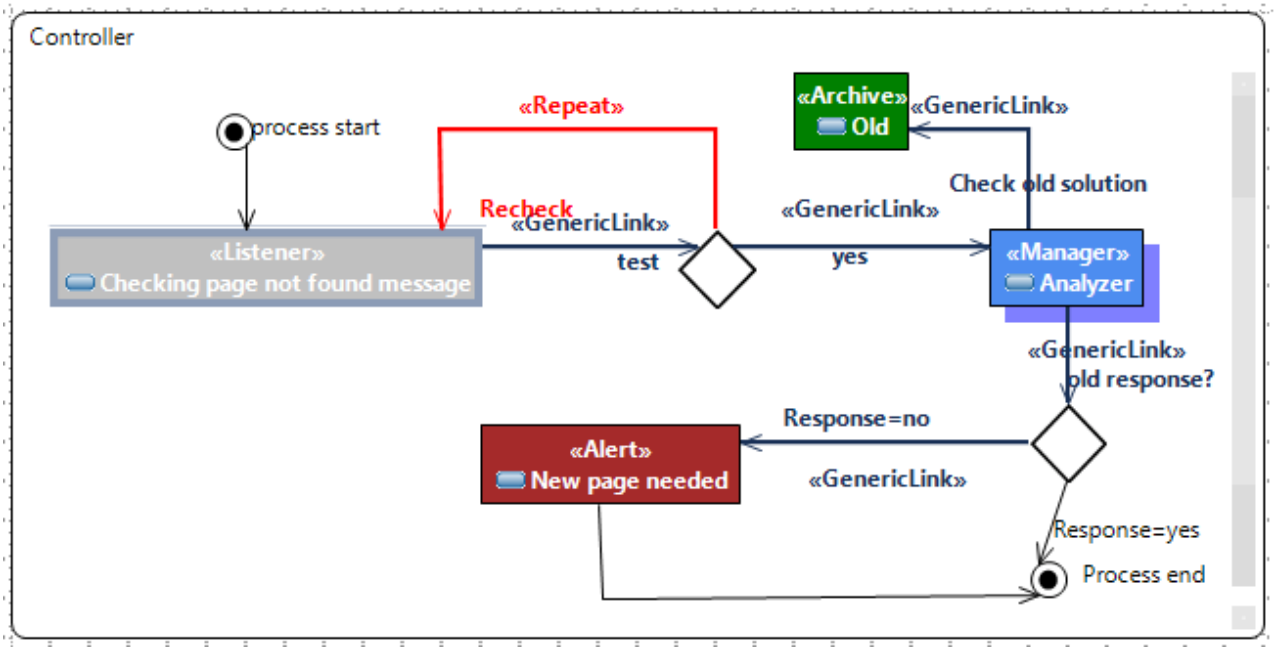

Figure 14:Solution request

C. Modeling of a proposed solution(Response process)

The response process always listens for a page need. When it reads a new page; a request sent from the controller part, the Main starts and generates multiple pages similar to the one whose rendering is impossible. All the generated pages will go through the selector which will make the decision on the best page to be sent to the user. This way the user will always get a satisfying answer when he requests a page. The following figure (figure 15) shows the process of generation et and the selection of the Response process part.

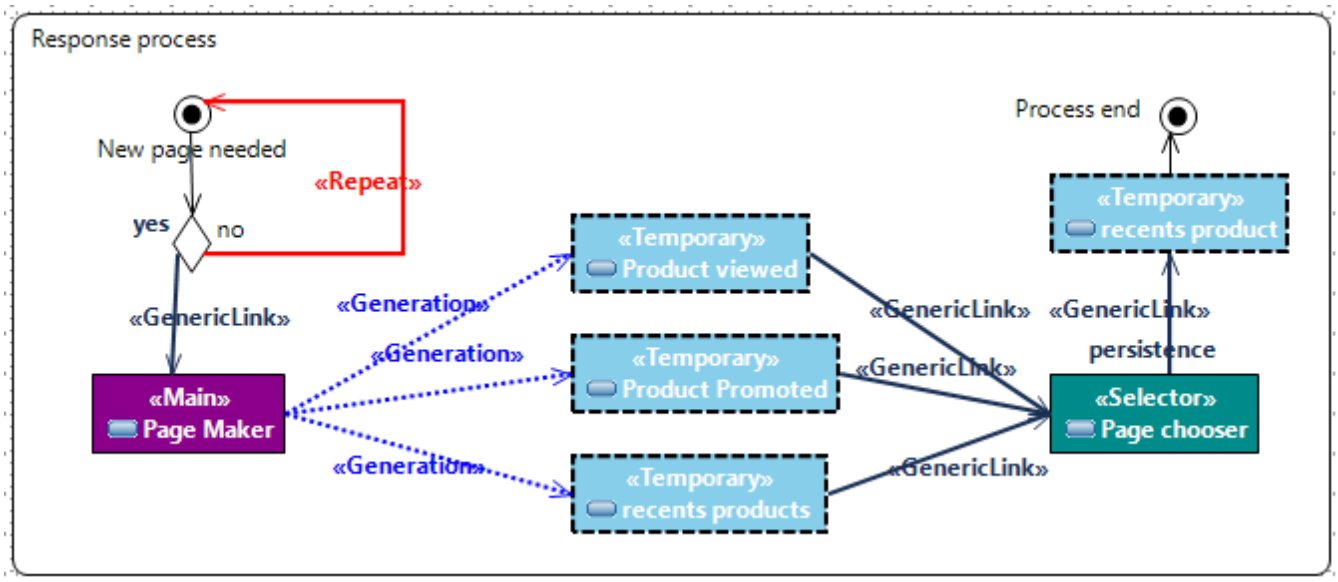

Figure 15:Process to provide response

VII. CONCLUSION AND FUTURE WORK

We headed in this work towards setting up a modeling environment. We explicitly presented the goal we wanted to achieve in the work. In order to specialize our environment and take into account all the components involved in managing the adaptation of a system, we have defined a UML profile. This profile is applicable to the UML language activity diagram. We have shown how to express the adaptation needs with the UML activity diagram to which we apply the profile to have the modeling. However, a real modeling environment must provide directly usable tools, generally in drag and drop, to represent the components of the domain. In addition, for its completeness, a modeling environment must provide a mechanism for automatic or semi-automatic code generation in the most widely used languages in the field. Therefore, it is clear that further work is needed to complete this modeling environment.

We thus aim, in the future, to extract directly usable models, embedded in the environment, in order to be able to represent each element contributing to adaptation management. We will therefore have an environment with elements graphically represented and ready to be used. In addition to this we

propose to integrate a code generation mechanism in the languages most used in the realization of adaptive information systems (JAVA, C # ...). We will be based on the techniques of transformation of the models; thus, we will have a complete environment for the modeling of adaptive systems.

REFERENCES

[1] R. S. Bashir, S. P. Lee, S. U. R. Khan, V. Chang, and S. Farid, "UML models consistency management: Guidelines for software quality manager," *Int. J. Inf. Manage.*, vol. 36, no. 6, pp. 883–899, 2016.

[2] C. Krupitzer, T. Temizer, T. Prantl, and C. Raibulet, "An overview of design patterns for self-adaptive systems in the context of the internet of things," *IEEE Access*, vol. 8, no. i, pp. 187384–187399, 2020.

[3] M. A. Bachir, J. Ismail, E. G. Said, and A. Souad, "Generic Model for Adaptive Systems Realization," *Int. J. Organ. Collect. Intell.*, vol. 12, no. 2, pp. 1–17, 2021.

[4] M. A. Bachir, J. Ismail, E. G. Said, and A. Souad, "Domain Specific Model Driven Approach for Adaptive Systems," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 10, no. 3, pp. 1705–1710, 2021.

[5] F. D. Macías-Escrivá, R. Haber, R. Del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Syst. Appl.*, vol. 40, no. 18, pp. 7267–7279, 2013.

[6] R. de Lemos *et al.*, "Software engineering for self-adaptive systems: research challenges in the provision of assurances," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9640 LNCS, no. December 2013, pp. 3–30, 2017.

[7] D. Weyns *et al.*, "On patterns for decentralized control in self-adaptive systems," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7475 LNCS, pp. 76–107, 2013.

[8] Y. Brun *et al.*, "Engineering self-adaptive systems through feedback loops," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5525 LNCS, pp. 48–70, 2009.

[9] P. Oreizy *et al.*, "An architecture-based approach to self-adaptive software," *IEEE Intell. Syst. Their Appl.*, vol. 14, no. 3, pp. 54–62, 1999.

[10] T. Lenton and P. Valdes, "Genie," *PIK Rep.*, no. 98, pp. 19–21, 2005.

[11] V. Cardellini, E. Casalicchio, and V. Grassi, "MOSES : a Framework for QoS Driven Runtime Adaptation of Service-oriented Systems," no. September, 2012.

[12] N. Gulfam Ahmad, Hafiz Tahir, Iqra Naeem Abbas, "Implementation of Novel Approaches in Bidirectional Model Transformation: a Systematic Literature Review," *Azerbaijan J. High Perform. Comput.*, vol. 4, no. 1, pp. 91–112, 2021.

[13] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, no. 3, pp. 621–645, 2006.

[14] M. A. Mohamed, G. Kardas, and M. Challenger, "A Systematic Literature Review on Model-driven Engineering for Cyber-Physical Systems," 2021.

[15] T. Mens, P. Van Gorp, D. Varró, and G. Karsai, "Applying a model transformation taxonomy to graph transformation technology," *Electron. Notes Theor. Comput. Sci.*, vol. 152, no. 1–2, pp. 143–159, 2006.

[16] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, vol. 1, no. 1. 2012.

[17] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, 2008.

[18] Object Management Group (OMG), "Meta object facility 2.0 Query/View/Transformation," *Final Adopt. Specif.*, no. January, pp. 1–230, 2011.

[19] Eclipse, "www.eclipse.org/acceleo/documentation/," 2021. [Online]. Available: https://www.eclipse.org/acceleo/documentation/. [Accessed: 08-Feb-2021].

[20] F. A. Somogyi and M. Asztalos, "Systematic review of matching techniques used in model-driven methodologies," *Softw. Syst. Model.*, vol. 19, no. 3, pp. 693–720, 2020.

[21] M. Luckey, B. Nagel, C. Gerth, G. Engels, and W. Straße, "Adapt Cases : Extending Use Cases for Adaptive Systems," pp. 30–39, 2011.

[22] D. Weyns and M. U. Iftikhar, "ActivFORMS : A Model-Based Approach to Engineer Self-Adaptive Systems ActivFORMS : A Model-Based Approach to Engineer," no. September, 2019.

[23] D. Weyns and M. U. Iftikhar, "Model-based simulation at runtime for self-adaptive systems," *Proc. - 2016 IEEE Int. Conf. Auton. Comput. ICAC 2016*, pp. 364–373, 2016.

[24] C. Krupitzer, S. Vansyckel, and C. Becker, "FESAS: Towards a framework for engineering self-Adaptive systems," *Int. Conf. Self-Adaptive Self-Organizing Syst. SASO*, pp. 263–264, 2013.

[25] B. H. C. Cheng, H. Giese, P. Inverardi, and J. Magee, "Software Engineering for Self-Adaptive Systems," vol. 5525, no. January, 2009.

[26] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: Requirements as runtime entities," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, no. June 2014, pp. 199–202, 2010.

[27] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schurr, "FESAS IDE: An integrated development environment for autonomic computing," *Proc. - 2016 IEEE Int. Conf. Auton. Comput. ICAC 2016*, no. March 2020, pp. 15–24, 2016.