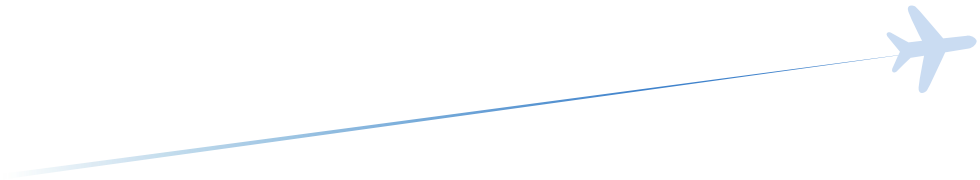
EXPLORATORY RESEARCH



# State of the Art

**D2.2**

**TaCo**

| | |
|---|---|
| **Grant:** | **699382** |
| **Call:** | **H2020-SESAR-2015-1** |
| **Topic:** | **Sesar-01-2015 Automation in ATM** |
| **Consortium coordinator:** | **Deep Blue** |
| **Edition date:** | **31 January 2017** |
| **Edition:** | **01.00.00** |

Founding Members

EUROPEAN UNION EUROCONTROL

SESAR JOINT UNDERTAKING

## Authoring & Approval

### Authors of the document

| Name/Beneficiary | Position/Title | Date |
|---|---|---|
| Stéphane Conversy/ENAC | Project Contributor | 19/12/2016 |
| Mathieu Cousy/ENAC | Project Contributor | 19/12/2016 |
| Jérémie Garcia/ENAC | Project Contributor | 19/12/2016 |
| Mathieu Poirier/ENAC | Project Contributor | 19/12/2016 |
| Stéphanie Rey/ENAC | Project Contributor | 19/12/2016 |

### Reviewers internal to the project

| Name/Beneficiary | Position/Title | Date |
|---|---|---|
| Damiano Taurino/DBL | Project Coordinator | 19/12/2016 |
| Johan Debattista/MATS | Project contributor | 19/12/2016 |

### Approved for submission to the SJU By — Representatives of beneficiaries involved in the project

| Name/Beneficiary | Position/Title | Date |
|---|---|---|
| Damiano Taurino/DBL | Project Coordinator | 30/01/2017 |
| Stéphane Conversy/ENAC | Project Contributor | 30/01/2017 |
| Johan Debattista/MATS | Project contributor | 30/01/2017 |

### Rejected By - Representatives of beneficiaries involved in the project

| Name/Beneficiary | Position/Title | Date |
|---|---|---|

## Document History

| Edition | Date | Status | Author | Justification |
|---|---|---|---|---|
| 00.00.01 | 19/09/2016 | Draft | Stéphane Conversy | First version |
| 00.01.00 | 19/12/2016 | Draft | Stéphane Conversy | Consolidated draft for SJU review |
| 01.00.00 | 31/01/2017 | Final | Stéphane Conversy | Final version including the integration of SJU's review |

THIS DELIVERABLE IS PART OF A PROJECT THAT HAS RECEIVED FUNDING FROM THE SESAR JOINT UNDERTAKING UNDER GRANT AGREEMENT NO 699382 UNDER EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME

# TaCo

TAKE CONTROL

This state of the art is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 699382 under European Union's Horizon 2020 research and innovation program.

## Abstract

This work is concerned with the end-user programming of reactive and autonomous systems. The main objective of this state of the art is to collect and classify input, techniques and innovative approaches coming from the fields that have conducted the more advanced studies on the topic. The document includes an analytical survey pertaining to Integrated Development Environments (Educational and Professional), Video Games, as they often include various means of automation for the gamer, and Music tools that range from preparation of scheduling of musical events to live, action-based sequencer during performances. It also includes as much as possible considerations on interesting representations of, and interaction with automation. The result contributes to the definition of a common framework for the design of automated airport solutions.

# Table of Contents

# 1 Introduction

## 1.1 Purpose and Scope of this document

TACO aims to define an automated system sufficiently powerful to both accomplish complex tasks involved in the management of surface movements in a major airport and self-assess its own ability to deal with non-nominal conditions. When needed, such system should be sensitive enough to transfer responsibilities for traffic management back to the controller, in a timely and graceful manner and in way that makes him/her comfortable with the inherited tasks.
For graceful take-over, the system must be visible and understandable. One of the goals of End-User Programming is to provide end-users with more visible and understandable means of programming. This suggests that the mechanisms of the automation should be programmed with and by the controllers. The document presents the work related to the end-user programming of reactive and autonomous systems.
The purpose of this state of the art is not to choose a particular tool for use in TaCo. Rather, its purpose is to gather some of the requirements and features devised by the authors of work related to End-User Programming, and inform the design of TaCo. The specific requirements of the TaCo project are investigated in the problem definition work package. However, the project proposal defines a number of pre-requisites that can be expressed with these requirements:

- The programming environment should be graphical, mostly with a 2D representation
- The programming environment should be amenable to ATC concerns.
- The programming environment should be usable by non-Computer Science specialists.

Due to the project's scope and preliminary requirements, this state of the art is not comprehensive. Rather, it includes older and recent works that pertain to End-User Programming and that fulfill one or more of the previous requirements.

## 1.2 Deliverable Structure

This project is not the first of its kind in the ATC activity. The related work will thus discuss some of the ATC projects (notably from SESAR) that include or are related to the programming of automation. Nonetheless, as a TRL-1 project, it is still time to benefit from completely external disciplines. Even if most of them are not devoted to safety critical environments, studying these disciplines will enable us to gather fresh, innovative and efficient features that will inform the design of the tools and scenarios for TaCo.

Programming is often supported by so-called Integrated Development Environment. The related work contains a review of some professional IDEs (especially those that are used in the aeronautical industry). It will also include a review of IDEs target at education, since those tools are often concerned with usability and learnability.
Video Games are also interesting for several reasons. Like ATC tools, videogames such as Real-Time Strategy games use highly interactive graphics to enable gamers manage the course of the game. They also often include various means of automation for the gamer, during live action. Finally, many of them include an IDE to enable gamers to program advanced behavior. They thus have requirements that are similar to ours.
Finally, music tools share much of TaCo's concerns. Music tools range from preparation of scheduling of musical events to live, action-based sequencer during performances. As a pioneering discipline mixing programming and interaction, it has much to inform TaCo.
Most related existing states of the art include considerations on traditional way of programming such as instructions or functions. The present state of the art is more focused on the end-user programming of automation and reactive behavior. It also includes as much as possible considerations on interesting representations of and interaction with automation.

## 1.3 List of Acronyms

| | |
|---|---|
| ATCO | Air Traffic Controller |
| ATM | Air Traffic Management |
| ACDM | Airport Collaborative Decision Making |
| IDE | Integrated Development Environment |
| LOA | Level of Automation |
| EUP | End-User Programming |
| EUD | End-User Development |
| EUSE | End-User Software Engineering |
| SESAR | Single European Sky ATM Research |

## 1.4 Note

The opinions expressed herein reflect the author's view only. Under no circumstances shall the SESAR Joint Undertaking be responsible for any use that may be made of the information contained herein.

# 2 Motivation and Definitions

Automation is one of the key solutions proposed and adopted by SESAR to tackle the challenges coming from the increase of capacity and complexity of the future ATM system. If automation aims at substantially reducing controllers' task load per flight through a significant enhancement of integrated automation support, it also poses a number of usability problems to their users.

## 2.1 Requirements for Automation and Usability

Levels of Automation (LOA) can be analyzed according to four information-processing stages: (a) information acquisition, (b) information analysis, (c) decision making, and (d) action, with each stage having its own LOA scale [24]. For stages (c) decision making and (d) action, "Field studies and surveys have revealed the low sense of trust and substantial degree of confusion that pilots have regarding the operation of [...] "opaque" systems [24]. [...] The operator's use of automation to control a simulated plant is related to his or her momentary trust, which in turn is related to the type and frequency of faults. If operators' confidence in their own ability to control was greater than their trust in automation, they tended not to use it. When the reverse was true, they tended to use automation."
The "opacity" mentioned in [24] is related to the absence of operator's visibility on the behavior of the program. Since the behavior is opaque, and so hard to predict, the operator must blindly trust the automation. Consequently, they might prefer to directly control the operation, instead of relying on difficult-to-predict automation. This mechanism leads to a dichotomy in the usage of automation (all or nothing [28]), that prevents the advantages of designing and implementing a flexible collaboration between user and automated solutions, that includes different LOAs for different tasks. [27]
Hence, the visibility of the automation, its status, and its future evolution is important, as it substantially affects the situational awareness of the user. In addition, visibility might help operators recover a situation. When automation fails, operators must engage into a Knowledge-based type of behavior, as opposed to Skills or Rules-based [26] that is typical a nominal situation. This implies that operators gather information, make sense of it, understand the causes and consequences and counter-act appropriately, leading to a substantial increase of his/her cognitive workload. An important aspect of this problem-solving activity for the operator is to have in mind the right conceptual model of the automation, together with a

correct understanding of the status of the whole system [23]. Hence, visibility of the states of operations and automation, joint to the visibility of the program, are of crucial importance.
End-user programming is a domain of computer science that is concerned with such property of visibility of automatic behavior. The following sections describe what End-user programming is, together with a number of related concepts.

## 2.2 Definitions

### 2.2.1 End-user programming

Programming is the process of planning or writing a program i.e. « a collection of specifications that may take variable inputs, and that can be executed (or interpreted) by a device with computational capabilities. » [17] As said before, automation relies on programs. If programs must be visible as assumed above, end-user will manipulate them. This might only consist in starting the program or stopping it. However, we envision more control of the programs by the end-user, and any other type of actions on the program will modify it and can be considered as programming.
End-user programming is defined as "programming to achieve the result of a program primarily for personal, rather public use." [17] End-user programming does not define skills: "A number of connotations of the phrase have emerged in research, many using it to refer to "novice" programming or "nonprofessional" programming, or system design that involves the participation of end users. [...] However, this definition also includes a skilled software developer writing "helper" code to support some primary task" [17] .
Still, here the primary task occurs in a domain of practice, ATC, where end-users might not be proficient in programming (see below). However, the benefits of having an end-user programming lie in coupling a deep knowledge of the involved activities (user) and the possibility to program how the system supports them (programming). As written in [17] , "this is the difference between a banker writing banking software and a professional programmer writing banking software. The banker would have to learn to program, whereas the professional would have to learn banking concepts." A domain such as ATC is "likely to involve different types of computational patterns and different software architectures". This is exactly what we want to explore: to which extent the right conceptual model and the right framework together with the right HMI may enable stakeholders (ATCOs, ATCOs with programming skills, Professional programmers) to design better (locally-tuned) automated systems?

### 2.2.2 End-user development and software engineering

A number of notions related to End-user programming (EUP) exist. End-User Development (EUD) can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [19]. End-User Software Engineering (EUSE) is

concerned with technologies that collaborate with end users to improve software quality [17] . EUSE emerged to deal with different phases of software life-cycle, including maintenance. More precisely, EUSE is concerned with:
(1) Requirements. How the software should behave in the world.
(2) Design and Specifications. How the software behaves internally to achieve the requirements.
(3) Reuse. Using preexisting code to save time and avoid errors (including integration, extension, and other perfective maintenance).
(4) Testing and Verification. Gaining confidence about correctness and identifying failures.
(5) Debugging. Repairing known failures by locating and correcting errors.

TaCo is primarily concerned with End-User Programming, since we envision that ATCOs will at least manage previously designed automation. However, we also have to take into account EUSE concerns if we want end-users to interact with automation of pre-programed automatic behavior in advance. Thus, programming almost always involve testing and debugging. Any tool designed to support programming must at least support them.

## 2.3 Type of Users and Programming Contexts

A question that may arise is that of the proficiencies of the targeted users at programming their own automation. If ATCos are professionally skilled to manage traffic, they may be not literate enough in thinking about, specifying or programming software that will be usable, safe and performing enough to effectively support their activity. In our opinion, there are two dimensions to this problem: programming level, and programming time.

The first one is the programming level. ATCos are trained as specialists of traffic management. The ATCo curriculum does not require any competencies in computing. However, we have to look into the future of ATCos education. Many of them studied science at a high level during high-school and for some at college. They thus have the scientific and reasoning abilities that are required to study the programming of the behavior of a software. In addition, there are numerous social debates and even decisions to include computer science lectures very early in the education process. In the near future, most ATCos will be educated in computer science all along their studies (at least in several Member States, France for

instance[1]). This will be a common knowledge and know-how, as much as humanities, physics, or mathematics.

Nevertheless, not all ATCos will be able to program automation. We envision rather that there will be a continuum between highly-skilled computer science professionals, ATCos well-versed into programming and willing to do it, or ATCo's as "simple users". All these people will collaborate to specify and program the automation, as much as they do now. However, our vision is to facilitate the collaboration through more usable programming concepts and tools that are more direct and more compatible with the targeted activity.

The second dimension is the time at which programming will occur. Again, there is a continuum between offline and very early design and programming of the systems, configuration on the actual settings, online and live programming of automation, and action-only during the control activity. The tools and scenarios that we envision will include the various situations of programming that will occur during the production and the utilization phases of the system life-cycle.

[1] Computer Science is now taught in « collège », « lycée » and « classes préparatoires scientifiques », and discussions started on « primaire »

# 3 Professional IDEs

All programming activities require tools to specify, produce and run program. An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development [48]. A large number of Integrated development environments (IDE) were designed to support the activity of professional programmers, by providing them with a comprehensive set of editors and components that they can use to specify and run applications. Some of the features and language constructs that are supported in these tools may be reused in the TaCo project.

## 3.1 Instructions flow

Automator [65] provides a graphical user interface for automating tasks without knowledge of programming or scripting languages (Figure 1). Tasks can be recorded as they are performed by the user or can be selected from a list. The output of the previous action can become the input to the next action. The workflow is then a sequence of consecutive actions.

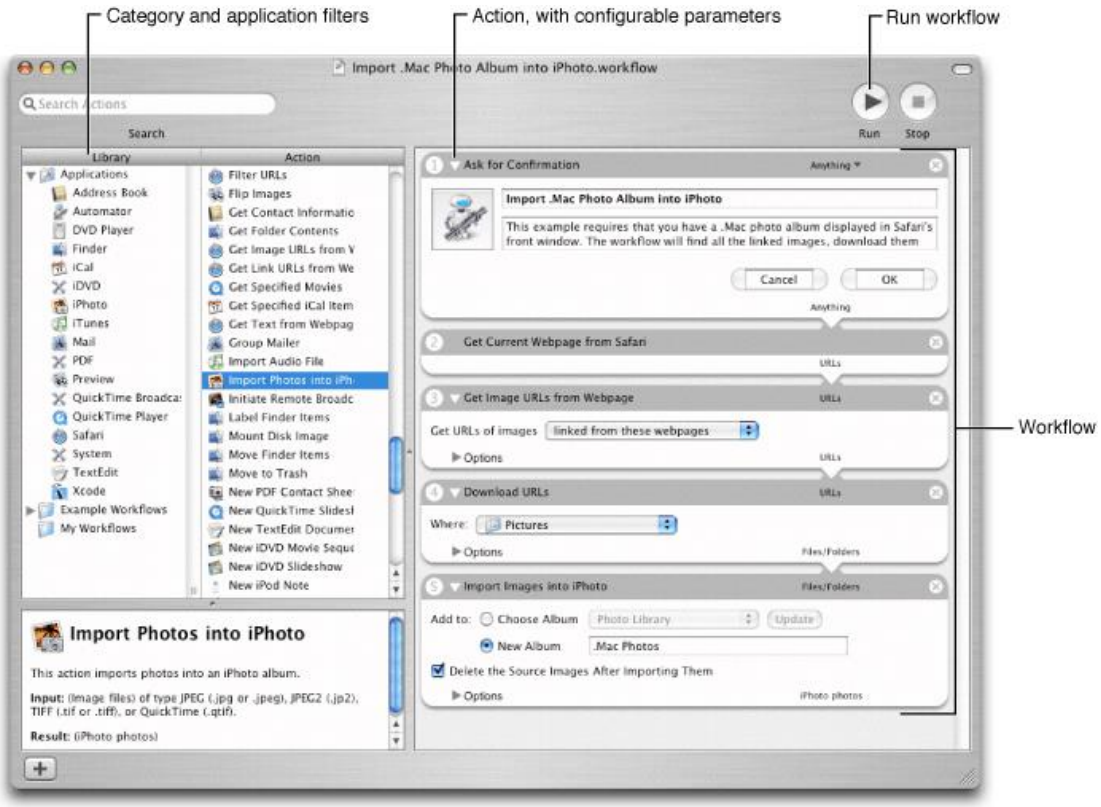



Figure 1: Apple Automator ®

## 3.2 Data flow

Other tools rely on the unifying concept of data. For example, Laboratory Virtual Instrument Engineering Workbench (LabVIEW [59]) is a system-design platform and development environment for a visual programming language from National Instruments. The graphical language is named "G" and is a dataflow programming language. Execution is determined by the structure of a graphical block diagram on which the programmer connects different nodes by drawing wires (Figure 2). These wires propagate variables and any node can execute as soon as all its input data become available.

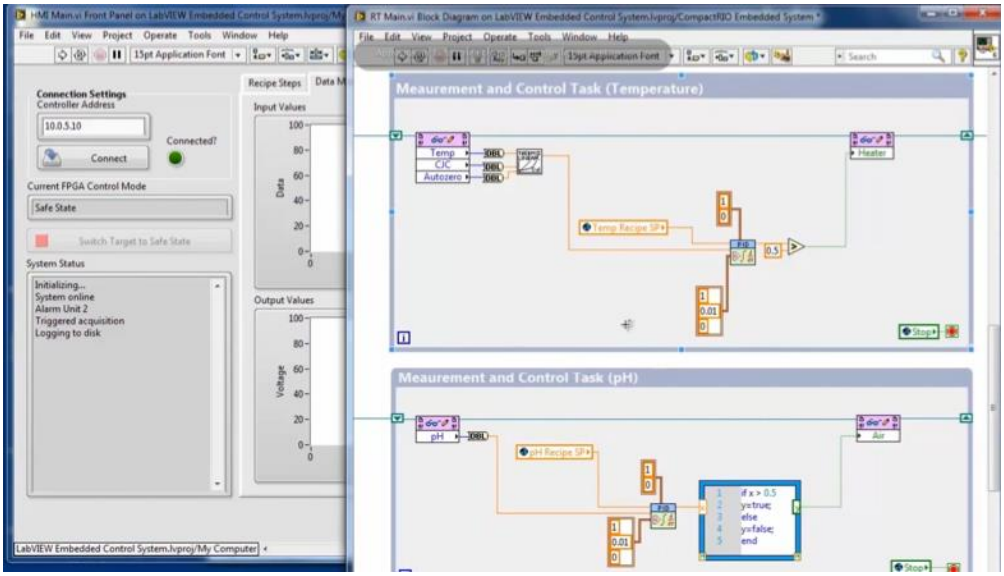



Figure 2: LabView ®

With a more recent design, NoFlo [66] is a JavaScript implementation of Flow-Based Programming (FBP), a programming paradigm that defines applications as networks of "black box" processes, which exchange data across predefined connections by message passing (Figure 3). In this interface a top-level mini-map makes it easy to track the location in the whole context and wires are colored to help follow the multiple crossing flows.

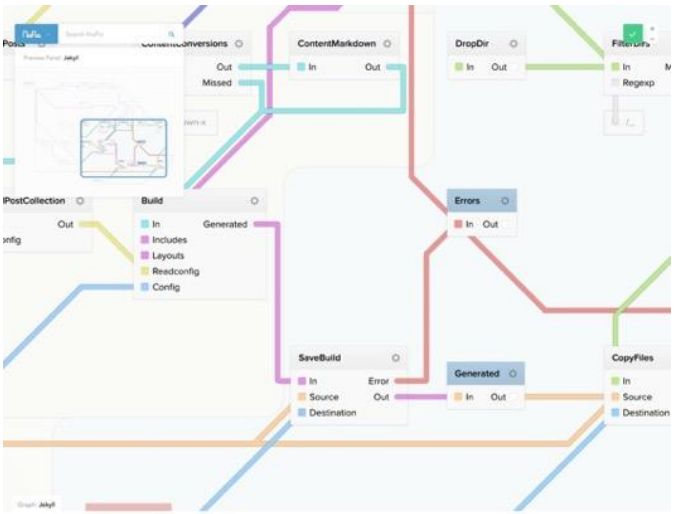



Figure 3: NoFLo – flow-based programming

## 3.3 Rules trigger

Another way to program tasks with more reactive behaviors and without knowledge of programming is the reactive programming paradigm, where the user can think “when this kind of thing happens, then trigger this action”.

The most representative is IFTTT [67], an abbreviation of "If This Then That". It is a free web-based service that allows users to create chains of simple conditional statements, called "applets". An applet is composed of a trigger (« This », Figure 4) based on web services events, and an action (« That », Figure 4) executed also thanks to web services. The programming interface consists of lists and sub-lists of triggers/actions in which the user can choose the desired item and access specific parameters.

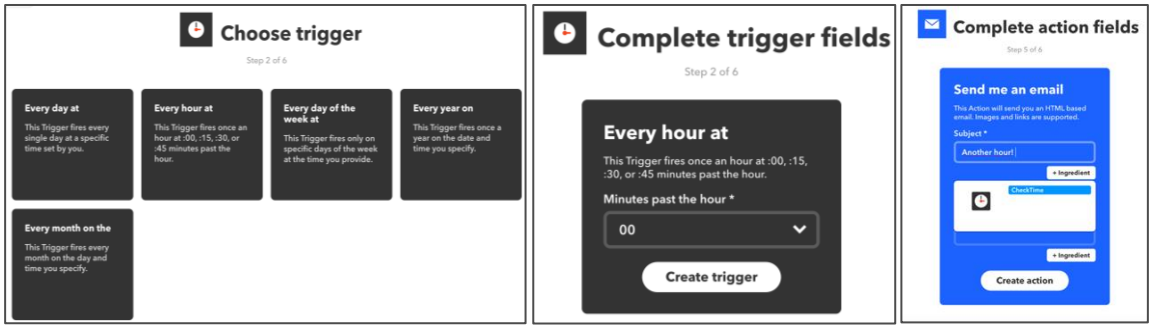



**Figure 4: IFTTT ® – Screenshots of the successive pages to define an automation.**

Another example is IntuiFace [68], a platform for creating, deploying and measuring interactive digital experiences for non-coder. Reactive behaviors are built thanks to triggers and actions chosen in lists, with a “When _ Then _” metaphor reminder on the top of the screen and lists and sub-lists of trigger/actions and parameters.

## 3.4 Hybrid IDEs

In aeronautics, two applications are mainly used to help developers implement the cockpit HMI and their behaviors. These applications rely on several programming concepts like state-machines and dataflow.

VAPS XT [33] is an embedded graphics tool that allows users to define both object appearance and display logic in a single graphical editor. VAPS XT includes several windows: a project view, an implementation view (graphical tree), a graphical view / runtime view, a property panel, a UML state-chart editor and a dataflow table (Figure 5).

SCADE Display [38] is a product line of the ANSYS Embedded software family of graphics design and development environments for embedded Human Machine Interfaces. In Figure 6, data flow is illustrated through the connection of a logical property from a dictionary (left) to the property of a graphical component (right).


Figure 5:Vaps XT ®

SCADE Suite (Figure 7) allows for designing the control logic associated with graphical HMIs designed in SCADE Display. Most of the editing is done in a graphical form. An operator is a basic building block with inputs and outputs. More complicated operators are done by assembling pre-existing ones. The operators (in blue) can be mathematical, comparison, logical, structure/array, time, choice, bitwise or higher order operators. The Scade language is formally define, and offers hierarchical state machines (in pink) that can be fully mixed with dataflow behavior.

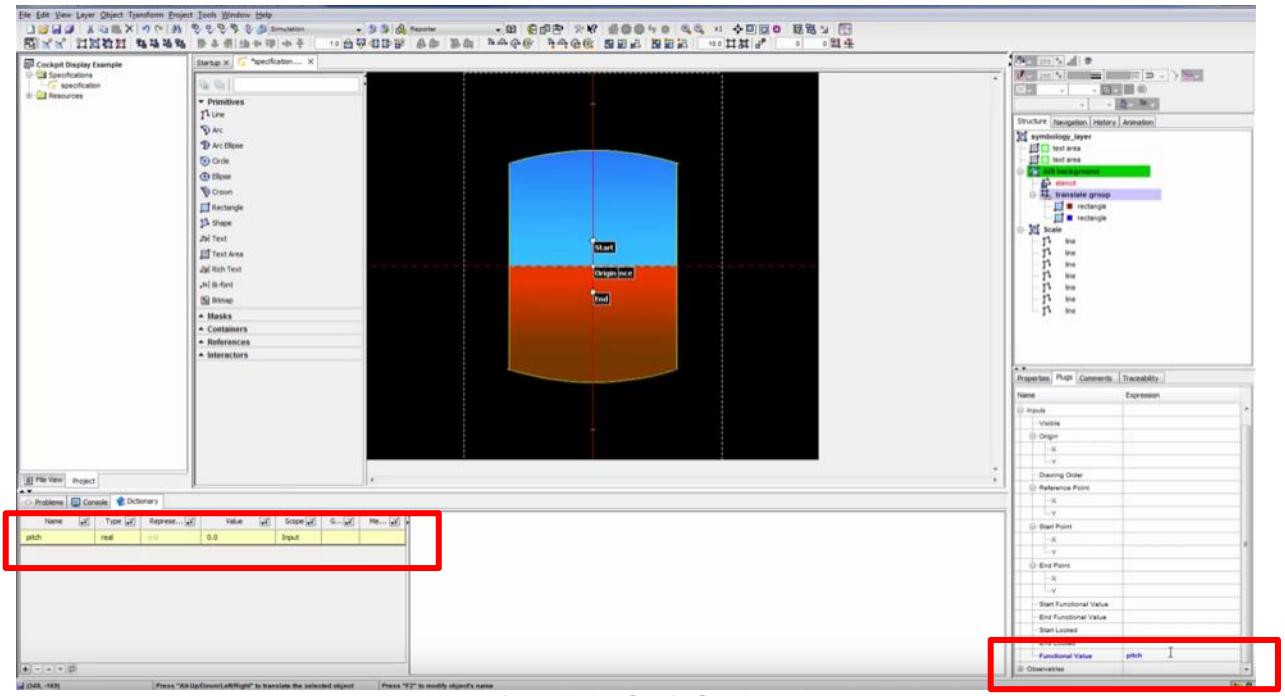


Figure 6: Scade Display ®

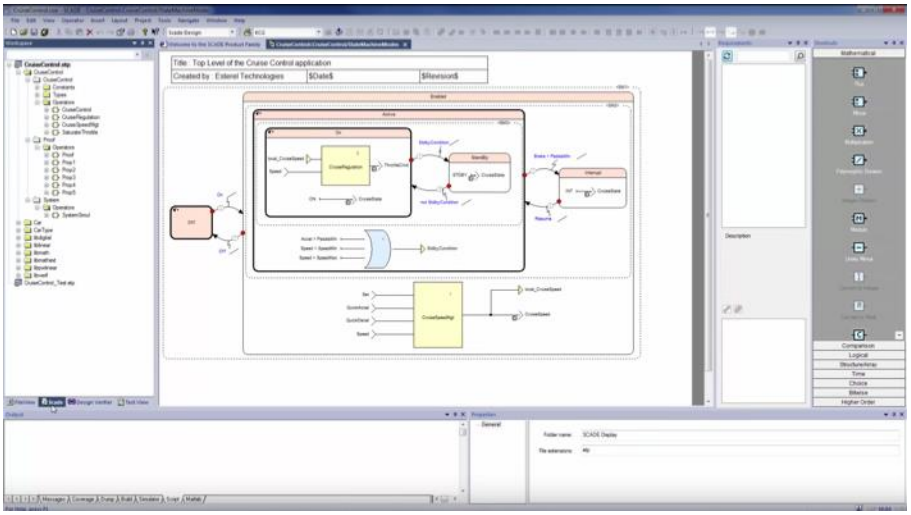


Figure 7: Scade Suite ®, in blue operators, in pink hierarchical state machines

Properties from Scade Suite and Scade Display can be connected in order to co-simulate both graphics and logics (Figure 8) and see dynamically the changes occurring on the system depending on the data.

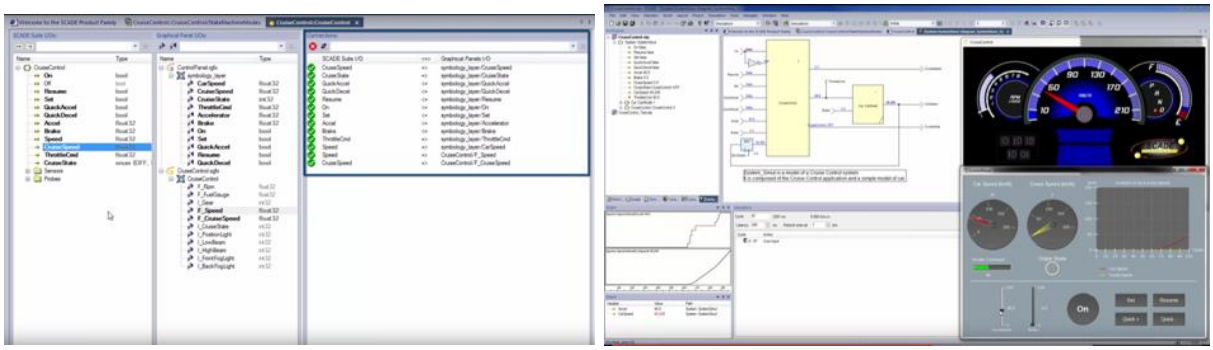


Figure 8: a) Connect Scade Suite ® and Scade Display ® properties for b) co-simulation

# 4 Educational IDEs

Some IDEs are specialized in facilitating the learning phase of programming. Much of them are oriented towards children. Still, their features may be of great interest for our project, as their designers were careful to provide detailed visualization and interaction to make it as easy as possible for children to understand programming.

## 4.1 Representing the program

Some attempts of programming vulgarization for children replace texts by icons to visually represent the traditional elements of programing languages. For example Little Wizard [49], a development environment for children, uses icons for variables, expressions, loops, conditions and logical blocks that children can drag and drop together to compose the program "sentences" (Figure 9). This kind of application reconsiders the algorithmic vocabulary, but without adapting the manipulated core concepts for children understanding.

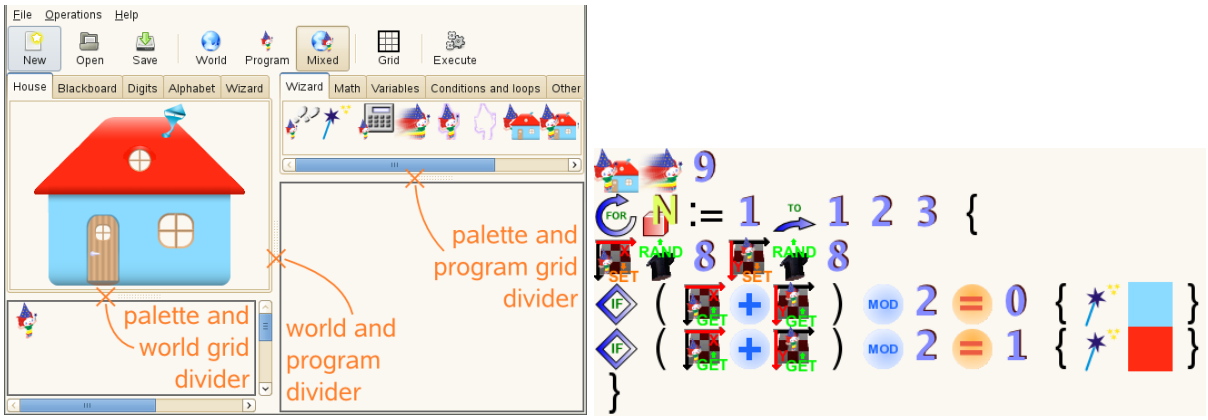



Figure 9: Little wizard editor

### 4.1.1 Instructions flow

Other projects try to lower the step of access to the world of computer programming by representing a program as a combination of command blocks. The child can drag and plug objects, parameters, actions or control blocks in order to create an imperative script eventually

executed sequentially. The resulting script is used as a computer program, as robot instructions or to create images, 3D models or even game levels…

Most of these projects are inspired by LOGO [1] and the programing of the Turtle Robot, a simple robot controlled from the user's workstation that is designed to carry out the drawing functions assigned to it using a small retractable pen. The computer equivalent, Turtle Art [50], lets children make images with their computer. The Turtle follows a sequence of commands. The child specifies the sequence by snapping together puzzle-like blocks (Figure 10). The blocks can instruct the turtle to draw lines and arcs, draw in different colours, go to a specific place on the screen, etc. There are also blocks that let children repeat or name sequences. Other blocks perform logical operations.

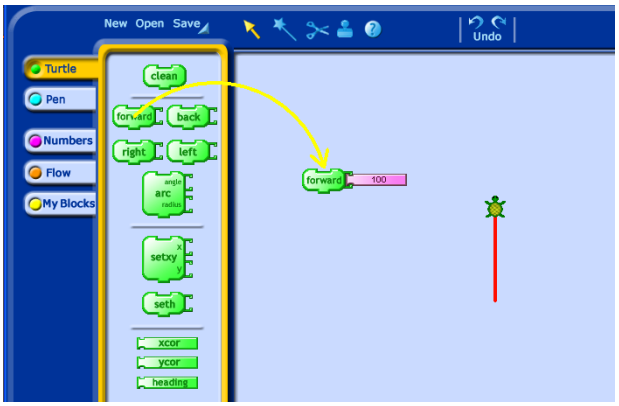



Figure 10: TurtleArt

Squeak Etoys [51] is an authoring environment to create computer programs using a scripting language based on blocks and containers. Figure 11 displays two traditional programing concept transpositions to Squeak: a) a conditional statement - a specific kind of container with holes for if then else to plug blocks into it and b) a dataflow - the connection of object parameter blocks into a script container (Figure 11. b).


Figure 11: Squeak Etoys. a) conditional statement b) dataflow

Following the same idea, the most used application in children initiation to computer science is Scratch [52], a project of the Lifelong Kindergarten Group at the MIT Media Lab. It allows children to program their own interactive stories, games, and animations and share their creations with others in the online community. The scratch interface (Figure 12) is composed of: a "stage" to run the program, a "sprite list" for each animated object, a "block palette" organized by categories and a "script area" to snap blocks together.

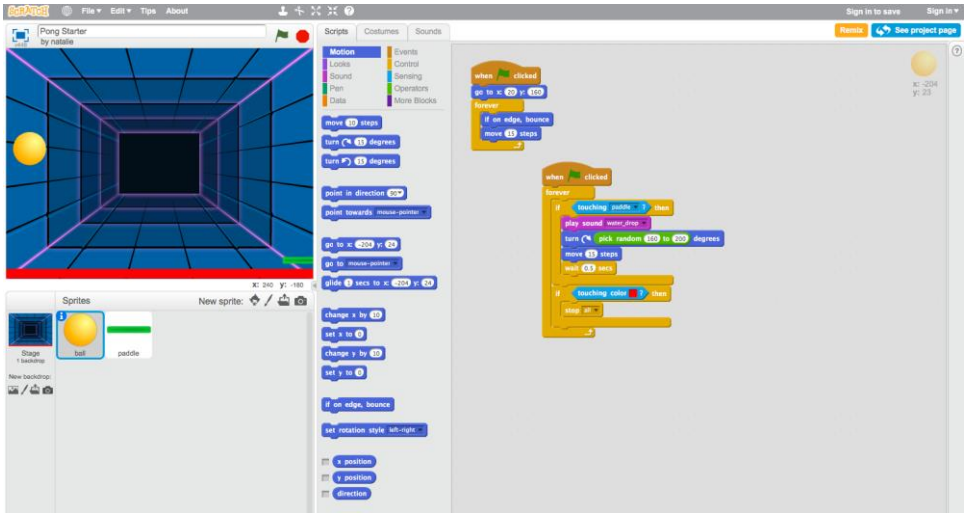


Figure 12: Scratch – C-shaped containers to hold the blocks

This system is also used by the MIT for App inventor [53] in order to create mobile applications. This representation of assembling blocks within C-shaped control structure (like loops or if/then/else) is widely used, for example in Beetle Blocks [54] for 3D shapes, in Google Blockly [55] for puzzle games or in Tynker [56] for creating Minecraft levels.

But the sequential execution of the blocks one after another can make uneasy the use of data sets and the programing of reactive behaviors. For example, in the following program (Figure 13) created with Lego Mindstorm EV3 **[57]**, the user has to define a "wait" loop behavior to

trigger a sound when the robot is touched. Such active wait prevents reactions depending on multiple instances of input.

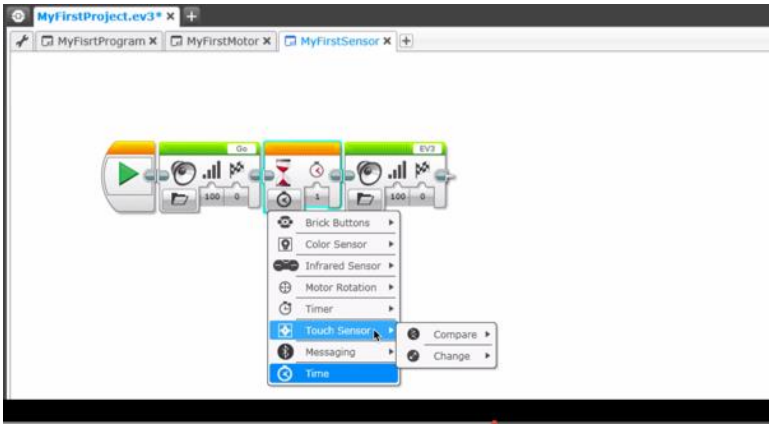



**Figure 13: Lego Mindstorm EV3 ® – wait until a touch is detected to play a sound**

### 4.1.2 Dataflow

The previous version of the same Lego Mindstorm programming tool, the NXT [58], solved this problem of data management with the concept of dataflow. This software was based on LabVIEW [59]: not only the user can plug bricks together but he can also connect the output of a brick to be the input of the next brick using a link (Figure 14) for the data to be further processed.

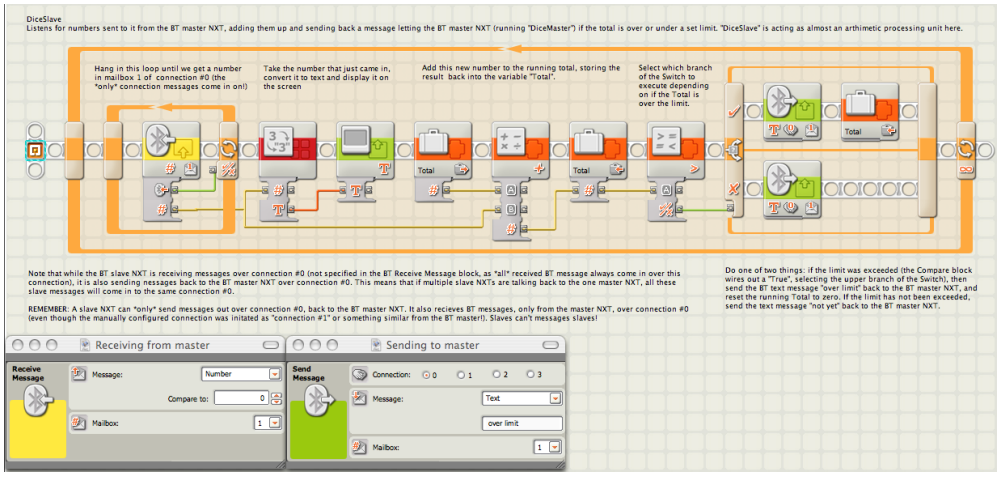



**Figure 14: Lego ® NXT-G dataflow**

### 4.1.3 Rules trigger

A solution to teach reactive programming to children is the trigger – action programming concept. Kodu [60] is a visual programming tool which builds on ideas begun with Logo [1] and other current projects such as AgentSheets [61], Squeak [51] and Alice [62]. The user programs the behaviors of characters in a 3D world, and programs are expressed in a rule-based system or language, based on conditions and actions (Figure 15).


**Figure 15: Kodu a) choose the character b) when this - do that c) choose the when action d) when sub-action e) sequences of when – do rules**

### 4.1.4 State Machine

Another solution to program reactive behavior is the use of state machines. Dash and Dot [63] teach kids how to program toy robots by using 5 apps on tablets and phones. One of these, the Wonder application [64] allows children to create state machines to control the robots.
Dash and dots robots can do all sorts of behaviors like lights, sounds, movement and animations. The user can link these behaviors together so they happen one after another, or go to different behaviors. Cues are what go on the links between the behaviors, they look like arrows and make the program go. Cues can be all different ways to make Dash or Dot go between their behaviors: a button press, a hand clap or a timer for an example. All of these behaviors, links and cues together make a state machine (Figure 16).


Figure 16: Wonder Workshop ® - Wonder application

All these projects are dedicated to simplifying the creation of interactive programs, whether these programs are dedicated to run a robot, animate a virtual character in a game, create an animation or an image. But there are very few hints on how the program will actually run and what are the causal links between the blocks/lines/sequences and the final behavior.

## 4.2 Interacting with the program

In some of the previous software, some visualization of the result of a program is possible by executing small chunks of code. For example, in Scratch the child can click on each block to see the resulting character behavior on the stage. This allows children to learn to code by doing iterations, but it is not sufficient to give them full understanding of the program behavior.
Bret Victor wrote an article [4] on how to design Integrated Development Environment (IDE) and programming language for enhancing the programmer understanding. The article describes a set of principles about the vocabulary, flow and states visualization, and on supporting coders create behavior by reacting to immediate feedback and abstracting. It also describes the properties of the language itself that are required, such as the connection to the programmer's world.

### 4.2.1 Transparent vocabulary

The environment should make the meaning transparent, so the learner can concentrate on high-level concepts, instead of the vocabulary. It must also explain the program concept in

context, annotate the data and not just the code. For example, in Figure 17, the label connects the code and its output and the red line shows the result of the argument modification on the running program.

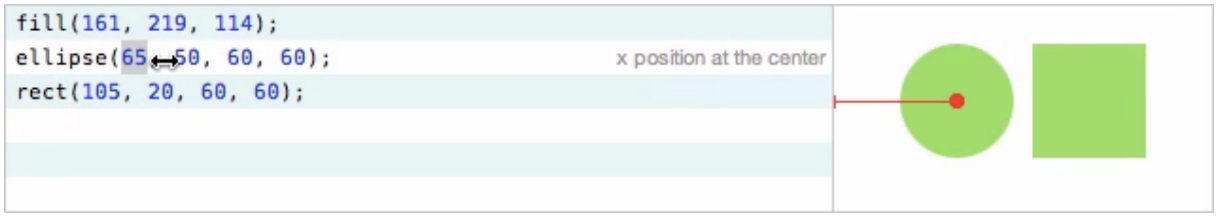



Figure 17: transparent vocabulary.

### 4.2.2 Flow and states visualization

The IDE must enable the learner to follow the program flow, by allowing her to control time and see the patterns across time. The environment can enable the programmer to explore forward and backward at her own pace, and even explore the time at multiple granularities, such as frames or event responses (Figure 18).

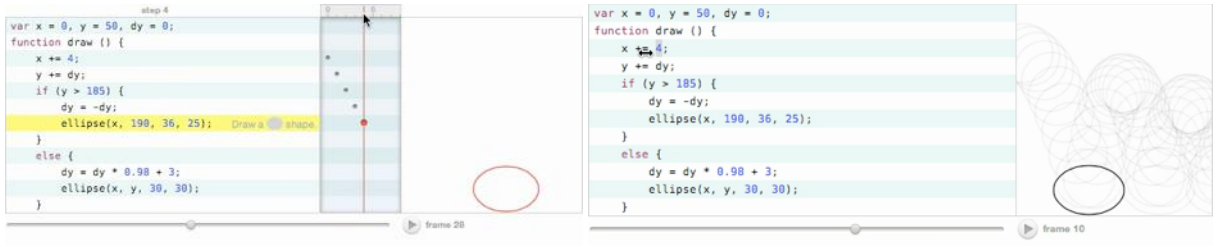


Figure 18: a) a timeline for line by line execution and a slider for frame by frame. b) the entire path of the ball

The learner must also see the data, the effect of code on the data and all the resulting states of the data, as shown in Figure 19.

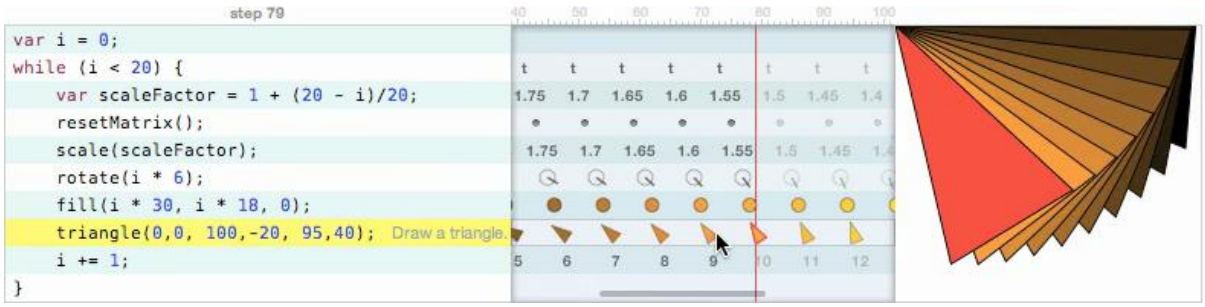



Figure 19: Show the states: a timeline that depicts all the data calculated in the flow

### 4.2.3 Reacting and abstracting behaviors

The IDE must support the developer in creating by reacting to immediate feedback, for example by autocompleting with significant default parameters (Figure 20 a) or allowing to "dump the parts bucket onto the floor" i.e. showing all possible constructs (Figure 20 b) to make programmers react to raw material and spark new ideas.

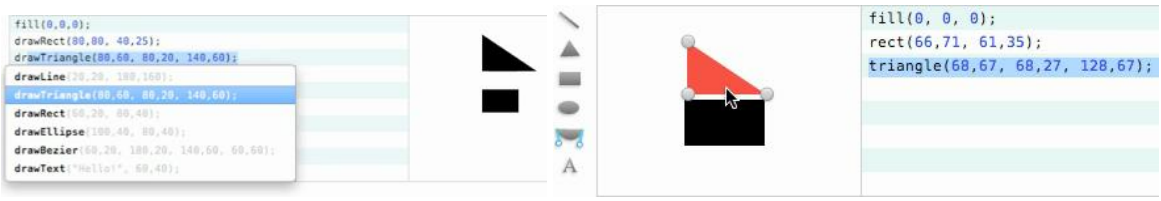



Figure 20: left) default arguments right) parts bucket

It must also allow the developer to control the lower-level details, before handing off the control to an abstraction and move to a higher level of control (Figure 21).

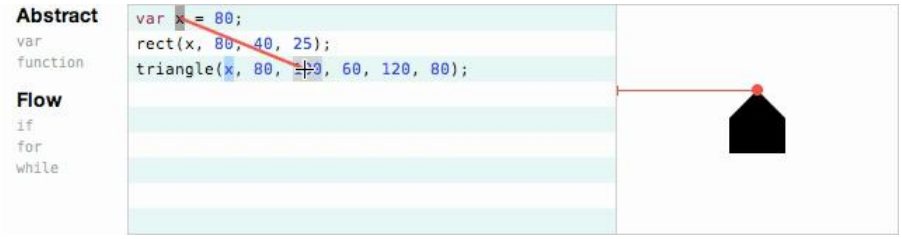



Figure 21: Bret Victor. Abstracting behavior

# 5 Video games and robotics

Like ATC tools, videogames such as Real-Time Strategy games use highly interactive graphics to enable gamers manage the course of the game. They also often include various means of automation for the gamer, during live action. Finally, many of them include an IDE to enable gamers to program advanced behavior. Such features, more and more targeted to non-specialist programmers, may inform the design of the TaCo project.

## 5.1 Programming game mechanics and animations

Several environments have been created to help develop games with high-level programming concepts that are closer to the game designers' concerns. In particular, several tools such as Unreal Engine [39], Unity [40] or Euphoria [41] rely on visual programming techniques to facilitate the definition of dynamic behaviors, rules and motions. For example, Figure 1 shows a program with the Blueprints Visual Scripting system used in Unreal Editor [39] in which the designer defined a rule to make the character jump when it is hitting a white block on the ground. In such visual programs, the boxes represent actions or conditions and are connected via wire to link them during the execution. Several mechanisms are made to facilitate reification and reuse of the scripts previously created in different contexts. For example, the script to jump when hitting a white block can be saved as a new block and reused in another level of the game.

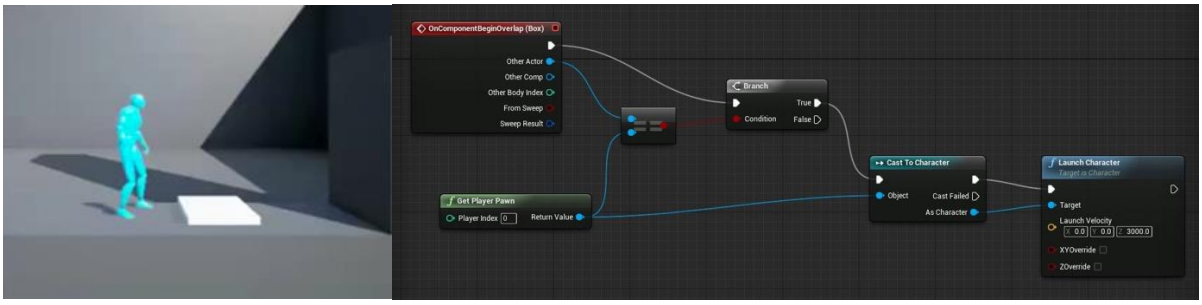



**Figure 1: Preview and visual program of a character jumping when hitting white blocks in Unreal Engine ®**

Unity and Euphoria provides different forms of visual programs, state machines. With such visual representations, developers graphically define states of the postures, or visual expressions and then define transitions that will be triggered by specific events. Unity also provides direct manipulation techniques such as drag and drop to add physical constraints such as gravity or bouncing in specific locations of the map or to apply them to virtual objects in a preview 3D scene. Instead of using spatial representations, Euphoria provides a timeline

in which specific animations can be dragged to define possible complex sequences of motions.
Those environments can be considered as Integrated Tools Environment as much as traditional ones, even if they are more tightly tied to the lower-level 3D run-time engine that will eventually run the game.

## 5.2 Defining positions and actions of semi-autonomous agents

Real Time Strategy (RTS) games enable players to control multiple instances of semi-autonomous agents in a simulated world. In RTS game and other Role Playing Games (RPG) the user defines behaviors that must be applied by the agents in specific situations. For example, agents can go to a location, collect some elements and bring them back. To define routes for the agents, a common mechanism is the use of way points that will be followed by the agents as presented in Figure 2 for the game Age of Empire [42]. The player can specify waypoints directly on the radar view to guide one or a group of agents. The user must first select agents (usually with mouse or by using shortcuts on the keyboard to filter agents by types) and then specify the way points on the map. The agents will move on the map and avoid obstacles or restrictions to reach the end points.

**Figure 2: Defining waypoints for an agent in Age of Empire ®**

In other similar games such as Anno 1404 [43] or Starcraft [44], the user can edit existing routes by adding, modifying or deleting waypoints that constitute the route. Another automation mechanism available in Starcraft is the definition of a sequence of actions. The user can include several waypoints and specific actions directly related to the game such as wait at this position or follow other agents.

In most of the cases, players do not have much feedback on what goes wrong in the automation or the workflow that they designed, except a simple alert when the enemy attacks them. A possible explanation is that the game designers want to challenge the player and put

them in a situation of awareness/alert/stress as s/he plays. If those games were fully automated, they might be less interesting to play.

## 5.3 Defining autonomous agent behavior

In some RPG games such as the Final Fantasy series [45], the user can give a general automation behavior to some of the playable characters or the agents that are in the game. Those behaviors are generally associated with "Classes" that define the main characteristics of the behaviors. Often, the only feedback that the player is the change of the outfits for the agents which symbolizes the class and so the behavior (AI) of the characters.
In Final Fantasy XII [45], the player controls the main character only but s/he can literally program the teammates AI by customizing the actions and reactions of all the characters independently. This is done in the form of rules that can be activated / deactivated during the game. This end-user programming system (gambit system) uses visual blocks that consists of a priority, an action type to trigger or disable and a condition. Figure 3 illustrates such program with various conditions that define the behavior of an agent. When these automations are executing during the game, the user gets visual feedback. A line is drawn between the automated agent and the other agent involved in the action. The color of the line depends on the type of action and a progress bar shows when other actions can be performed by the agent.


**Figure 3: a) complete AI workflow for one character - b) feedback on the battlefield**

## 5.4 Programming robots

Specific robots for private use aim at replacing human action with automated one performed by a robot. For example, Farmbot [46] is a farm robot that take care of plants. To make the robot autonomous, the user must plan what will be growing using visual programming (Figure 4). Then the robot will be fully autonomous to take care of these specific plants for which it has predefined instructions. Farmbot also offers a graphical interface to monitor the garden.

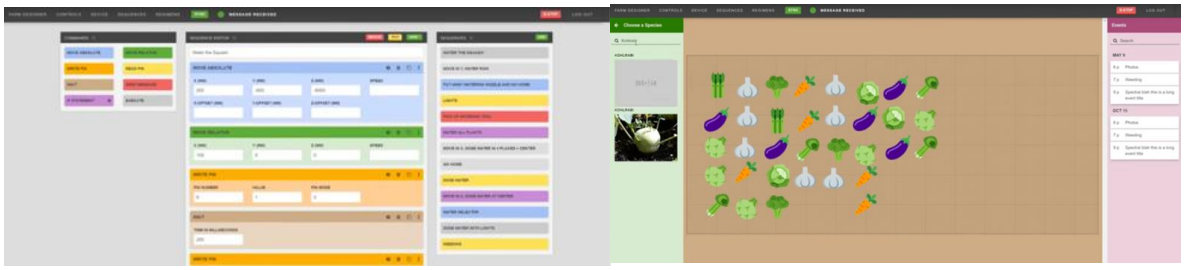


Figure 4 : Farmbot graphical user interface to specify the plants and to monitor the gardening.

Choregraphe [47] is a software program from Softbank robotics to program the movements and behaviors of the humanoid robots NAO and Pepper. Choregraphe uses direct manipulation of predefined blocks representing functions, posture, sound, flow control, tools and so on. The user can connect those blocks to create a program, and play it with a feedback on the program state, either on a 3D representation of the robot or directly if connected to the real robot (Figure 5). The user can also add some scripts in different language for further and more complex uses.

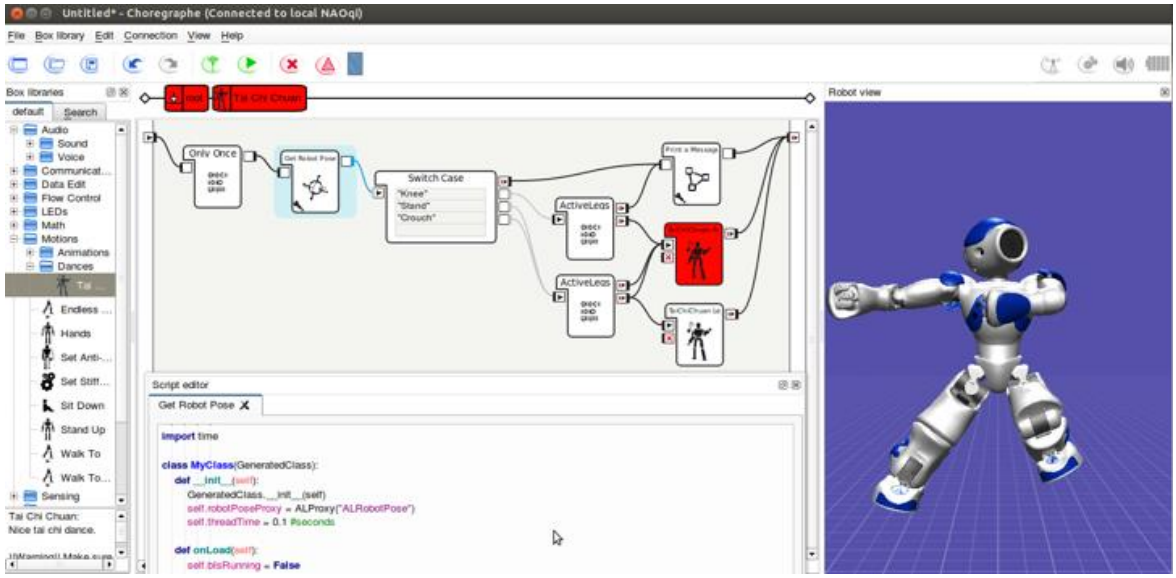



Figure 5: Visual and textual programming for the Nao robot with 3D preview of the result.

# 6 Music tools

Music composers are well served with technologies that help them create new sounds, input and produce musical scores as well as record and organize sound material into songs. Among these technologies, a great number of tools and techniques assist the composers in programming and executing temporal plans for real-time performance. Such tools and techniques can be relevant for the TaCo project as they cover end-user programming techniques, visual representations dedicated to temporal automation and performance oriented interfaces.

## 6.1 Temporal sequencing and automation

Music is a temporal phenomenon and as such composers need to use various tools, called sequencing tools, to arrange musical elements in time and specify the temporal evolutions of various parameters. In a musical context, automations are defined as temporal evolutions of parameters that affect sounds such as the evolution of the amplitude or the frequency of a filter. Duignan et al [7] identified five main categories of sequencers available to musicians: textual languages, sample and loop triggers, visual programming, linear sequencer. While classification illustrates the variety of existing approaches, both commercial products and research outcomes frequently blend these categories to offer a wider workflow to users.

### 6.1.1 Temporal sequencing with an absolute timeline

A sequencer lets composers and musicians organize in time various musical elements such as audio or midi that can be imported from files or recorded from inputs. In such paradigm, the direct manipulation of graphical objects representing musical objects on a timeline allows to specify their onset and duration. For example, Figure 29 (left), illustrates a timeline with several audio files organized in different tracks in the Ableton Live software[2]. In this software, users can use two different views: the arrangement view that presents the musical elements organized on a timeline and the session view that presents the individual clips by instruments similarly to a physical audio mixer.

[2] https://www.ableton.com/en/live/

**Figure 29: Ableton Live graphical user interface in Arrangement (left) and Session (right) mode. Colored clips represent audio or midi files that are organized on a timeline (left) or by order (right). Ableton ®**

**Defining automations:**

To specify the evolutions of specific parameters over time such as the amplitude of the sound or the value of a specific parameter, users can either record a performance or graphically input the automation curves.

To record an automation, users usually manipulate physical controllers embedding buttons, knobs and sliders while playing their musical sequence. This process generally involves a trial and error approach in which users play along to explore ideas and later edit their performances to obtain a better result.

For graphical input, most of the software feature simple drawing tools to draw automation curves or to set individual points and interpolate between them. These values are frequently overlaid over the waveforms or the midi files as presented in Figure 29 to facilitate their input.

**Monitoring the software status**

As signal computing is consuming CPUs, this can be an issue for projects using several audio processors in parallel or without sufficient computing power. When the CPU is overloaded, it audio artefacts and glitches appear. To help users monitor this aspect and avoid overwhelming the system, most of the software feature a CPU indicator. If the CPU is too high a "freeze" feature can be used to perform an offline rendering of parts of the project to save CPU. This offline rendering applies all the audio transformations to the original signal to create a static audio file on the disk.

**From composition to real time performance: taking control over a defined plan**

Once created, projects including several audio tracks with many automations are usually rendered as audio files on the disk. However, the projects can also be used during live performance and real-time modification of the content. For instance, a musician might want to increase a specific parameter during a song to accentuate a musical effect in response to the audience queries. In this case, the original content is a global plan that will be modified by the

performer during the performance, thus involving the user to take the control back from the planned automations. Figure 30 presents a simple scenario that illustrates the visualizations and interaction mechanisms in Ableton Live to support these kinds of actions by the performer. On the left, the automation is played according to the plan. On the right, the users changed the automated parameter via the graphical interface or an external controller while playing. This result in a gray version of the automation curve and a dashed line representing the value that is overriding the automated one. A button highlighted in orange allows to get back to the automation.


**Figure 30: Automated parameters during performance and manual control of an automated parameter in Ableton Live. Ableton ®**

### 6.1.2 Score following and textual descriptions of events

In automatic score following paradigm, a real-time listening machine reacts to musical events described in a score and interpreted by a human performer to launch predefined actions. Antescofo [7] is an example of a such a program used by composers and computer music designers to implement score following during performances. In Antescofo, a score is textual and has to be expressed in their own specific language. More recently a graphical user interface has been developed to let users input textual instructions along with a visual representation of the resulting score over a tempo-based timeline [6] as shown in Figure 31. The graphical representation lets users specify actions and curves to trigger when specific notes of the score are played during the performance. It also displays a playhead to monitor the current position in the score during the performance.

In order to address the potential errors during the performance from both the human performer and the listening machine, Cont et al. propose to defer the different possible error handling strategies to the user [8]. They define two strategies, tight and loose, that respectively corresponds to whether the system will necessarily trigger all actions or avoid some parts if events are missed. They also define two scopes, local and global, to define if there is a specific ordering to preserve or not. The user must define in the textual language using decorators the scope and the strategies that need to be applied.

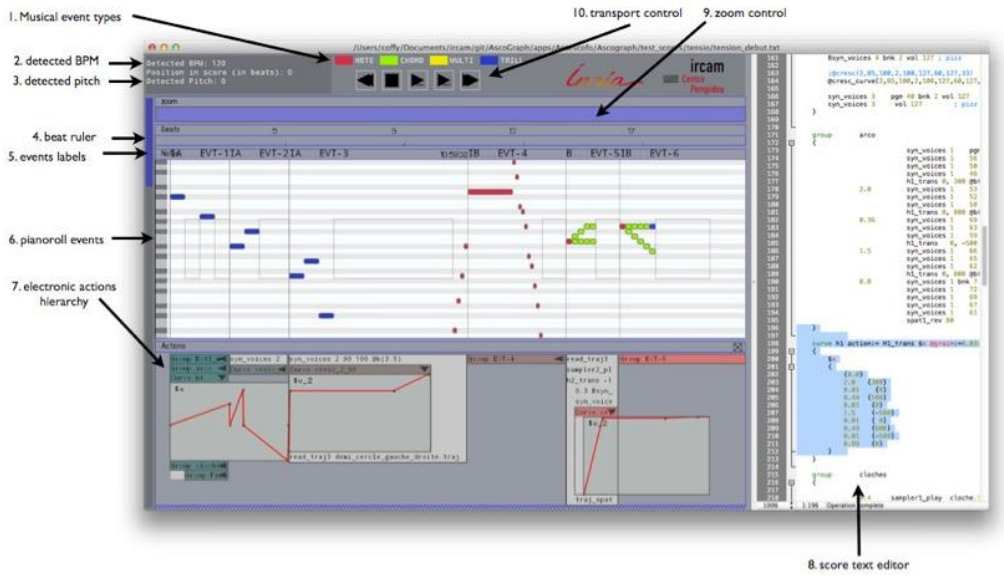



**Figure 31: Ascograph interface to write and monitor score following using Antescofo. The left panel shows a visual representation of notes and associated curves or events to trigger. The right panel shows the textual score description using the domain specific language. Antescof ®**

## 6.2 Visual programming

Visual programming plays an important role in computer-aided composition as it allows composers not familiar with conventional computing languages to implement their ideas with advanced graphical user interfaces. Computer aided composition requires composers to define computational processes and to interact with them through different musical data representations. Computer-aided composition environments, provides several musical objects such as scores, waveforms or lists of parameters as input and output to support both signal and symbolic operations over musical content.

### 6.2.1 Data flow environments

Programming environments such as Max and Pure Data [25] are based on data-flow programming. In such languages, functions or "objects" are linked or "patched" together in a graphical environment that models the flow of the control and audio. These two environments are frequently used for building real-time audio synthesis engines controlled with various input devices or features extracted from audio input. In real-time data flow programs, users get real-time audio visual feedback as they program. This makes it very convenient to quickly assess, both visually and with sound if the result matches the expectations. Figure 32 illustrates a visual program in Max/MSP and the debugging window that allows to inspect, at runtime, the value of the data flowing along the cords at specific watch points. Other software

environments such as Sensomusic Usine or Reaktor (see Figure 33) provide similar functionalities with different approaches and different set of built-in objects or modules.


**Figure 32: Max/MSP visual program (patch) containing several objects on the left. The debugging window on the right shows the signal flow at specific watchpoints. Cycling'74 ®**

**Figure 33: Two visual programming environments (left: Reaktor ®, right: Senseo Usine ®) using two related views: the visual programs and the resulting graphical user interfaces hiding the connections.**

**Programming and performance views:**

Both Max and similar tools feature two related visual representations: a visual program view blending interface elements with connections between modules and an interface only view for performance. The later facilitates the use of the program during the performance. It also let developers produce standard software tools for other users that won't be programming it. The main limitation of these dual view mechanisms is the difficulty to switch between modes when the spatial configuration is different in both views. For example, in Max, the same window hosts both visualizations so objects can be at very different locations on the screen when switching from one mode to another. Reaktor addresses this problem through two distinct views but then the user must switch between the two visualizations which can be problematic as the number of visual elements gets larger.

### 6.2.2 Demand-driven data-flow environments

Even if they are very popular tools for creating audio effects, synthesizers or more generally to create real-time instruments, these environments provide limited support for classical notation and time-based representations of musical scores. It is then difficult for musicians and composers to express possibly complex temporal relationships or to explicitly use time in their programs. A different approach of visual programming for composition with symbolic music representations was first introduced by PatchWork and extended by OpenMusic and PWGL [2]. These environments are based on a demand-driven data-flow paradigm, where objects recursively evaluate their inputs.

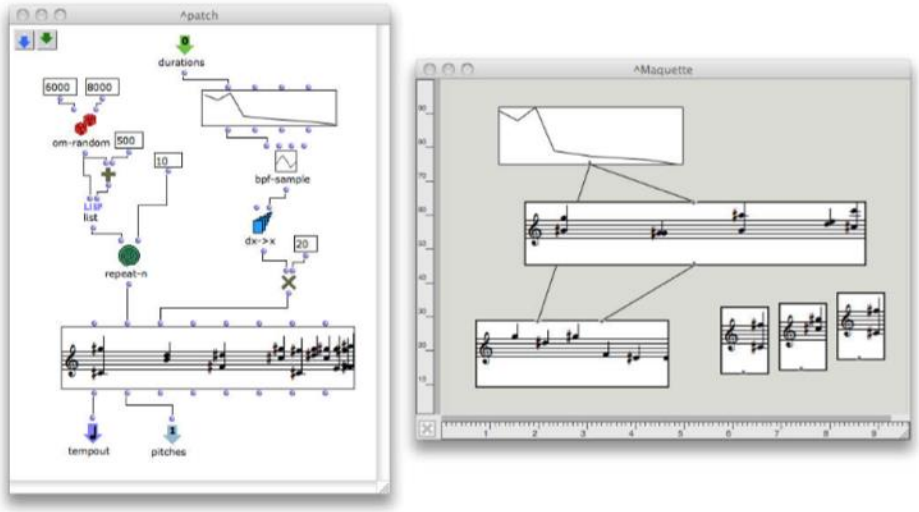



**Figure 34: Left: OpenMusic patch to compute a musical sequence. Right: A maquette that represents temporal objects.**

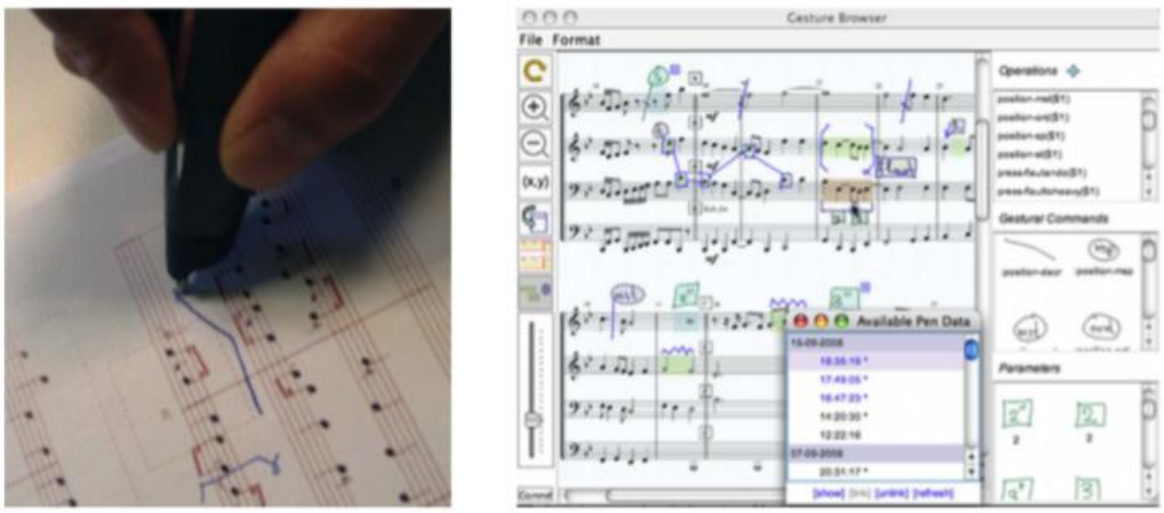
Figure 34 presents an example of patch and a “maquette” created with OpenMusic. In a maquette, objects evaluate their inputs to compute a result in a patch where their horizontal position determines their onset and duration on a timeline. This paradigm provides greater support than data-flow programming for composition as it is an "out-of-time" activity. In demand-driven data flow programs, composers do not have real-time feedback on the output of their program but they can evaluate only parts of their program and compare the outputs of several evaluations.

## 6.3 Creating vocabularies and mappings by example

In addition to visual programming approaches, other approaches support end-user definition of personal vocabularies or gestures through the use of gesture recognition algorithms or machine learning techniques.

### 6.3.1 Creating and using personal paper interfaces

Interaction with a pen device often involves the recognition of specific gestures. While this can be well adapted to predefined tasks, it does not suit well the workflow of music composers that frequently define their own set of representations on paper. Instead of proposing a predefined vocabulary, Musink [34] helps composers create and evolve their personal notations on paper over time (Figure 35). It provides an extensible gesture-based syntax, giving composers significant freedom to create their own composition languages and allows them to link their gestures with music programming software. However, the recognition occurs only once the pen strokes are uploaded to the computer, not when the user is writing.




(a) Annotating scores (b) Gesture browser

**Figure 35: Musink helps composers define their gesture vocabulary on paper.**

### 6.3.2 Creating mappings by example

In order to associate input devices or sensors streams to sound synthesizer, composers or musicians must create mappings between these two entities. Existing approaches relies on supervised machine learning techniques to let user create such mappings from example [13,14]. Wekinator [13] lets users associate examples of input with specific output configuration to build a model that can then be used with new data. For instance, a musician can associate specific postures of his hand in front of a camera with specific sounds. After the training phase, the model can output the estimated posture for every new image and thus trigger the corresponding sound. This approach accelerates the creation of complex interactive systems that would have otherwise required the implementation of complex heuristics to recognized different gestures. However, such a system is highly dependent to the training data and learning algorithm used so that it can hardly be generalized to different settings or to other users.

# 7 ATC Tools

ATM specialists already called out for more automation [36] [22] in order to mitigate the bottleneck that is the high number of tactical decisions to be taken by the ATCOs and implemented by aircraft crew by using a weak (voice G/A communications) link (suffering, among other limitations, misunderstandings, and latencies). They recognize that the design should be necessarily iterative, and supported by Modeling and Simulation (M&S) with humans in the loop, especially prior to product delivery. They also insist on the fact that effectiveness of M&S means make automation based on human-centered design possible and efficient.
Scientific challenges dealing with specific very relevant issues associated to automation processes include: resilience and control system degradation; the adequateness and correctness of the human role in the control system, in particular the ability to ensure human motivation, trust, and dependence on automation, and the ability to maintain situational awareness [22]. Indeed, humans trust and understanding of automation is key to safety, as recent accidents seem to demonstrate [10].
Past work in ATM studied the inclusion of some forms of automation (e.g. global traffic management [35], or Single Pilot Operation [21]). We focused on recent works pertaining to ground operations only. The authors of these studies draw some conclusions worth mentioning here.

## 7.1 SESAR project 6.7.2 surface routing and datalink-taxi

As part of SESAR Work Package 6 “Airport Operations”, project 6.7.2 addresses operational issues such as low visibility, icing conditions, radio occupancy, stop and go, route deviations and taxi time estimate (Figure 36). The primary function is to provide the ATC officer with a planned route that can be edited. This route can then be sent by Datalink to the pilot.

The following positive results from the validation exercises are:

- the tool was able to compute relevant routes
- the route planning function has been accepted by the users

Less positive results include:

- HMI improvements are needed.
- operations on platforms such as Roissy-CDG are too complex to use Datalink only, hence mixed use of electronic and radio communication must be considered.

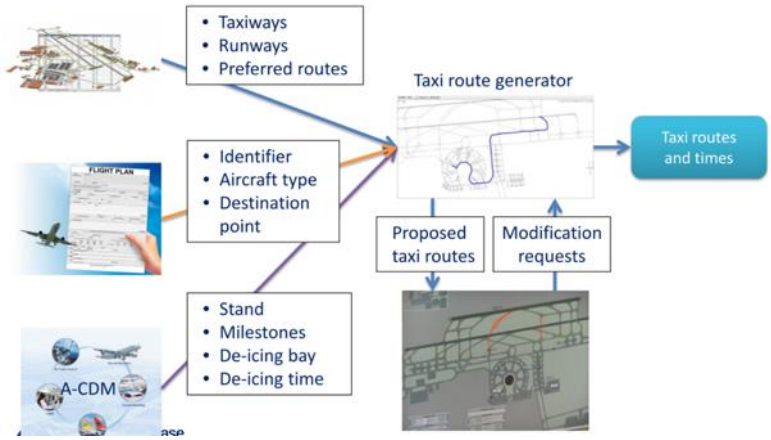



Figure 36: SESAR project 6.7.2

Other results at Madrid gave the following positive results:

- Surface Planning and Routing functions were found to be beneficial.
- Data link was considered as useful in the context of non-time-critical communications and resulted in reduced voice frequency channel occupancy.

However:

- Routing and data link functions decreased situational awareness and increased the controllers and apron managers' workload.
- Human Machine Interface (HMI) was found to be too time-consuming in the manual mode.


Figure 37: SESAR 6.7.2

These results highlight the importance of the HMI design and its impact on the performances and the satisfaction of the users. Since the routing function allows for a more predictable taxi time, new functionalities are possible, such as virtual stop bars that complement physical stop bars with digital ones at selected airport location, or lights on airfield ground activated along the cleared route (Figure 37). To the best of our knowledge, none of these features are dynamically programmable by the ATCO.

## 7.2 EMMA2

EMMA2 developed a generic operational concept for higher-level Advanced-Surface Movement Guidance and Control Systems (A-SMGCS) services, implemented the essential parts of the concept at three European airports, and performed comprehensive trials in simulation and in the field to evaluate the new services [16]. The higher-level A-SMGCS services concept were developed, integrated and tested on more than 15 different test platforms by the EMMA2 consortium. One of the outputs of the project is a set of recommendation on different topics: Operations of A-SMGCS Services (Air Traffic Controllers, Flight Crews and Service to Vehicle Drivers), Technical and Operational Requirements, Validation Process, Implementation Issues.
The EMMA project has provided clear evidence that the evolution of A-SMGCS toward more advanced services shall be accompanied with studies to improve and optimize the Human Machine Interface of new services that will be proposed to the A-SMGCS users, i.e. ATCOs, pilots and vehicle drivers. Especially when delegating tasks or giving clearances, ATCO must be confident that the level of information is consistent between actors (usage of same maps) and that communication is clear (Datalink failures). Concerning the monitoring system, an alert associated with a detected conflict should be provided with an adequate time and brought to the attention of the ATCO (alarm coding). An alert associated with a predicted conflict (information coding) should also be provided and differentiated.
Some recommendations dealing with DMAN implementation might be interesting for the TaCo project. They concern the potential interactions between the user and the automated system:

- Take into account that a considerable amount of time is needed to adapt the tool to the particularities of the concerned airport and its local procedures (e.g. runway configuration, intersection take off- and de-icing procedures, helicopter operations or pushback constraints).
- Keep the DMAN always informed about the current operational status of each flight (e.g. pushing back or taxiing) by information provided by the surveillance and clearance inputs of the ATCO.
- Use color-coded “Recommended Time Until next Clearance” (RTUC) information to support the ATCO to implement the DMAN-planned outbound traffic.

## 7.3 Modern Taxiing (MoTa)

Project Modern Taxiing (MoTa) [5] studies the impact of future taxiing technologies such as Datalink and autonomous taxiing tugs on airport taxiing operations and air traffic controller workload. The tool consists of an integrated ground control interface featuring the latest progress in modern taxiing methods and multi-agent algorithms for enhanced ground automation while still supporting current and conventional ground control procedures during the transition period. In addition to the new integrated ground control interface (Figure 38), autonomous taxiing tugs (inspired by the TaxiBot system) were simulated. The concept is to use the tugs to continue towing the aircraft after pushback, along the taxiways until the runway holding point, thus saving fuel since aircraft engines would be lit later in the taxiing sequence. In that manner, a departure aircraft would be handled as usual by ground control, but when the tug is detached from the aircraft after depositing it at the runway, the empty tug would return to the parking areas via the same taxiways as the rest of the traffic.

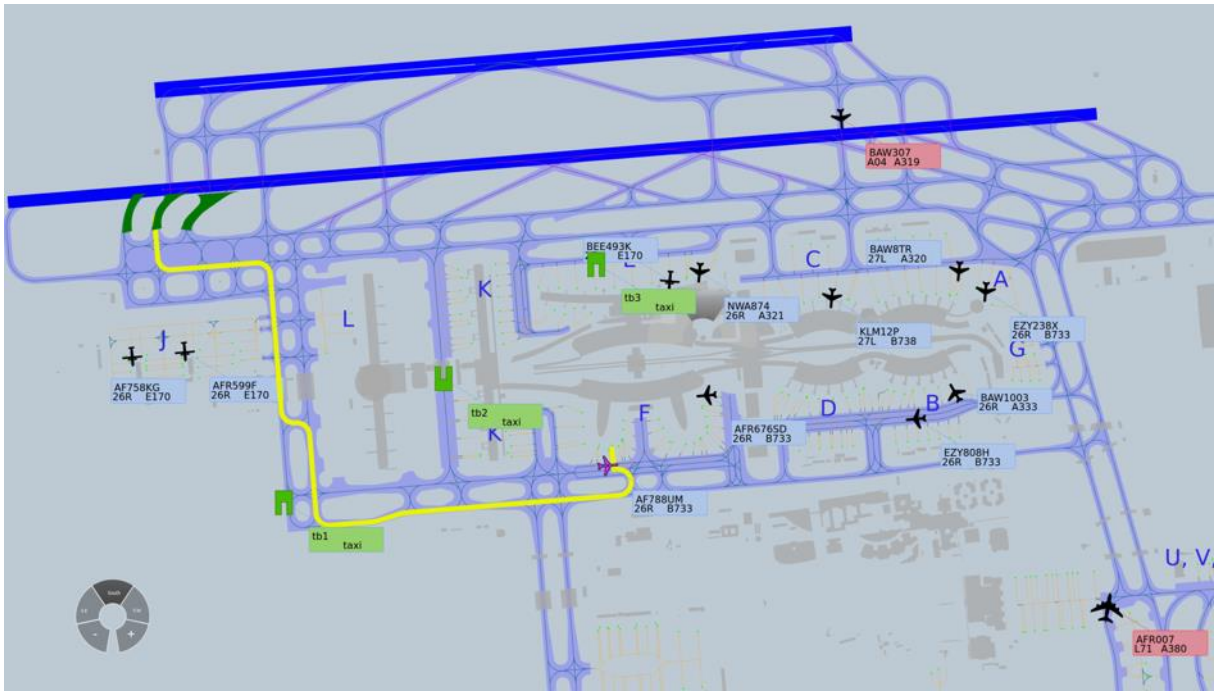



**Figure 38: The MoTa HMI**

Route solutions, determined using a multi-agent system (MAS), are presented to the ATCO. As the suggestions are based on standard paths according to aircraft type, destination, and airline, the ATCO can gain time by simply validating the suggestion instead of creating a solution. ATCOs may modify the suggested path as necessary. The level of service that the MoTa platform is able to provide is notably due to this interaction between user and machine. Thus, the system can monitor the situation at various levels. For example, the system can warn the ATCO when a pilot does not follow the instructed route. Additionally, the system

can detect when an aircraft has reached the end of its clearance and advise the ATCO to transfer this aircraft so that the trajectory is continuous.
The MoTa Project validation campaign consisted of evaluating the platform at different stages under a range of work scenario difficulties in order to understand the impact of such technologies on ATCO performance and workload. Three human-in-the-loop experiments were conducted during the course of almost two years, each centered on a specific technology level, with both scenarios simulated in each experiment. All experiments were conducted in an ATC simulator.
The initial results of this study show three main results. First, automated taxiing technology assistance (e.g. Datalink, path suggestion, autonomous tugs) can help improve ATCO taxiing performance. Second, more technology may increase the overall workload. Third, the MoTa system is promising but is not currently mature enough to replace current systems.
The tendencies in percentage of aircraft correctly treated show that the addition of the tactile interface with Datalink and path suggestion improved overall throughput. However, the inclusion of the tugs and increase in datalink usage reduces these gains, thus implying that a) tugs do not provide a significant performance advantage; b) too much Datalink may contribute to the workload; or c) a combination of both may overall throughput.
While participant comments suggest that the tugs are the only culprit, the experimental design does not allow for isolation of this effect. Indeed, while the use of both technologies seems to improve performance, there is not enough statistical power to state whether the improvement is attributable to the effect. In general, the MoTa platform seems to be the most effective in a hard scenario, with performance fairly regular across all technology levels in a medium scenario. Participants have mixed feelings with regards to the technology, with about half reporting ease and noting advantages to the technology and the other half expressing discontent with its functionality or usability.

## 7.4 MAMMI: Multi Actors Man-Machine Interface

Collaboration is key to safety and efficiency in Air Traffic Control. For example, ATCOs monitor each other’s actions. When a flight must turn to follow the planned route, or when the controller has devised an avoidance strategy, the controller needs to give orders to the pilot at the right moment. Hence, part of the activity is devoted to remembering which actions to do at present, or in the near future. Furthermore, resolution of problems depends on the actual execution of orders by the pilots. Hence, controllers must monitor that pilots actually follow orders as given. The planning controller also checks and monitors the actions of the tactical controller and possibly corrects them in high workload situations.
MAMMI (Multi Actors Man-Machine Interface) is an interactive system designed according to a set of requirements to support collaboration: support mutual awareness, communication and coordination, dynamic task allocation and simultaneous use with more than two people. To fulfill these requirements, MAMMI uses a multi-user tabletop surface, appropriate feedthrough, and reified and partially-accomplishable actions. The horizontal multi-touch screen displays an environment that includes a number of interactive graphical objects.

Even if MAMMI's principal topic is not Automation, one can consider some of the interactions as ways to delegate important tasks to another entity, be it human or machinery. For example,
in order to help remember future actions, controllers can place Post-its on a timeline. The timeline is a horizontal strip that lies at the top of the screen. The X dimension of the timeline depicts the time: current time is at the center, and the future extends outwards in both directions, from the center to the edges of the timeline. A ruler that depicts the time according to the X position helps users to position Post-its. Once attached to the timeline, Post-its move automatically towards the center at a pace that follows real time (see Figure 37: MAMMI and its timeline). As Post-its reach the center, controllers are encouraged to accomplish the associated action, before the Post-its disappear. The double-sided aspect of the timeline enables users to allocate responsibility: each user is responsible for the Post-its that lie on his or her side. Controllers can rearrange the strips, either to specify a different action time or to implicitly redistribute responsibility by moving a Post-it from one side to the other. As Post-its move to the center, they become easier to take from the other controller.

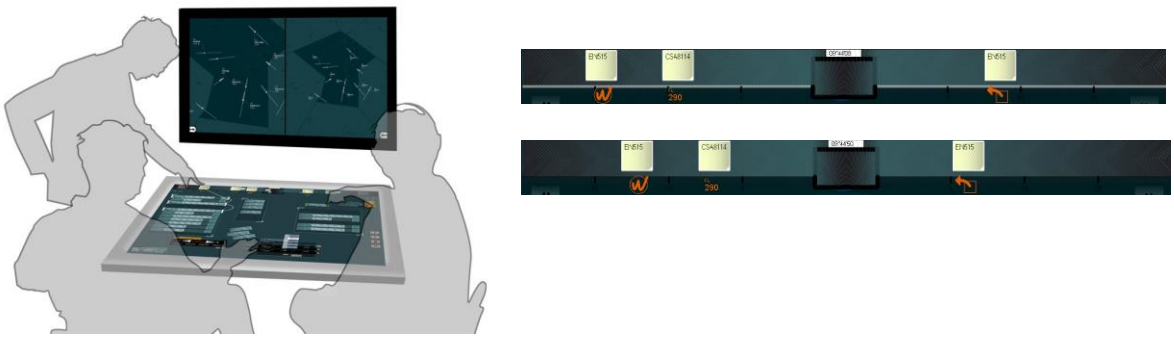


Figure 37: MAMMI and its timeline

Placing post-its on a timeline to remember things to do can be considered as programming an automation to notify oneself about an action to perform. This imposes the same requirements as discussed before e.g. to make visible the status of the notification (see it moving on the timeline, assessing the distance left from the middle of it and hence assessing the remaining time). It might thus be interesting to reuse some of the design guidelines from MAMMI:

- Reify actions into objects. Since objects lie on the table, their manipulation may enable accountability [29]; furthermore, they can be passed around and allow for task reallocation.
- Enable partial accomplishment of actions. An action can be separately prepared, checked and accomplished, possibly by different users, thus offering seamless workload allocation.

- Provide as much feedthrough as possible. Since activities must be accountable, it is important that appropriate feedback provide an opportunity for teammates to observe one another's actions.

# 8 Summary and research directions

The state of the art developed in the previous sections shows that End-User Programming is a widespread discipline, even in professional settings. The next step for the TaCo project will be to conduct workshops to design the relevant EUP features that will enable ATCOs to specify and program automation. In order to succeed, we must take into account the lessons learned by previous experiences, and design accordingly. Based on this state of the art, this sections formulates a set of research directions (RD) that we will use during the workshop.

## 8.1 Type of support

End-user development (EUD) is about taking control—not only of personalizing computer applications (end-user computing) and writing programs, but of designing new computer-based applications without ever seeing the underlying program code. [31]
As we discussed before, it is likely that the end-users will have a range of programming abilities. Still, it is difficult to envision a system where the user will never "see the underlying code". Code is not equal to "programming language text", code may also be visual. And even if it's visual, it still represents "behavior", and the various conceptual ways of thinking about behavior (data-flow, instruction flow, state machine) should be understood. What makes a programmer is not only knowing to read the code, but also knowing the underlying concepts.

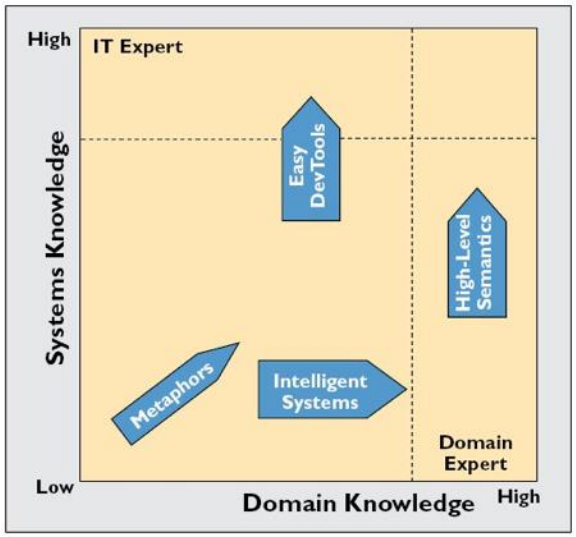



**Figure 39: Expertise tension**

In fact, as acknowledged by [3], an expertise tension exists in a two-dimensional continuum of job-related domain knowledge and system-related development knowledge (Figure 39). TaCo's users might be *Lacking systems knowledge*. Thus we should strive to enable domain experts to modify or extend software without having a deep understanding of a computer system or coding skills,. Expertise tension would then be in the direction of system knowledge. This can be mitigated with two approaches [3]:

- *Easy development tools* provide end-user design times that come with a simplified function set, higher-level primitives, direct manipulation, WYSIWYG, or logical user interfaces that encapsulate system-oriented low-level details [3].
- *High-level semantic building blocks* enable users to create new things by composing familiar domain specific entities. Such applications usually do not generalize across different domains but are specialized solutions for a well-defined task domain [3].

*Easy development tools* is discussed in the next section. As for *High-level semantic building blocks*, after all, the whole air traffic system is already an automation-based activity from the ATCO's point of view. ATCOs delegate some actions to the pilots, and they must monitor whether the pilots perform them. In some ways, one can consider pilots as equivalent to an automatic agent that will succeed or fail, which requires either monitoring or notification. One of the tasks that we will have to do is to conduct research on the similarities between the current ATC activity and programming concepts. This will help us design the high level semantic building blocks as discussed above. This will require some training, but we are confident that ATCOs will be able to perform some programming, as already discussed in the introduction.

*Research Direction (RD) 1: define programming language constructs relevant to ATCOs*

Nonetheless, designing a programming language is not a small task. Designing a usable one is even more difficult. Recent work on the topic suggests that one should run experiments to get

some evidence on the usability of the representation of programming language (not so on the underlying conceptual model) [30]. This might be too ambitious considering the scope of the TaCo project. However, as specialists in Programming and User Interaction Designers, we will take special care in studying the usability of the tools we design, be they concrete or conceptual.

*RD2: make relevant programming language constructs usable by ATCOs*

End-User Programming does not occur into isolation, and requires some tasks borrowed from End-User Development or even End-User Software Engineering. One of the questions we will have to answer is to find out which of the subtasks from EUP, EUD or EUSE is suitable for TaCo, and at which time (during offline, in advance automation design by specialists, or in real-time during the Air Traffic Control Activity).

*RD3: select relevant dimensions of EUP, EUD and EUSE for the task at hand*

## 8.2 Type of IDE

As seen in section 3 and 4 IDEs are an essential tool to support the programming activity. Much of the IDEs allow the programmer to specify behavior by building up the program. Another way to specify behavior is to use a “programming by demonstration” or “programming by example" style [11]. Among the tools we referenced, Automator allows this style of programming. There were some attempts to provide support for this kind of programming in academia [12][37], but also for web automation [18]. Though they did not give rise to successful products in the industry, some of the concepts might be borrowed for the TaCo project. Mixed with the set of semantic building blocks above, they could be valuable as a way to make automation imitate what an ATCO would do manually in some of the situations.

However, solving PBE usual problems will be beyond the scope of the TaCo project. One of them is the automation of programming itself i.e. inferring programming constructs from the user’s actions/demonstration. We should be careful not to solve the problem raised from programming automation instead of ATC automation, and find the right balance between investigating interesting PBE concepts and designing for ATC automation.

*RD4: borrow relevant concepts from Programming by Example*

Another dimension is the type of “liveliness” we should strive for. In order to be close to the activity and to the final result, we might turn the current interface into a live environment that could be programmed and updated in real-time without starting over. This might be challenging to develop, but mandatory if we plan to enable the ATCO to reprogram the behavior during actual control. Another related question would be to investigate the use of an

external editor if necessary, or rather how much programming interface we can provide tools inside the existing MoTa interface without hindering its usability, and what programming interface to distribute on external windows or screens.

*RD5: foster and develop an interactive, live programming environment based on MoTa*

This live programming [32][20][15] aspect may also be linked to recent considerations on what to display and to interact with [4]. If we are to facilitate the predictive assessment of the results, we might have to use some of the representations and interactions as described in [4]. Granted, those were invented/called out to facilitate computer programming education. However, much of these considerations might be also be valuable for production code.

As an already highly graphical interface, TaCo might leverage the GUI to help express the conditions that triggers events and automation [9]. One question is how far we can go into designing programming construct using the graphics already available in the interface, or 'invisible' graphics that would well suit above the visible interface.

Another question is the type of code representation we need. Professional IDEs tend to use more and more graphical means of representing code. However, there are multiple types of representation, and they may be adapted to particular tasks [9]. IDEs already provide multiple views of the same programs: data-flow and state-machines in Vaps XT, route and list of actions ala Anno 1404. One particular area of research would be to study the use of multiple representations and how a programmer can pass from one to the other. Representations might range from purely graphics ones to textual ones, as the distinction between the two might not entirely pertain to usability [9].

*RD6: leverage the existing graphical environment*

Finally, concepts from music and videogames are applicable to the project. In particular, concerns on monitoring, notifying or taking over is particular important during live concert or live playing in order to perform well. Contrarily to videogames where difficulty may be "by-design" to offer challenges for the player, TaCo will explore a better use of well-designed notification to warn about possible automation difficulties. Beyond notification, which is often too late, a challenge for TaCo will be to offer better representation of the status of automation and especially its ability to handle the current traffic. This may lead to the use of harbinger (or precursor) cues to inform the user the 'load' of the automation so that s/he can predict her/his future load in an optional take over. Similar to music tools, the interface should foster interactive tools to take control even if the system did not forecast any trouble so as to let the user unilaterally decide to manually control, in case s/he thinks that the automation will fail eventually.

*RD7: foster predictability with harbinger cues*

The research directions are summarized in Table 1.

| | |
|---|---|
| RD1 | define programming language constructs relevant to ATCOs |
| RD2 | make relevant programming language constructs usable by ATCOs |
| RD3 | select relevant dimensions of EUP, EUD and EUSE for the task at hand |
| RD4 | borrow relevant concepts from Programming by Example |
| RD5 | foster and develop an interactive, live programming environment based on MoTa |
| RD6 | leverage the existing graphical environment |
| RD7 | foster predictability with harbinger cues |

**Table 1 : Research directions**

# 9 Bibliography

1. Hal Abelson, Nat Goodman, and Lee Rudolph. 1974. LOGO Manual. Retrieved December 7, 2016 from http://dspace.mit.edu/handle/1721.1/6226
2. Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. 1999. Computer-Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal* 23, 3: 59–72. https://doi.org/10.1162/014892699559896
3. Joerg Beringer. 2004. Reducing expertise tension. *Communications of the ACM* 47, 9: 39. https://doi.org/10.1145/1015864.1015885
4. Victor Bret. Bret Victor, Learnable programming. *worrydream*. Retrieved July 19, 2016 from http://worrydream.com/#!/LearnableProgramming
5. Zarrin Chua, Mathieu Couzy, Mickael Causse, and François Lancelot. 2016. Initial Assessment of the Impact of Modern Taxiing Techniques on Airport Ground Control. Retrieved December 20, 2016 from http://oatao.univ-toulouse.fr/16192/
6. Thomas Coffy, Jean-Louis Giavitto, and Arshia Cont. 2014. Ascograph: A user interface for sequencing and score following for interactive music. In *ICMC 2014-40th International Computer Music Conference*. Retrieved December 14, 2016 from https://hal.inria.fr/hal-01024865/
7. Arshia Cont. 2008. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *International Computer Music Conference (ICMC)*, 33–40. Retrieved December 14, 2016 from https://hal.archives-ouvertes.fr/hal-00694803/
8. Arshia Cont, José Echeveste, Jean-Louis Giavitto, and Florent Jacquemard. 2012. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. Retrieved November 10, 2016 from https://hal.inria.fr/hal-00718854/document
9. Stéphane Conversy. 2014. Unifying Textual and Visual: A Theoretical Account of the Visual Perception of Programming Languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Onward! 2014), 201–212. https://doi.org/10.1145/2661136.2661138
10. Stéphane Conversy, Stéphane Chatty, Hélène Gaspard-Boulinc, and Jean-Luc Vinot. 2014. The accident of flight AF447 Rio-Paris: a case study for HCI research. 60–69. https://doi.org/10.1145/2670444.2670459
11. Allen Cypher. 1991. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '91), 33–39. https://doi.org/10.1145/108844.108850
12. Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky (eds.). 1993. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, USA.

13. Rebecca Anne Fiebrink and Perry R. Cook. 2011. Real-time human interaction with supervised learning algorithms for music composition and performance. Princeton University. Retrieved November 24, 2016 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.1409&rep=rep1&type=pdf
14. Rebecca Fiebrink, Perry R. Cook, and Dan Trueman. 2011. Human model evaluation in interactive supervised learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 147–156. Retrieved November 24, 2016 from http://dl.acm.org/citation.cfm?id=1978965
15. Christopher Michael Hancock. 2003. Real-time programming and the big ideas of computational literacy. Massachusetts Institute of Technology. Retrieved December 18, 2016 from http://dspace.mit.edu/handle/1721.1/61549
16. Jörn Jakobi. 2009. EMMA2 Recommendations Report.
17. Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* 43, 3: 21:1–21:44. https://doi.org/10.1145/1922649.1922658
18. Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08), 1719–1728. https://doi.org/10.1145/1357054.1357323
19. Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-User Development: An Emerging Paradigm. In *End User Development*, Henry Lieberman, Fabio Paternò and Volker Wulf (eds.). Springer Netherlands, 1–8. https://doi.org/10.1007/1-4020-5386-X_1
20. Sean McDirmid. 2013. Usable Live Programming. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Onward! 2013), 53–62. https://doi.org/10.1145/2509578.2509585
21. Christopher A. Miller. 2014. Delegation and Intent Expression for Human-automation Interaction: Thoughts for Single Pilot Operations. In *Proceedings of the International Conference on Human-Computer Interaction in Aerospace* (HCI-Aero '14), 6:1–6:10. https://doi.org/10.1145/2669592.2669649
22. Francisco Javier Sáez Nieto. 2015. The long journey toward a higher level of automation in ATM as safety critical, sociotechnical and multi-Agent system. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*. https://doi.org/10.1177/0954410015596763
23. Donald A. Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.
24. Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans* 30, 3: 286–297.

25. M Puckette. 1996. Pure Data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer \ldots*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3903&rep=rep1&type=pdf
26. J. Rasmussen. 1983. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, 3: 257–266. https://doi.org/10.1109/TSMC.1983.6313160
27. Lucas Save and Beatrice Feuerberg. Designing Human-Automation Interaction: a new level of Automation Taxonomy. In *Human Factors: a view from an integrative perspective*.
28. Thomas B. Sheridan and William L. Verplank. 1978. *Human and Computer Control of Undersea Teleoperators*.
29. Stéphane Sire, Stéphane Chatty, Hélène Gaspard-Boulinc, and François-Régis Colin. 1999. How Can Groupware Preserve our Coordination Skills? Designing for Direct Collaboration. In *Human-Computer Interaction INTERACT '99: IFIP TC13 International Conference on Human-Computer Interaction, Edinburgh, UK, 30th August-3rd September 1999*, 304–312.
30. Andreas Stefik and Stefan Hanenberg. 2014. The Programming Language Wars: Questions and Responsibilities for the Programming Language Community. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Onward! 2014), 283–299. https://doi.org/10.1145/2661136.2661156
31. Alistair Sutcliffe and Nikolay Mehandjiev. 2004. Introduction. *Communications of the ACM* 47, 9: 31. https://doi.org/10.1145/1015864.1015883
32. Steven L. Tanimoto. 1990. VIVA: A Visual Language for Image Processing. *J. Vis. Lang. Comput.* 1, 2: 127–139. https://doi.org/10.1016/S1045-926X(05)80012-6
33. TMP. Avionics Display Development Software - VAPS XT™ | Presagis. Retrieved November 25, 2016 from http://www.presagis.com/products_services/products/embedded-graphics/hmi_modeling_and_display_graphics/vaps_xt/
34. Theophanis Tsandilas, Catherine Letondal, and Wendy E. Mackay. 2009. Musink: Composing Music Through Augmented Drawing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '09), 819–828. https://doi.org/10.1145/1518701.1518827
35. Kagan Tumer and Adrian Agogino. 2007. Distributed Agent-based Air Traffic Flow Management. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS '07), 255:1–255:8. https://doi.org/10.1145/1329125.1329434
36. christopher wickens, anne mavor, and james mcgee. 1997. *Flight to the Future: Human Factors in Air Traffic Control*. National Research Council. Retrieved December 19, 2016 from https://www.nap.edu/catalog/5493/flight-to-the-future-human-factors-in-air-traffic-control
37. 2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
38. 2012. SCADE Display®. *Esterel Technologies*. Retrieved November 25, 2016 from http://www.esterel-technologies.com/products/scade-display/
39. Unreal Engine Technology | Home. Retrieved November 16, 2016 from

https://www.unrealengine.com/
40. Unity - Game Engine. *Unity*. Retrieved November 16, 2016 from https://unity3d.com
41. NaturalMotion - Home Page. Retrieved November 16, 2016 from http://www.naturalmotion.com/
42. Age of Empires | Play The Greatest Stories in History. Retrieved November 16, 2016 from https://www.ageofempires.com/
43. Page Principale | ANNO Portal | Ubisoft. Retrieved November 16, 2016 from http://anno.fr.ubi.com/pc/
44. StarCraft II. *StarCraft II*. Retrieved December 14, 2016 from http://eu.battle.net/sc2/fr/
45. Accueil | SQUARE ENIX. Retrieved November 16, 2016 from http://eu.square-enix.com/fr
46. FarmBot | Open-Source CNC Farming. Retrieved November 15, 2016 from https://farmbot.io/
47. SoftBank Robotics. *SoftBank Robotics*. Retrieved November 17, 2016 from https://www.ald.softbankrobotics.com/en
48. Integrated Development Environment. *en.wikipedia.org*. Retrieved from https://en.wikipedia.org/wiki/Integrated_development_environment
49. Little Wizard's Home Page. Retrieved November 15, 2016 from http://littlewizard.sourceforge.net/tutorial.html
50. TurtleArt. Retrieved July 26, 2016 from http://turtleart.org/index.html
51. squeakland : resources. Retrieved July 26, 2016 from http://www.squeakland.org/resources/
52. Scratch - Imagine, Program, Share. Retrieved July 21, 2016 from https://scratch.mit.edu/
53. Explore MIT App Inventor. Retrieved November 17, 2016 from http://appinventor.mit.edu/explore/ai2/beginner-videos.html
54. Beetle Blocks - Visual code for 3D design. Retrieved July 21, 2016 from http://beetleblocks.com/
55. Jeux Blocky. Retrieved November 15, 2016 from https://blockly-games.appspot.com/
56. Coding for Kids | Tynker. Retrieved November 15, 2016 from https://www.tynker.com/
57. Apprendre à programmer - Mindstorms LEGO.com. Retrieved November 15, 2016 from https://www.lego.com/fr-fr/mindstorms/learn-to-program
58. NXT – LEGO Engineering. Retrieved November 25, 2016 from http://www.legoengineering.com/category/support/nxt/
59. Logiciel de conception de systèmes LabVIEW - National Instruments. Retrieved November 25, 2016 from http://www.ni.com/labview/f/
60. Kodu | Home. Retrieved November 28, 2016 from http://www.kodugamelab.com/
61. Agentcubes. Retrieved November 21, 2016 from http://www.agentsheets.com/agentcubes/index.html

62. Alice.org. Retrieved December 8, 2016 from http://www.alice.org/index.php
63. Wonder Workshop | Home of Dash and Dot, robots that help kids learn to code. *Wonder Workshop*. Retrieved November 15, 2016 from https://www.makewonder.com/apps
64. Wonder - the Newest Coding App For Dash & Dot from Wonder Workshop - YouTube. Retrieved November 17, 2016 from https://www.youtube.com/watch?v=9K2vv48iwHk
65. OS X Automation: Automator. Retrieved November 21, 2016 from https://macosxautomation.com/automator/foo.html
66. NoFlo | Flow-Based Programming for JavaScript | NoFlo. Retrieved December 7, 2016 from http://noflojs.org/
67. IFTTT. Retrieved November 21, 2016 from https://ifttt.com/discover
68. General principles for triggers and actions / Triggers and actions / Knowledge Base - IntuiFace Support. Retrieved November 21, 2016 from http://support.intuilab.com/kb/triggers-and-actions/general-principles-for-triggers-and-actions

# Annex A

This annex contains additional screenshots of the tools and systems mentioned in this document. These visual resources will be used to support the design of new visual tools for end-user programming in the TaCo project.

**Unreal Engine ®**

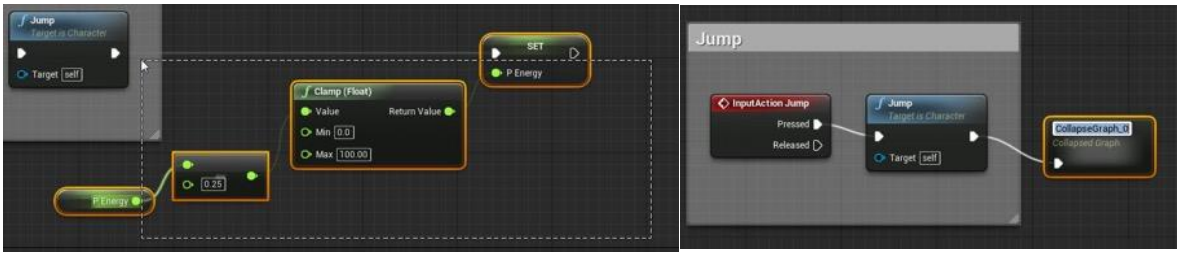



**Figure 40: Selecting several blueprint or boxes and transform it into one box**

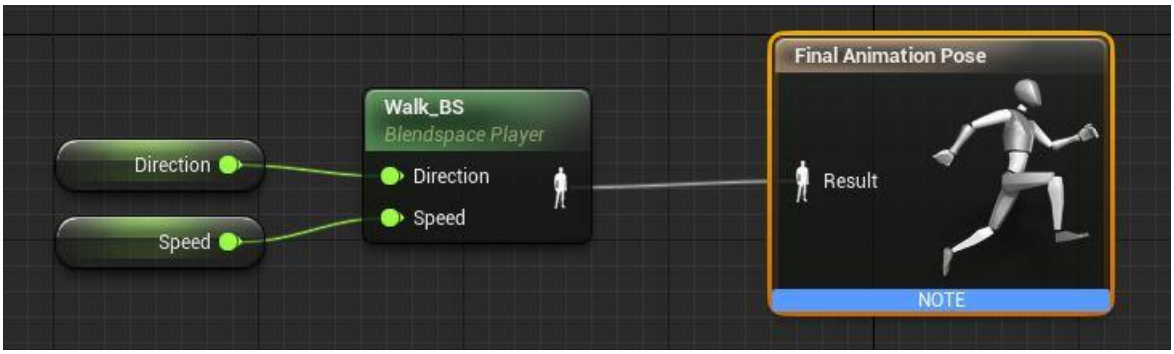



**Figure 41: Adding a predefined animation into Blueprint ®.**

## Euphoria ®

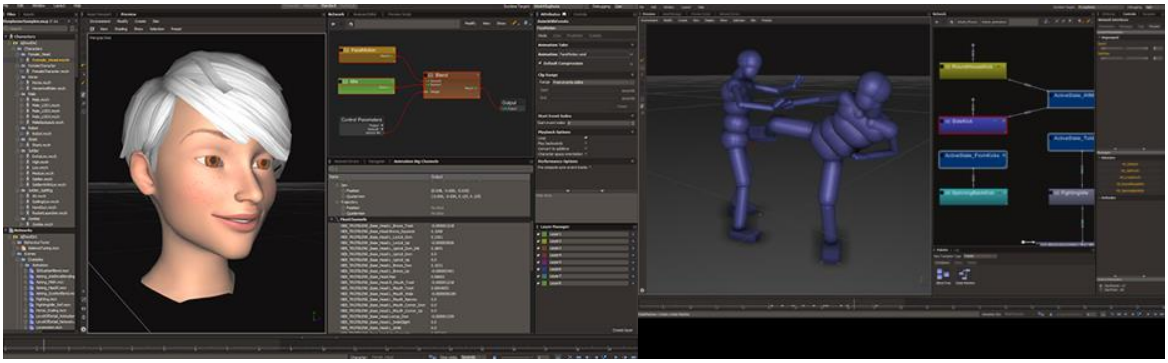


Figure 42: Facial and body animation using state machine.

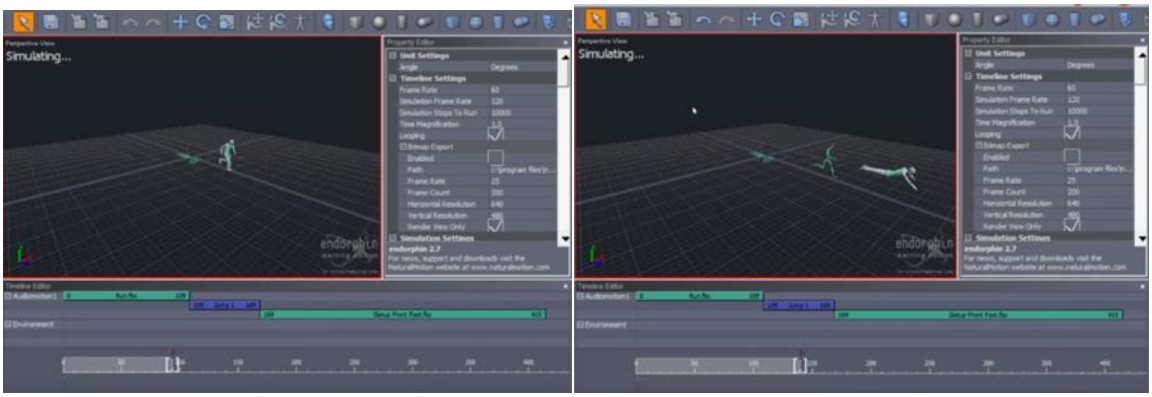



Figure 43: Blending between a running and a diving animation.

## Unity 5 ®

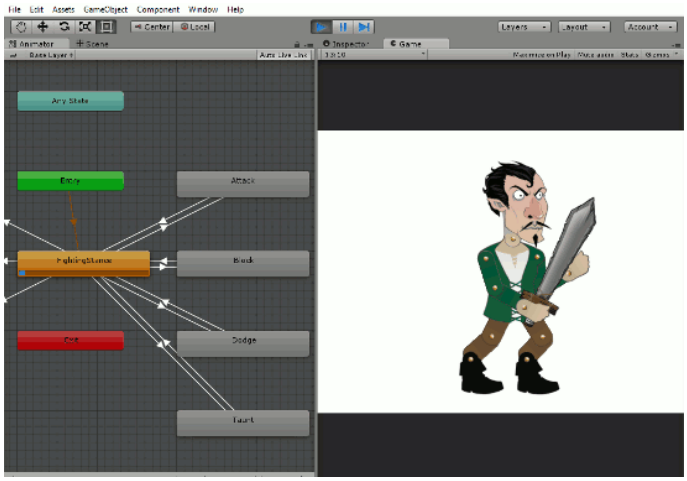



Figure 44: Example of state machine blending animation used in Unity ®

## Anno 1404 ®

Figure 45: Creating a trade routes

## StarCraft 2 ®

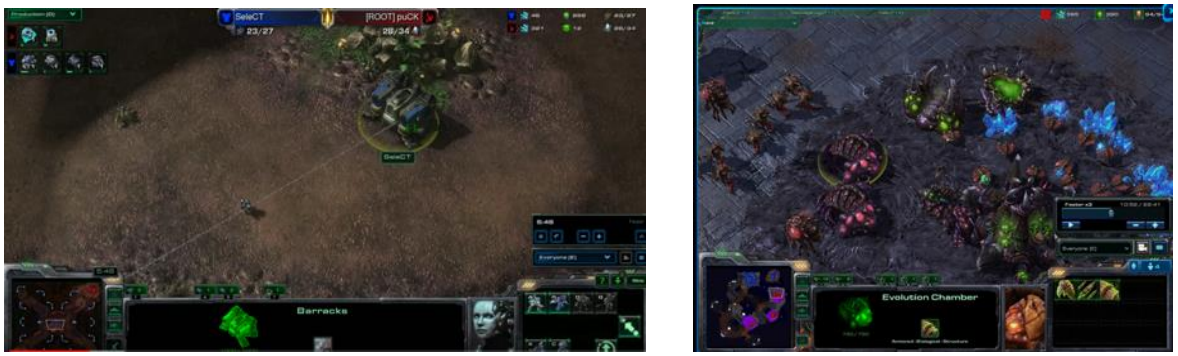



Figure 46: creating a sequence by selecting troops and orders

## Total War ®


Figure 47: Troops selection and moving on field

## Final Fantasy ®


Figure 48: Example of classes in Final Fantasy 3. The outfit gives feedback on some of the behaviors

## Infiny Factory ®

Figure 49: a) action of selected functional block - b) assembly line

## IntuiFace ®

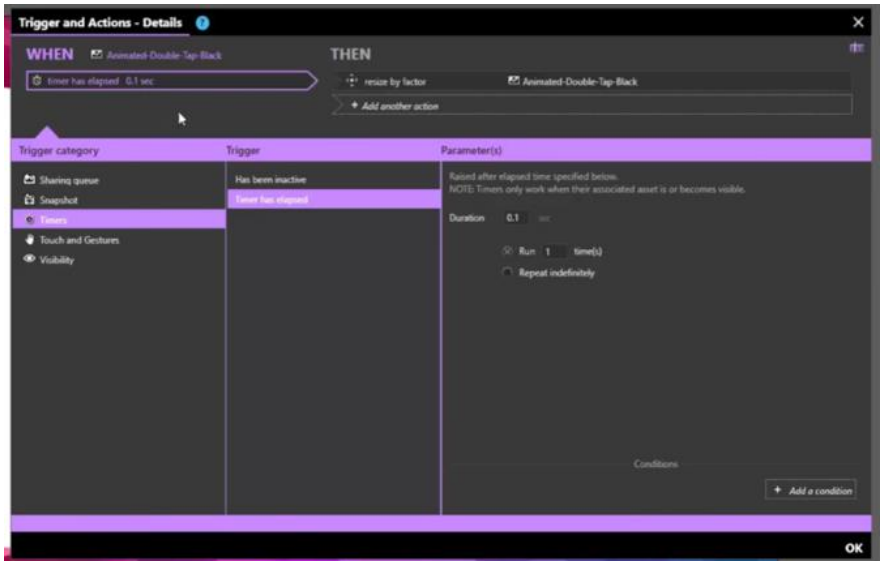



Figure 48: IntuiFace ® trigger and action