VOL. 18, NO. 4, FEBRUARY 2023 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

USING A GESTURE RECOGNITION MODELING APPROACH TO IMPROVE GRAPHICAL USER INTERFACE

Issam El Magrouni[1], Abdelaziz Ettaoufik [2], Aouad Siham[3], Aberrahim Maizate[4] and Bennani Lotfi[5]
[1]RITM- ESTC/CED -ENSEM, Hassan II University, Casablanca, Morocco
[2]LTIM, FS Ben M'SIK, Hassan II University, Casablanca, Morocco
[3]SSL ENSIAS, Mohamed V University, Rabat, Morocco
[4]RITM- ESTC/CED -ENSEM, Hassan II University, Casablanca, Morocco
[5]Innovatel Engineering Sarl, Hassan II University, Casablanca, Morocco
E-Mail: magrouni@gmail.com

ABSTRACT

The gesture recognition technology based on visual detection employs image processing for detection, segmentation, tracking, and recognition of hand gestures. Static and dynamic gesture recognition are the two types of gesture recognition. In this paper, we present our hand gesture interface consisting of three main layers: Hand Gesture Interface, Mapping Gesture, Action, and Graphical User Interface with our approach named IGMA (Improving Graphical User Interface Based on Gesture Recognition Modeling Approach). IGMA organized in two stages: the modeling stage and the code generation stage. The model is Domain Specific Model; it allows action and gesture modeling. At the code generation stage, a specific code generator transforms all models to final source code using a framework specifically designed for a given target platform. Finally, to simplify the implementation of this method, we propose defining a process for improving user interface control based on gesture recognition. We validate our approach by performing experiments on a gastronomy application. The result is an adaptation framework that can guide software engineers in developing Graphical User Interface Based on Gesture Recognition.

Keywords: hand gesture recognition, deep learning, neural network, user interface, domain specific modeling (DSM).

1. INTRODUCTION

Human-computer interaction technology is gradually changing from computer-centric to human-centric. As an important human-computer interaction method, gesture control provides people with a natural and intuitive way of communication. It has been initially used in the entertainment industry and it has strong practicability in robot control [1]. Rich gestures are difficult to recognize due to the small number and consistency of markers. Along with the development of visual technology, unmarked visual gesture recognition has become the current Mainstream.

The main process of visual gesture recognition includes: (1) Image acquisition: using a visual camera collect gesture images; (2) Hand detection and segmentation: detect the position of the hand in the palm gesture image and segment the hand area; (3) Gesture recognition: extract image features hand area, and recognize the gesture type based on these characteristics.

Static gesture recognition and dynamic gesture recognition are two types of vision-based gesture recognition technology. The hand is in a static state for static gesture recognition, so the pose, shape, location, and other data details of the hand do not shift. It has the benefit of having a high recognition rate. Static gesture recognition, on the other hand, has its own drawbacks. Static movements, for example, can only convey a limited amount of knowledge and do not represent the characteristics of real human hand movement. Static gestures are a special condition of dynamic gestures, since they are composed of static gestures frame by frame.

Dynamic gesture recognition has the advantage of being able to convey more detail with continuously changing gestures, and it is commonly used in the field of human-computer interaction.

Gesture recognition can be classified into two types based on the theory and creation process: gesture recognition using conventional methods and gesture recognition using deep learning. This paper examines and analyzes recent research on visual gesture recognition, as well as the most popular gesture recognition methods. The key technologies used in each gesture recognition process are then compared and analyzed.

Gesture identification and segmentation, gesture monitoring, gesture feature extraction, and gesture classification are examples of these processes

Finally, aiming to detail and explain our proposal, this paper is structured as follows: Section 2 brings the related works; Section 3 and 4 describes our proposal; Section 5 presents our experiments; Section 6 discusses the results and finally, in Section 7, our conclusions and future works.

2. RELATED WORKS

In recent years, the field of dynamic gesture recognition has gotten a lot of attention. A number of recent studies have attempted to create a more natural interface for interacting with computers and other devices. Recognizing a complex gesture is difficult, however, since a gesture can be interpreted in a variety of ways within the same context. Furthermore, the manner in which gestures are performed is influenced not only by the sequence of body movements, but also by the cultural background of those who perform the gestures. As a result, there is still a lot of work to be done in order to create an interface that

allows humans and machines to communicate effectively using gestures.

However, it appears that the essence of gestures, their applicability, fitness for use, and effectiveness were not always the main focus of the articles reporting on the prototypes created. Gestures were either the only mode of interaction or one of many, but neither mode seemed to incorporate the roles of gestures in communication into the interfaces.

The methods of visual gesture recognition that will be reviewed will be classified into the following families: static, dynamic, depending on supports (Kinect, Leap...), works focusing on the application of gesture recognition to robotics, and works focusing on gesture recognition at the browser level

Ameur *et al* [14] suggested a dynamic hand gesture recognition approach over a leap motion system, which uses touchless hand movements. To begin, they use recurrent neural networks with Long Short-Term Memory (LSTM) to evaluate the sequential time series data collected from leap motion for recognition purposes. Basic unidirectional and bidirectional LSTM are used separately. The final prediction network, called Hybrid Bidirectional Unidirectional LSTM, is generated by combining the aforementioned models with additional components (HBU-LSTM)

Santos, Clebeson Canuto dos, *et al* [15] proposed the dynamic gesture recognition technique. The plan is made up of two key steps: Pre-processing: using a modified version of the aforementioned star representation each input video is represented as an RGB image. • Classification: An ensemble of CNNs is used to train a dynamic gesture classifier. The image from the pre-processing phase is fed into two CNNs that have already been trained. After passing through a soft-attention mechanism, the effects of these two CNNs are weighted and sent to a completely connected sheet. Finally, a softmax classifier determines which class the gesture belongs.

Almasre *et al* [17] proposed a dynamic prototype model (DPM) that recognizes some ARSL gestured dynamic words using Kinect as a sensor. The prototype model (DPM) DPM is used with different parameter settings based on three algorithms (KNN, SVM and RF). Specifiying that, the SVM models had the highest recognition accuracy rates for the complex words gestured.

In the hand gesture interface proposed, Egemen *et al* [25] enhanced the finite-state-machine (FSM) technology by allowing users to perform unique GUI operations by leveraging gesture-specific parameters such as distance between hands, distance from the camera, and time of occurrences.

The RealSense SDK, which is used in our hand gesture interface, detects hand movements and extracts these attributes. Users can use these gesture-specific properties to trigger static gestures and then perform them as dynamic gestures. The authors also improved the efficiency, convenience, and user-friendliness of the hand gesture GUI by adding more capabilities.

In an Operating room environment without personal basis training and skeleton features from the Leap MotionTM sensor, A-reum Lee *et al* [27] used a gesture-based interface to identify five hand gestures and compared it to various deep learning algorithms such as DCNNs and CapsNet. CapsNet performed best in recognizing complex hand movements, which could be used as a non-contact interface in the OR to monitor clinical software.

Rihem Mahmoud *et al* [28] suggested a new recognition method to solve the issue of large-scale continuous gesture recognition using depth and gray-scale input images. The proposed recognition scheme is divided into three stages. To begin, continuous gesture sequences are segmented into isolated gestures using mean velocity information derived from deep optical flow estimation. Deep signature features are a collection of relevant descriptors extracted for each isolated segment in order to describe different intensities and spatial information describing the movement's location, velocity, and orientation.

M. Meghana *et al* [29] developed a robotic vehicle model that is powered by speech signals and hand movements. The main component of this model will be an Android smartphone that will communicate with the robot through Bluetooth. This technique can be used to assist people with disabilities, as well as in industrial applications such as working robots that are controlled by voice and hand gestures.

The circuit is divided into two parts: hand motion recognition and voice recognition. Transmission and receiver modules make up the portion of the robot that deals with hand gestures. A software framework and a Bluetooth module are used for voice recognition. The voice and gesture recognition software is written using the Arduino IDE, which connects the microcontroller hardware to dump programs. This robot is around 5 different hand gesture inputs through a sensor MPU6050, namely the stop state, forward, backward, right, and left movements. When a user gives the voice commands ‘‘FRONT”, ‘‘BACK”, ‘‘LEFT”, ‘‘RIGHT”, and ‘‘STOP”, it works similarly.

Chen *et al.* exploited the Intel RealSense sensor in their work [30]. They propounded a network for dynamic hand gesture recognition based on skeleton data. As well, they extracted features from finger motions and global features with the skeleton sequence taken from the DHG-14/28 dataset and the SHREC’17 dataset. Afterwards, they used the LSTM recurrent network to predict the class of input gestures.

Sarkar *et al.* [32] employed depth-information delivered from a Time-of-Flight (ToF) sensor to train a D-LSTM network for hand gesture recognition purposes. High accuracy was achieved with real-time processing.

Google's TensorFlow.js [40] is an in-browser machine learning library that allows users to define, train, and run models entirely in the browser using JavaScript. It replaces deeplearn.js, which is now known as TensorFlow.js Core. TensorFlow.js is a high-level API for defining models that is controlled by WebGL. Both Keras

layers are supported by TensorFlow.js (including Dense, CNN, LSTM, and so on). Therefore, it is easy to import models pre-trained by the native TensorFlow and Keras into the browser and run with Tensorflow.js.

This table summarizes the characteristics of the different methods using the following criteria sensor and methods.

Table-1. The characteristics of the different methods using for gesture recognition.

Sensor	Related Work	Methods	Year
Kinect	Almasre, Miada A, et Hana Al-Nuaim	SVM, RF, KNN	2020
Leap Motion	Ameur, Safa et al	BU LSTM	2020
	Santos, Clebeson Canuto dos et al	CNNs	2020
	Mahmoud, Rihem et al	SpyNet	2020
IntelRealSense	Chen et al	LSTM	2019
ToF-sensor	Sarkar et al	D-LSTM	2017

3. METHODOLOGY

A. System Overview

Hand Gesture Interface, Mapping Gesture Action, and Graphical User Interface (Figure-1) are the three basic layers of our system. The proposed Hand Gesture Interface's is shown in Figure-2.

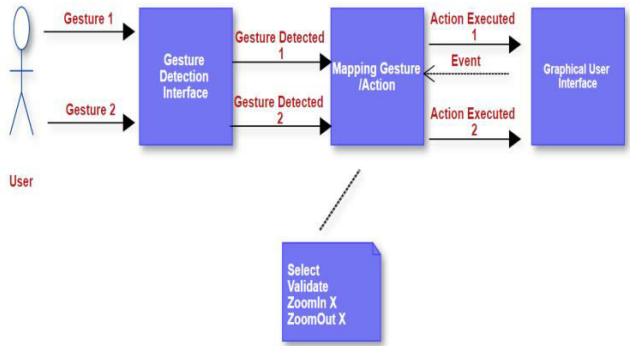

Figure-1. Overview of system showing main components.

The suggested Hand Gesture Interface is the first layer, which is intended for acquiring control over current GUIs such as Web sites. To access full-hand skeleton/joints and recognize user gestures, we used the sole dispositive camera for recording and processing the stream of pictures acquired by depth and color cameras. In addition to our interface, we employed the tensorflow.js framework, which serves as a vital link and layer between our interface and the ultimate layer, the built-in GUI.

We use the layers mapping gestures to map the motions recognized by the Hand Gesture Interface (Layer 1) to keyboard or mouse input and simulate them (Layer 2).

This figure presents a flowchart diagram of proposed hand gesture.

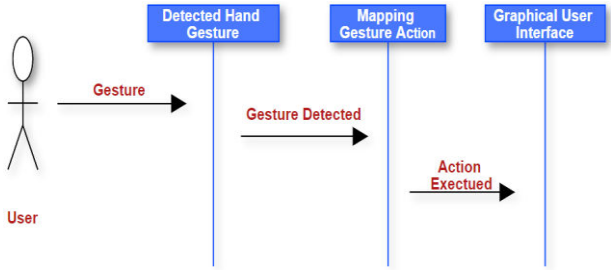

Figure-2. Flowchart diagram of proposed hand gesture interface.

B. IGMA Approach Synoptics

Our approach adopts the concepts and principles of Model Driven Software Development (MDSD). It aims to facilitate modeling and development of our hand gesture interface layer based on Gesture recognition, using DSM approach. This approach is entitled IGMA (Improving GUI based on recognition gesture Modeling Approach). The latter consists in the elaboration of specific models and their transformation to the final source code using a model-to-text transformation language. In order to separate the concerns and to control the complexity of this type of system.

Our approach is divided into two stages: The modelling stage and the code generation stage.
The modeling stage is based on models:

a) Command model: is a representation of the domain specific actions. We have defined a generic abstract syntax which can be extended by a specific abstract syntax to produce a domain specific abstract syntax; The domain specific code generator uses the domain specific framework to transform models into full source code. The Figure-3 illustrates our approach IGMA.

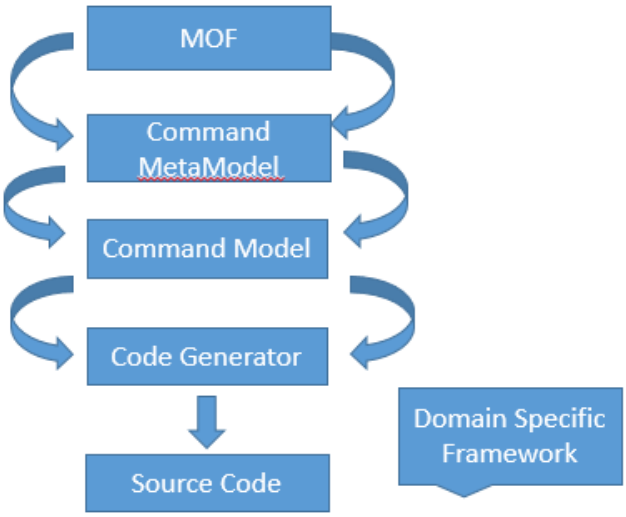

Figure-3. IGMA approach Synoptic.

In this section, we have been focusing on the modeling side of our approach : command modeling.

B. 2 Command Modeling

An action is a command to perform actions in software, such as validations, selection. We have proposed a meta-model of Command to organise the elements of a GUI based on gesture recognition. This proposal is inspired by the design pattern command. The command pattern is a behavioral design pattern that is included in the GoF‘s formal list of design patterns. The pattern intends to encapsulate in an object all the data required for performing a given action (command), including what method to call, the method's arguments, and the object to which the method belongs. The Command element represents the core of our generic command meta-model. The attributes "name", "version" and "description" represent the identity of the action.

- Command: specifies an interface for executing an operation.
- Concrete Command: defines a binding between a Receiver object and an action and implements Execute by invoking the corresponding operation(s) on Receiver.
- State: defines a behavior associated with a particular state of the Context
- Geste: the input symbol we consider Geste and State as the invoker who asks the command to carry out the request
- Receiver (GUI): knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.

The Figure-4 illustrates action meta-model.

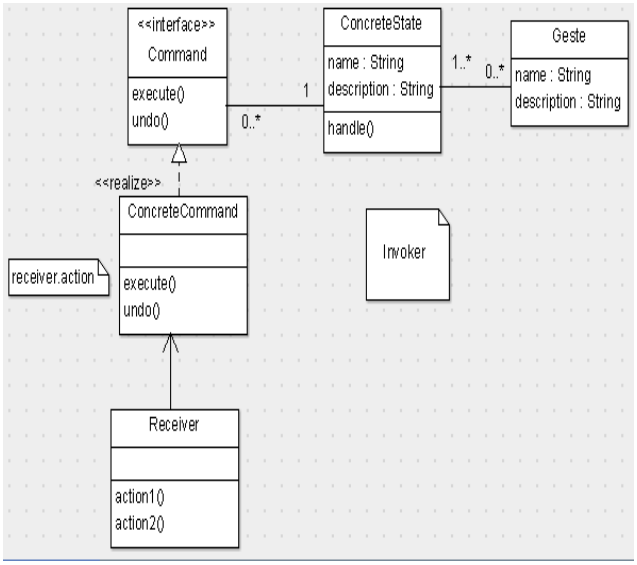

Figure-4. The generic action meta-model.

The main concerns addressed by our meta-model are as follows:

Generic meta-model: to be specialised by elements of a specific domain using the inheritance mechanism;

Extensible: the model must allow the addition of a new type of action in a simple and intuitive way; close to a standard: Integration of all the common concepts of the meta-models studied, in order to facilitate its adoption by existing systems;

The meta-model represents only a description of the abstract syntax of a language. A correspondence between the concepts of a meta-model and a description using a common language is necessary.

B. 3 Formalizing Action Gesture

A state machine includes a set of states and a set of input symbols, where the state-transitions in the machine is controlled by the occurrence of a given input symbol at a given state. In the context of our work, a Mealy FSM can be formally defined as a triple (S, A, f) where:

- S: is a set of states.
- G: represents the set of possible input Symbol or geste
- f: is the combined state-transition and output function: f: S×A→S Thus, for every combination of current state and input, f gives the next state. Let S = {s1, s2,… sn} be a set of states and G = {g1, g2, …, gm} be a set of m input symbols. Consider a deterministic state machine, where occurrence of an input symbol gk at a given state si causes a transition in the machine to a fixed new state sj.

If there exists a valid transition of state si to sj for any input symbol ak. It is indeed important to note that in a deterministic state machine f(si,gk) = (sj), there cannot exist any other symbol al, such that f(si,gl) = (sj).

$\exists$ si, sj$\in$S si$\neq$sj $\exists$ gk$\in$A $\forall$ gl$\in$A gk$\neq$gl
f(si,gk) = (sj) => f(si,gl) $\neq$ (sj) (1)

C. Code Generation Stage

In the code generation stage the language developer must produce the domain specific action meta-model. The domain specific code generator uses the domain specific framework to transform models into full source code. The domain framework essentially removes duplication from the generated code, hides the target environment and facilitates integration with existing code.

The Figure-5 illustrates the code generation.

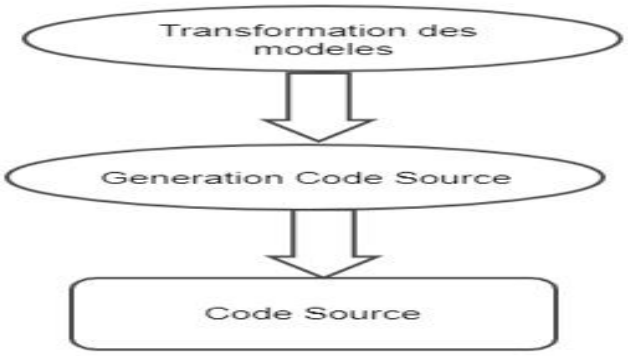

Figure-5. Code generation.

4. PROCESS FOR IMPROVING USER INTERFACE CONTROL BASED ON GESTURE RECOGNITION

To simplify the implementation of this system, a process for increasing user interface control based on gesture recognition must be defined. This process should define the phases, activities and artifacts that facilitate the identification, specification and implementation of mechanisms and which brings an answer to the constraints such as:

- The treatments must be real time.
- The methods must be robust to the acquisition conditions.
- The constraints imposed on users must be minimal.
- Make Web site Touchless without impacting the existing.

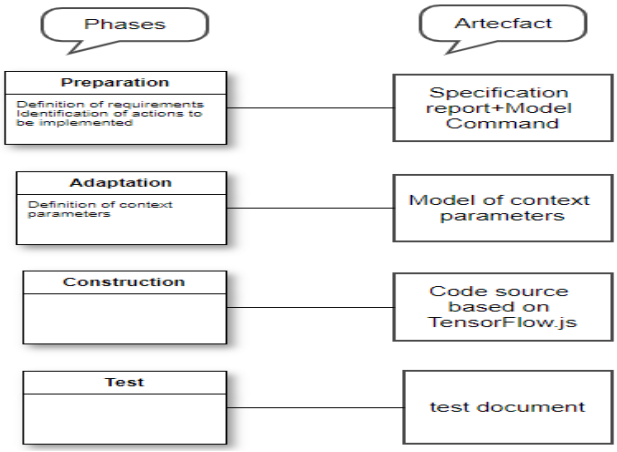

Figure-6. Process for interface based on gesture recognition.

This process will consist of the following phases:

- Preparation

This phase captures requirements, in order to produce a model focused on the end-users' needs.

The results of the analysis are not dependent on any particular technology. The figure 6 present an example of the uses cases of the application based on recognition of gesture

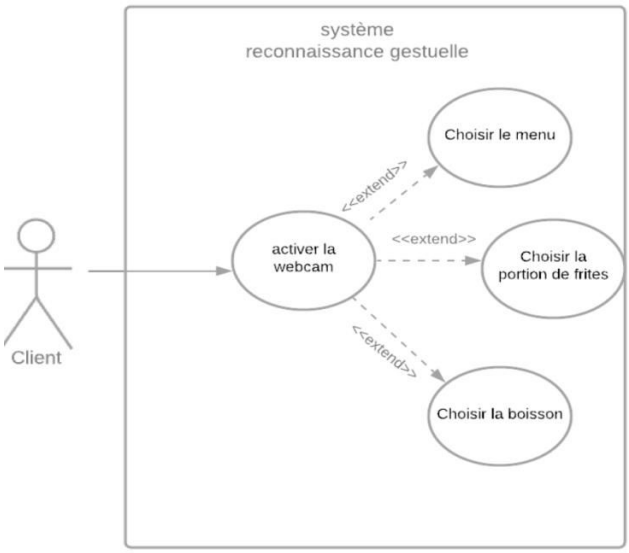

Figure-7. Uses cases of the application based on recognition of gesture.

This branch is articulated on the two steps: Definition of requirements and identification of actions to be implemented.

- Adaptation:

Following the appearance of new events: the user interfaces can at a time ti can evolve to another user interface Cj at time tj. Therefore, this mechanism must change its behavior according to the context. In order to

solve the problem of adapting user interfaces based on gesture recognition to changes in context, namely information on the characteristics, preferences and knowledge of the user:

We propose some adaptation steps and among the steps: definition of context parameters what will be the added value of a user interface improvement system based on gesture recognition if it does not take into account the possible contexts? With this in mind, we will identify all the variables that can modify the behaviour of the system. However, these variables must be classified in 3 categories:

a) Environment: the parameters of this category are determined mainly by the business analysts, since they are the most capable of defining information about the infrastructure and the spatio-temporal environment that surrounds it.
b) User: this category of context specifies information about the example user (profile, language, preferences, location, etc.).
c) Device: contains the parameters that describe the device, it is composed of two categories software (e.g., operating system, navigator type, supported type of data, etc.) and hardware (e.g., Kinect, leap motion, camera).

- Construction:

This phase concerns the implementation of this mechanism, for this reason, we may propose in the future a framework based on TensorFlow.js

- Test:

Finally a phase of testing is fundamental to be able to avoid a risk of regression and to check the 2 constraints which concern the precision and real time.

5. CASE STUDY MORGIA

Our approach can be illustrated by MoRGIA project (Model recognition gesture Based artificial intelligence) aims to design and implement a portable recognition system project, to personalize the Meal Menu without direct hand interaction with the screen and simply using hand gestures.

The objective of this project is to propose an improvement of the techniques and methods used in the field of gesture recognition to help people with loss of motor skills or to avoid contact with electronic equipment during pandemics like Corona virus. Artificial intelligence will be the main technology adopted to correctly decode human gestures and understand sign language.

Our hand gesture recognition application captures images from a webcam in real time. Using a webcam image is captured, processed for hand detection, features, gestures are identified.

This Figure-7 present 3 types of gestures that can be recognized: 1 finger, victory, three fingers to select and thumb up to confirm. These motivating scenario provides a good illustration of context based on recognition gesture.

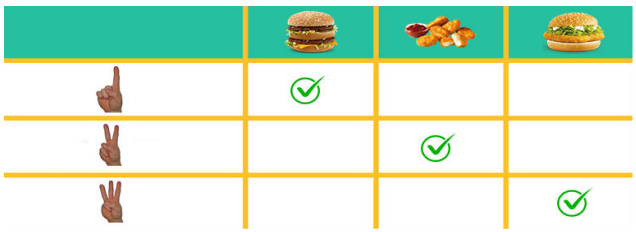

Figure-8. Application fast food based on recognition gesture.

6. CODE GENERATION STAGE

A. Model Command

Based on the command Meta model, we make the command model that concerns our Morgia application. To simplify the example, we will focus on two actions select Big Menu and Validate Big Menu using the two gestures one Finger and thumb up.

Figure-8 illustrates the examples of detailed model Command

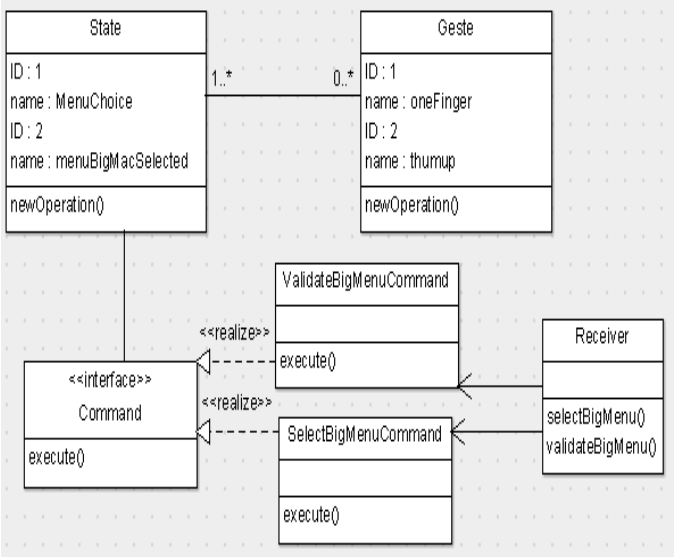

Figure-9. Model command.

B. Models to Code Transformation

We determine the transformation that allows the generation of the model Command implementation targeting the TypeScript implementation platform. In order to automatically generate the Model command implementation targeting TypeScript as an implementation platform, we must first define the TypeScript metamodel. Secondly, we need to define the mappings between the elements of the source metamodel, namely the Command metamodel, and the elements of the target metamodel, namely the TypeScript metamodel. Thirdly, we have defined transformation rules allowing to implement the correspondences between the elements of the two metamodels. Finally, we have defined ATL helpers for the generation of the TypeScript code constituting the implementation of the Model Command. The following

sections aim at describing the transformation steps defined above.

C. Type Script Meta Model

The Model Driven Architecture (MDA) transformation process requires a source and a target metamodel to be able to perform the model transformation. In our approach, the source metamodel chosen is that of the Model Command. The Figure-10 below presents a TypeScript Metamodel.

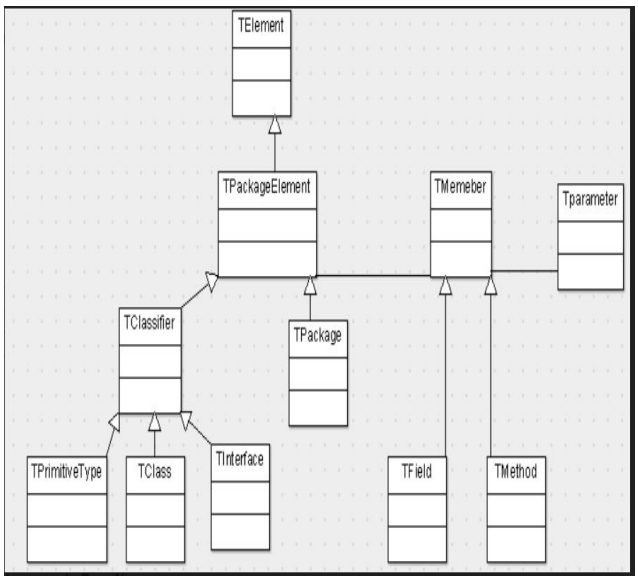

Figure-10. Type Script meta model.

The main elements of the Type Script meta model are:

- **TElement-** This element of the metamodel is a generalization of other elements such as TPackageElement, TMember and TParameter.
- **JPackageElement-** This element of the metamodel is a generalization of TPackage and TClassifier.
- **TPackage-** This element of the metamodel is a container for classes (tClass) and typescript interfaces (Interface). It is used to define packages that can contain TElement elements.
- **TClassifier-** This element of the metamodel is the basis for the TPrimitiveType, TClass and Interface elements.
- **TPrimitiveType-** This is an element of the metamodel that defines the primitive types that are used to build models. It can be Byte, Short, Boolean, String, integer, etc
- **TClass-** This is an element of the metamodel that is a specialization of TClassifier that has TField and TMethod as members. Each tClass implements one or more interfaces.
- **TInterface -** This is an element of the metamodel that is a specialization of TClassifier that has the prototypes of a set of java methods (TMethod).
- **TMember -** This is a metamodel element that defines the members of each class (tClass) including its fields and methods. It extends the TElement element and is associated with the jClass element to specify the class to which tMember belongs and to specify its type.
- **TMethod -** This is an element that allows the definition of methods. It extends the ClassMember element. It is associated with TParameter for the definition of the parameters of each method.
- **TParameter -** this element allows the definition of parameters for a given method. This element extends the TElement element. TParameter is associated with TClass and TMethod to specify the type of a parameter of a given method and to define the method to which the parameter belongs.
- **TField:** This metamodel element extends ClassMember. It allows the definition of the fields of a given class.

D. The Transformation Rules

The specification of the correspondences allows the determination of the equivalences between the elements of the source and target metamodel. The second step in the transformation process is the definition of transformations based on the transformation rules using the ATL language to map the Command metamodel to the TypeScript metamodel. As illustrated in Table-2, we have defined several rules for the transformation of models. This mapping is achieved through a set of transformation rules that specify the elements of the source metamodel that are equivalent to the elements of the target metamodel. In this table, the first column contains the elements of the source metamodel. The second column shows their TypScript metamodel equivalents. The third column contains the rules. These allow the implementation of transformations from the target metamodel elements to the target metamodel. The Figure-2 below presents a mappings between IGMA approach models and TypeScript code elements.

Table-2. mappings between IGMA approach models and TypeScript code elements.

Meta modele Command	Meta models Type Script	The transformation rules
Command Concrete	T Class	ComCon2Class
Command Interface	Tinterface	Com Inter 2 Interface
execute	T method	Execute 2 Method
Geste	T Class	Geste 2 Class
State	T Class	State 2 Class

The ComCon2Class transformation rule allows the creation of a TClass instance from the Command Concrete element of the Command metamodel. The characteristics of each TClass are assigned with the characteristics of the command. The interfaces of each created class are a collection of the command interfaces. Figure 10 shows the ATL code for this rule.

```
rule ComCon2Class{
        from
                s: COMMAND!CommandConcrete
        to
                t: TYPESCRIPT!JClass
}
```

Figure-11. Transformation rule command to class.

The Execute2Method rule allows the creation of method instances from the command operations. The characteristics of methods are assigned with the characteristics of the operation. Thus, name is assigned with the name attributes. Figure-11 illustrates this rule

```
rule Execute2Method  {
        from
                s: COMMAND!Execute
        to
                t: TYPESCRIPT!JMethod(
                        name <- s.name)
}
```

Figure-12. Transformation rule Operation to Method.

The commandinterface2interface rule allows the creation of interface instances by assigning the characteristics of command Interface. The Figure-12 shows the ComInter2Interface rule.

```
rule ComInter2Interface {
        from
                s: COMMAND! CommandInterface
        to
                t: TYPESCRIPT!JInterface(
                        name <-s.name)
}
```

Figure-13. Transformation rule Interface to Interface.

E. Transformation of Models to Source Code

In this perspective, we have defined additional transformations thanks to a set of helpers (functions) defined within each element of the TYPESCRIPT meta model. Figure-13 defines the code of the helpers allowing the realisation of the model to code transformation.

```
query J2String_query =
TYPESCRIPT!JClass.allInstances()->
                collect(x |
x.toString().writeTo('C:/test/' + '/' +
x.name + '.ts'));

uses J2String;
```

Figure-14. TypeScript code generation request.

Figure-14 contains the query used to trigger the model-code transformation. The allInstances() operation returns a collection consisting of all instances of the TYPESCRIPT!JClass element present in the TypeScript model. Then, an operation on the collections called collect() allows us to retrieve each element of this collection (represented by x) and call the toString() operation on each of them. This operation is a helper in ATL defined in the J2String library. After the execution of this helper, the string is written to a "*.ts" file through the writeTo() operation. The helper toString() defined in the context of TYPESCRIPT!JClass and TYPESCRIPT!Interface also chains other calls to other helpers defined in the context of State, Geste, and so on

```
helper context TYPESCRIPT!TInterface def :
getmethods() :
       Set(TYPESCRIPT!TMethod)=
       self.tmethod->asSet() ;
helper context TYPESCRIPT!TClass def:
toString() : String =
       'package ' + self.packagej.name +
       ';\n\n'+ '\npublic
       class ' + self.name +
       ' implements '+ self.interfaces -
>iterate ( i ; acc: String ='' | acc + i.name
+ ',') +
       '{\n'+ self.interfaces->collect ( e|
e.typescriptmember) -> iterate ( i; acc:
Sequence (TYPESCRIPT!JMethod)= Sequence{}|
       acc ->union( i)) ->iterate(i; acc :
String = '' | acc + i.toString()+
       ' {\n\t //[TO DO ]\n}' ) + '\n}';
helper context typescript!TInterface def:
toString() : String=
       'package ' + self.packagerpc.name +
';\n\n'+
       '\npublic interface ' + self.name +
       ' {\n' +
       self. typescript->iterate(i; acc :
String = '' | acc + i.toString())+
       '\n}';
```

Figure-15. ATL code for typescript code generation.

the figures below illustrate the generated code of the class implementing the gesture hand layer based on typescript

The command object provides one method, execute (), that encapsulates the actions and can be called to invoke the actions on the receiver

Implementing the Command interface First things: all Command objects implement the same interface, which consists of one method. We called this method order Up(); however, we typically just use the name execute (). Here's the Command interface:

```
public interface Command {

   public void execute(); /* All we need is one methpd
called execute()
}
```

Figure-16. The Command interface.

The SelectBigMenu class has two methods: selectBigMenu (). Here's how we can implement this as a command:

```
public class SelectBigMenuCommand implements
Command{

   Receiver receiver;

   public SelectBigMenuCommand(Receiver receiver) {
      this.receiver = receiver;
   }

   @Override
   public void execute() {
      receiver.selectBigMenu();
   }
}
```

Figure-17. Implementing a command to select Big Menu.

The ValidateBigMenu class has two Methods: validateBigMenu (). Here's how you can implement this as a command:

```
public class ValidateBigMenuCommand implements
Command {

   Receiver receiver;

   public ValidateBigMenuCommand(Receiver receiver)
{
      this.receiver = receiver;
   }

   @Override
   public void execute() {
      receiver.selectBigMenu();
   }
}
```

Figure-18. Implementing a command to validateBigMenu.

The Client calls HandGesteControl () on an invoker object and passes it the Command object, where it gets stored until it is needed:

```
public class HandGesteControl {

   Command command;

   public HandGesteControl(Command command) {
      this.command = command;
   }

   public void moveHand(){
      command.execute();
   }
}
```

Figure-19. Implementing Hand Geste Control.

The client is responsible for creating the command object. The command object consists of a set of

actions on a receiver Creating a simple test to use the Remote Control.

Here's just a bit of code to test out the simple remote control. Let's take a look and we'll point out how the pieces match the Command Pattern diagram:

```
public class HandGesteControlTest {
   public static void main(String[] args) {
      HandGesteControl remoteControl = new
HandGesteControl();
      SelectBigMenuCommand selectBigMenuCommand
= new SelectBigMenuCommand();
      ValidateBigMenuCommand
validateBigMenuCommand = new
ValidateBigMenuCommand();
     Geste geste = new Geste();
     State state = new State();

     if (("oneFinger".equals(geste)) &&
("menuChoice".equals(state))) {

remoteControl.setCommand(selectBigMenuCommand);
         remoteControl.moveHand();
         state.setNameState("bigmenuselected");
      }
     if (("thumb".equals(geste)) &&
("menuChoice".equals(state))) {

remoteControl.setCommand(validateBigMenuCommand
);
         remoteControl.moveHand();
         state.setNameState("bigmenuvalidated");
      }
   }
}
```

Figure-20. Implementing HandGesteControlTest.

This Figure-21 shows gesture stop for initialize our application Morgia

Figure-21. Symbol stop for initialize application.

This Figure-22 shows the one finger symbol to select Big Menu in our application.

Figure-22. Use symbol one finger for select big menu.

This Figure-23 shows the Thump Up symbol to validate select Big and passed another page.

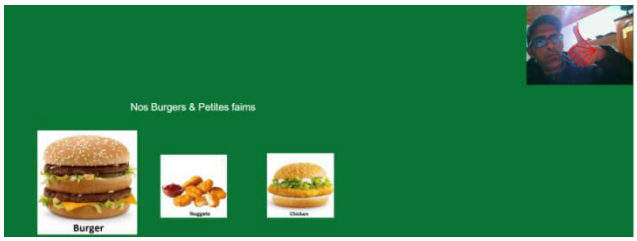

Figure-23. Use symbol thump up for validate select and passed anther page.

7. CONCLUSIONS

This paper examines the most common vision-based gesture recognition methods in recent years, discusses several representative methods of gesture recognition based on conventional methods and gesture recognition based on deep learning, summarizes the benefits and drawbacks of different methods, and highlights current research technological challenges and development trends. We proposed a hand gesture interface for controlling existing GUIs and approach entitled IGMA, is divided into two phases: the modelling phase and the source code generation phase using the IGMATB toolkit. We have proposed also a process to guide GUI improvement based on gesture recognition. This process defines the phases, steps, activities and artifact. The main phases of this process are: 1) Preparation; 2) Elaboration; 3) Construction; 4) Transition.

We are currently finalizing our modeling tool and code generator by incorporating the domain specific command model and context model into our toolbox. Model validation is also planned, which will help to avoid semantic errors in the generated code. We are also putting the last touches on another proposal taking into account several dimensions such as authentication by faces.

ACKNOWLEDGMENTS

This work is partially supported by "Innovation and Technology Transfer Center in Casablanca and Laboratory Research Team RITM (ESTC)".

It is also partially supported by compangy "INNOVATEL INGINEERING SARL".

REFERENCES

[1] K. Qian, Niu, H. Yany. 2013. Developing a gesture based remote human-robot interaction system using kinect [J]. International Journal of Smart Home. 7 (4): 203-208.

[2] S Bhowmick, A. K. Talukdar, K. K. Sarma. 2015. Contimuous hand gesture recognition for English alphabets [C]. International Conference on Signal Processing and Integrated Networks, IEEE. 443-446.

[3] D. Tang, *et al*. 2017. Latent regression forest: structured estimation of 3d hand poseds [J]. IEEE Transaction on Pattern Analysis and Machine Intelligence. 39(7): 1374-1234.

[4] S. Q. Ren, *et al*. 2017. Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39(6): 1137-1149.

[5] Li *et al*. 2017. Joint distance maps based action recognition with convolutional neural networks. Lett. 24(5): 624-628, doi: 10.1109/LSP.2017.2678539 .

[6] X. Liu, G. Zhao. 2019. 3d skeletal gesture recognition via sparse coding of time- warping invariant riemannian trajectories, in: Proceedings of the International Conference on Multimedia Modeling, Springer. pp. 678-690, doi: 10. 29007/xhfp.

[7] X. Liu *et al*. Hidden states exploration for 3d skeleton-based gesture recognition, in: Proceedings of the IEEE Winter Conference on Applications of Computer Vision.

[8] X. Chen, M. Koskela. Using appearance-based hand features for dynamic RGB-D gesture recognition, in: Proceedings of the International Conference on Pattern Recognition.

[9] A. Yao, L. V. Gool, P. Kohli. 2014. Gesture recognition portfolios for personalization, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1923-1930, doi: 10.1109/CVPR.2014.247.

[10] Wu *et al.* Leveraging hierarchical parametric networks for skeletal joints based action segmentation and recognition, in: Proceedings of the IEEE Com-puter Society Conference on Computer Vision and Pattern Recognition.

[11] A. Fernando, E. Gavves, M. José Oramas, A. Ghodrati, T. Tuytelaars. Modeling video evolution for action recognition, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

[12] E. Escobedo-Cardenas *et al.* A robust gesture recognition using hand local data and skeleton trajectory, in: Proceedings of the IEEE Inter- national Conference on Image Processing

[13] A. Joshi, C. Monnier, M. Betke, S. Sclaroff. 2017. Comparing random forest approaches to segmenting and classifying gestures, Image Vis. Comput. 58: 86-95, doi: 10.1016/j.imavis.2016.06.001 .

[14] Ameur, Safa, *et al.* « A Novel Hybrid Bidirectional Unidirectional LSTM Network for Dynamic Hand Gesture Recognition with Leap Motion ». Entertainment Computing.

[15] Santos, Clebeson Canuto dos, *et al*. 2020. « Dynamic Gesture Recognition by Using CNNs and Star RGB: A Temporal Information Condensation ». Neurocomputing, 400, août: 238-54. DOI.org (Crossref), doi:10.1016/j.neucom.2020.03.038.

[16] Mahmoud, Rihem, *et al.* 2020. « Deep Signature-Based Isolated and Large Scale Continuous Gesture Recognition Approach ». Journal of King Saud University - Computer and Information Sciences, p. S1319157820304559. DOI.org (Crossref), doi:10.1016/j.jksuci.2020.08.017.

[17] Almasre, Miada A., et Hana Al-Nuaim. 2020. « A Comparison of Arabic Sign Language Dynamic Gesture Recognition Models ». Heliyon, 6(3): e03554. DOI.org (Crossref), doi:10.1016/j.heliyon.2020.e03554.

[18] 2018. TensorFlow.js. https://js.tensorflow.org/.

[19] 2018. ConvNetJS. https://cs.stanford.edu/people/karpathy/convnetjs/.

[20] 2018. Keras.js. https://github.com/transcranial/keras-js.

[21] 2018. WebDNN. https://github.com/mil-tokyo/webdnn.

[22] 2018. Mind. https://github.com/stevenmiller888/mind.

[23] JSHG. Javascript Hand Gesture Plugin. https: //nhudinhtuan.github.io/jshg/ (current March 5, 2018)

[24] Korhan Akcura, NoTouch.js. A JavaScript Library for Touch-Free Web Browsing.

[25] Egemen Ertugrul *et al*: On attaining user-friendly hand gesture interfaces to control existing GUIs.

[26] Sharma, Ram Pratap and Gyanendra K. Verma. 2015. Human computer interaction using hand gesture. Procedia Computer Science. 54: 721-727

[27] Lee, A-reum, *et al*. 2020. «Enhancement of Surgical Hand Gesture Recognition Using a Capsule Network for a Contactless Interface in the Operating Room». Computer Methods and Programs in Biomedicine, 190, juillet: 105385. DOI.org (Crossref), doi:10.1016/j.cmpb.2020.105385.

[28] Mahmoud Rihem, *et al*. 2020. « Deep Signature-Based Isolated and Large Scale Continuous Gesture Recognition Approach ». Journal of King Saud University - Computer and Information Sciences, p. S1319157820304559. DOI.org (Crossref), doi:10.1016/j.jksuci.2020.08.017.

[29] Meghana M., *et al*. 2020. « Hand Gesture Recognition and Voice Controlled Robot». Materials Today: Proceedings, 33: 4121-23. DOI.org (Crossref), doi:10.1016/j.matpr.2020.06.553.

[30] Huang, Yao, et Jianyu Yang. 2021. « A Multi-Scale Descriptor for Real Time RGB-D Hand Gesture Recognition ». Pattern Recognition Letters, 144, avril: 97-104. DOI.org (Crossref), doi:10.1016/j.patrec.2020.11.011.

[31] X. Chen *et al.* Motion feature augmented network for dynamic hand gesture recognition from skeletaldata, Sensors (Basel, Switzerland).

[32] Sarkar A., Gepperth U., Handmann T. 2017. Kopinski Dynamic handgesturere-cognition for mobile systems using deep LSTM, in: International Conference on Intelligent Human Computer Interaction, Springer, Cham, pp. 19-31. Doi: 10.1007/978-3-319-72038-8_3.

[33] Eclipse EMF. Available: http://www.eclipse.org/modeling/emf/, 2014.

[34] Steinberg D., Budinsky F., Paternostro M. & Merks. 2008. Ed. EMF: Eclipse Modeling Framework. Addison-Wesley Professional.

[35] Mohamed M., Romdhani M. and Ghedira K. 2007. MOF-EMF Alignment. In the International Conference on Autonomic and Autonomous Systems, Athens, Greece. pp. 1-7.

[36] Eclipse GMF. 2014. Available: http://www.eclipse.org/modeling/gmp/.

[37] Richard C. Gronback. 2009. Eclipse Modeling Project: A Domain Specific Language (DSL) Toolkit. Addison-Wesley.

[38] Eclipse Acceleo. 2014. Avalable: http://www.eclipse.org/acceleo/. 2018. TensorFlow.js. https://js.tensorflow.org/.

[39] Navotas IC, Santos CN V., Balderrama EJM, *et al.* 2018. Fish identification and freshness classification through image processing using artificial neural network.