Co-construction of Computer Science Knowledge-to-be-taught in a French Context

Vanea Chiprianov, Laurent Gallon, Timothée Duron

Co-construction of Computer Science knowledge-to-be-taught in a French context

T. Duron[1], V. Chiprianov[2] and L. Gallon[1],
[1]LIUPPA, University of Pau and Pays de l'Adour, Pau, France
[2]IRISA, University of South Britain, Rennes, France

Abstract— *The French national curricula for elementary and secondary schools introduced teaching Computer Science (CS) concepts as mandatory, beginning the 2016-2017 school year. This also raised questions related to specifically what CS concepts should be taught and how. Several proposals of textbooks, pedagogical kits and other knowledge-to-be-taught have been made; some of them contain apparently surprising and even what seems, at a first glance, scientifically incorrect knowledge, which could prove to be obstacles in pupils' learning. In this paper we analyze such proposals, and advance explanations based on the Theory of the Didactic Transposition of Knowledge (TDTK). The TDTK considers that the knowledge-to-be-taught is the result of a complex process of various interactions and negotiations between the numerous actors of the educational system. We identify such interactions, which explain the existence of didactic obstacles. Being aware of such caveats may reduce the apparition of this type of obstacles in future construction of similar CS bodies of knowledge-to-be-taught.*

Keywords: K-12, Computational Thinking, Didactics, curricula design and analysis

1. Introduction

The mandatory introduction of Computer Science (CS) concepts in the French elementary (6-10 years old) and middle (11-14 years old) school national curricula, beginning the school year of 2016-2017 [1], has had profound impacts on what and how CS is taught in France. However surprising may this have been for some, it was not totally without precedent: in the 1980s there was a movement of teaching CS concepts (and training teachers accordingly), which was replaced in the 1990s with teaching Digital Literacy; for details on the history of teaching CS in France cf. [2].

Following the international trend on Computational Thinking [3], and reports, such as that of the Academy of Sciences [4], pinpointing France's lateness in adopting proper CS teaching in elementary and middle school, the political decision was taken to introduce CS concepts in the national curricula for these levels. It should be noted that, due to historical decisions [2], at this point France did not have any primary or middle school teachers that had been formally trained in CS, only in Digital Literacy.

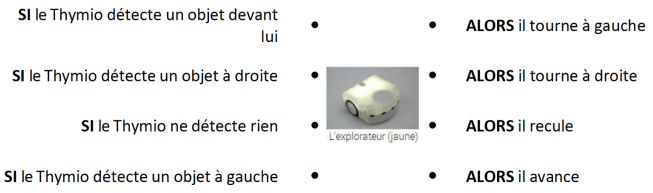

Fig. 1: Excerpt from Mission 3, IniRobot [5]

As a consequence of this introduction, a number of resources, in the form of textbooks, (robot) pedagogical kits, recommendation documents, etc. have been proposed. Designing such proposals is important, however, it is only part of the continuous improvement education process. To evaluate the potential of such resources, questions need to be answered, such as: What is the knowledge such resources aim to teach? How is this knowledge actually taught by teachers and learned by pupils? Moreover, some of these resources contain a number of "features" that could, at first glance, be classified as errors, and which could become obstacles in pupils' learning. How were such "features" introduced and possible in the first place?

One such example concerns the *IniRobot* [5][6] pedagogical kit, using the *Thymio*[1] robot for introducing robotics and programming, especially in elementary[2], but also middle school. IniRobot is distributed under a CC-BY license and has engendered several developments and versions. We focus here on the 2014 version [5].

IniRobot proposes a sequence of 14 missions; of which mission 3 *If ... then ...* (fr. "Si ... alors ...") introduces, in its own terminology, the concepts of *event* and *action*, from event-driven programming. Thymio has 6 preprogrammed behaviors, of which, mission 3 proposes exercises, to discover 4. The connect-the-dots exercise for the behavior called *The explorer (yellow)* (fr. "L'explorateur (jaune)") - is presented in Fig. 1. The "events" (or rather, as we will see below, the *conditions* on which the events are filtered) are introduced by the word IF (fr. "SI"), and the actions are introduced by the word THEN (fr. "ALORS"). The expected result is the discovery of the explorer behavior (using the IniRobot terminology): "IF Thymio detects an object in front

[1]https://www.thymio.org/en:thymio
[2]https://dm1r.inria.fr/t/inirobot-descriptif-des-activites-autour-de-la-robotique-a-lecole-primaire/23

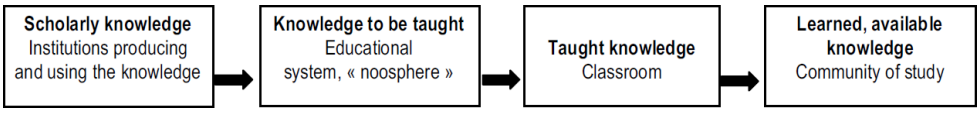

Fig. 2: The didactic transposition process, from [7]

of it, THEN it goes back"; "IF Thymio detects an object at its right, THEN it turns left"; "IF Thymio doesn't detect anything, THEN it advances"; "IF Thymio detects an objects at its left, THEN it turns right". The use of the word IF may come as a surprise, and rightly so, as the concept it introduces is related to an event. The Aseba textual editor for programming Thymio even uses the keyword WHEN, but the Aseba Visual Programming Language (VPL) uses the keyword IF. One would expect the use of the same word in the graphical editor as in the textual one - WHEN (fr. "QUAND"). While this may seem insignificant, there are cases in which pupils are disturbed (c.f. Fig. 3).

In what follows we argue that such "features" are not simple "errors", but the result of a complex process of transformation of the scholarly knowledge into knowledge-to-be-taught. In this process, many actors are involved, with different concerns, numerous constraints, whose solutions are sometimes surprising ...

2. Didactic Transposition of Knowledge

Knowledge, as it is taught in the School, is not immutable. It is a human construct, so as to fulfill a particular goal. As goals evolve through time and to follow changes in society, the knowledge taught in the School evolves as well. As some of the main goals are related to teaching beginners, the knowledge taught in school differs from the knowledge as it was first created by researchers: it is "simplified", differently structured, overlaps and contradictions have been limited - the scholarly, scientific knowledge has been subject to a series of transformations, for it to become knowledge to be taught in School. It has been argued that "[b]odies of knowledge are, with a few exceptions, not designed to be taught, but to be used. To teach a body of knowledge is thus a highly artificial enterprise." [8].

In this paper we investigate how such transformations are happening in the CS curricula that is being defined currently in France. The decisions of which subjects, from the broad CS scientific knowledge, to choose for teaching is influenced by various actors, from high political levels such as ministers and academicians to actors closer to the terrain such as teachers, researchers as collaborators of teachers and as analysts of the system. To analyze this system and how it influences the transformations on the CS knowledge to be taught, we adopt a theoretical background based on the Theory of the Didactic Transposition of Knowledge (TDTK).

Didactics is the science of the diffusion of knowledge in any institution (e.g. class of pupils, society at large). More particularly, it is the scientific study (and the knowledge resulting thereof) of the innumerable actions taken to cause (or impede) the diffusion of such and such a body of knowledge in such and such an institution [9]. The TDTK has originally been proposed in French, in the 1980s, and has achieved a wide spread and acceptance in the French-speaking communities, also in the Spanish-speaking, but much less so in the English-speaking communities, no doubt also because of relatively few and late translations (of which we selected a few in the bibliography of this paper, cf. infra). It is a theory initially proposed in the context of mathematics teaching, but has since evolved to encompass teaching of other science subjects, such as biology and geography.

The *Theory of the Didactic Transposition of Knowledge (TDTK)* [8], [9], [10], studies the "transition from knowledge as a tool to be put to use, to knowledge as something to be taught and learned". Let us note that for us, in this paper, the concept of *knowledge*, in all of its further declinations, comprises both skills, know-how (fr. *savoir faire*), as well as "theoretical" knowledge (fr. *savoir*).

The TDTK distinguishes thus several types of knowledge. The *scholarly (bodies of) knowledge* (fr. *savoir savant*) denotes "an organized and more or less integrated whole" [8]. It is produced by researchers and scholars, usually integrated in theories, to be found in research articles and scientific books. Taking this type of knowledge as starting point, the TDTK studies how it is transformed into *knowledge-to-be-taught* (fr. *savoir à enseigner*), which is the "scholarly knowledge that exists only in contexts than cannot be faithfully replicated within school" [8], usually as it appears in curricula, textbooks and other similar resources. This is further transformed into *taught knowledge* (fr. *savoir enseigné*) - "the knowledge which becomes visible, so to speak, in the classroom" [10], as it is presented in the particular context of a class, by a particular teacher, to a particular group of pupils etc. Of course, there is a further difference between the taught knowledge and what is actually learned by pupils, the *learned knowledge* (fr. *savoir appris*, *connaissances*). A synthetic view of the TDTK is presented in Fig. 2.

These transformations are the result of interactions between actors of the educational system. The totality of these actors form what is called, in the TDTK, the didactic *noosphere* - the "sphere" of those who "think" about teaching [10], all those who share an interest in the teaching system. It consists of the *agents* - those actors who are in charge of knowledge (e.g. teachers), but also of the *laity* - those standing outside the teaching sphere. Obviously, different people have different positions in respect to education.

Therefore, "it is the task of the noosphere to cope with the demands made by society on the teaching system, by transmuting them into conditions acceptable to both parties - society and its teaching system" [10]. Consequently, the central function of the noosphere is the *negotiation* with the society, taking into account different conditions, constraints, resulting in compromises.

We focus in this paper on analyzing the didactic transposition of CS knowledge related to the introduction of event programming based on IniRobot, as it happens in the context of a research and training project - PERSEVERONS[3]. We do emphasize that other actions, knowledge and research are happening in PERSEVERONS, which are out of the scope of this paper. In this context, we have identified as agents of the educational system: the Curricula Superior Council (fr. *Conseil supérieur des programmes*) - the body which defines the national French CS curricula, CS and Learning Sciences researchers, CS pedagogical counselors (fr. *Enseignants référents pour les usages du numérique*), primary and secondary school teachers. In the next sections, we analyze how these agents are interacting in a process of negotiation and *co-construction* - a process of defining together, through interactions more or less direct - of a CS curricula.

3. The co-construction process

In the context of our case study based on IniRobot, it seems as a fair assumption that the knowledge-to-be-taught should revolve around robots and their programming. We investigate in this section the transposition of such knowledge, from the definition of scholarly knowledge of what is called event-driven programming (one of the main paradigms used for programming robots), to its transformation into knowledge-to-be-taught, as found in the French national curricula and in the *IniRobot* pedagogical kit (and one of its declinations, the *IniRobot for School*), and further into taught and learned knowledge, as observed in regular French schools, while emphasizing the actions of different actors involved in each phase.

3.1 Defining the scholarly knowledge

Providing a complete historical and epistemological study on the CS concepts attached to event-driven programming is out of the scope of this paper. However, we do indicate that the concepts usually considered as in relation with the so-called event-driven paradigm have been defined in several places, in different manners, with different names, and although there are commonalities among these definitions, there are also numerous and important differences.

For example, [11] defines a *computational event* as "anything that happens in the course of a computation [...] both occurrences in the program itself [...] as well as occurrences outside the computation proper". A similar and more precise definition is that of [12], for which events are "the transitions between states that may appear in a system or in its environment". Please notice that these definitions consider both the *external* and the *internal* nature of events.

Other authors, like for example Turing-award winner L. Lamport [13], offer a more complete picture, speaking of several concepts: *events* (which are not formally defined), *processes*, *messages* (defined as the means of communication among spatially distinct processes - distributed systems - and whose sending or receiving are considered an event), *partial ordering* (of the sending and reception of messages), *logical* and *physical clocks* (for synchronization), etc.

[14] speaks about *waiting* for an event - in which case the program cannot complete an operation immediately and thus it registers a *callback* - a function that will be invoked when the event occurs. This waiting is typically done in a *loop* that *polls* for events and executes the appropriate callback when the event occurs. To differentiate between events, mechanisms of "filtering [...] by a condition" [12], which provide an event every time a condition is true, are needed. A concept similar to the *callback* is the *event handler* - a method "able to respond to one kind of external action (or event)" [15]. An *event* is defined as an *external* action.

Other researchers make a difference between what it is called threaded (or procedure-oriented) and event-driven (or message-oriented) programming models/systems [14], trying either to show their duality e.g. [16], or the precedence of one over the other (at least in some contexts) e.g. [14][17].

Defining the concepts related to the so called "event-driven programming paradigm" has therefore been a long process, in which several *CS researchers* (the only type of actors - and more precisely laity in TDTK terminology - in this phase) have built on or against the work of previous researchers. The resulting scholarly knowledge may have attributes that enable construction of real-world applications (use of knowledge), but it is in itself a rather difficult and specialized body to understand, sort and classify.

3.2 Defining the knowledge-to-be-taught : the curricula

In France, a distinction is usually made between 2 types of knowledge-to-be-taught: the national curricula, resulted from the external transposition process of scholarly knowledge, and which serves as a reference document for the second type of knowledge-to-be-taught, which comprise textbooks, teacher training documents and other similar documents. The curricula is itself the result of a negotiation process between several organizations and councils. The Curricula Superior Council is the body which defines the national French curricula for elementary and middle schools. However, it does not work in isolation, and an analysis of its CS 2016 curricula [1] shows a number of influences, from a report of the Academy of Sciences [4], to other organizations such as

[3]http://perseverons.espe-aquitaine.fr/

EPI[4] and ATIEF[5].

Regarding knowledge related to robots and event-driven programming, the French national curricula for middle school for example [1, pp. 365, 380], includes requirements such as: "write a program in which actions are triggered by exterior events", or "triggering of an action by an event". It is important to note that the CS concepts are mentioned in the curricula in 2 distinct parts: *Mathematics* and *Technology*. While the requirements evoked above are common to both parts, there are CS concepts specific to only one of them. As such, the *Mathematics* part also includes: "pupils are introduced to event programming (fr. programmation événementielle)", "programming actions in parallel", "control structures related to events", while the *Technology* part includes: "sensor, actuator, interface". One can see that the *Mathematics* part is more focused on the programming side, while the *Technology* part also includes more pronounced engineering and mechanical elements.

One can identify, in the curricula, CS concepts that have been selected from the scholarly knowledge, such as *event* (which in the curricula seem to concern mainly external events - so *messages*?), *action* (probably similar to the concepts of *callback/handler*), *triggering* and *control structures related to events* (probably related to *waiting* and *loop polling*), *actions in parallel* (related to *processes*? or another name for events?). Why and how have been these concepts selected? A possible answer may come from what seems to be a major source of inspiration for the CS part of the curricula, a report of the Academy of Science [4, pp. 23, 25], which indicates, in relation to event-driven programming, concepts such as: "notion of parallel algorithm", "sensors and actuators", "algorithms [...] control the system by acting on actuators depending on the sensed values", "the retroactive command in a closed loop".

However, one can notice that, while the Academy report presents together more mechanical (sensors, actuators) and more programmatic (algorithm, loop) concepts, the curricula segregates them into a *Mathematics* and a *Technology* part. How could this be explained? As mentioned in the introduction on the history of teaching CS in France, starting from the 1990s, teachers were trained mainly to Digital Literacy, but not to CS (programming). Therefore, in the 2010s, when the (political) decision to introduce CS teaching was taken, the Curricula Superior Council found itself confronted with the reality of having no teachers properly trained. It seems the retained solution was to separate the CS concepts into 2 subjects and assign their teaching to teachers which were the most likely to have connections to (and hopefully interest in) these concepts. The French national curricula is therefore a good example of how different actors, from political decision makers, the Academy of Science and the Curricula Superior Council negotiated (indirectly), according to their purposes and with real-world constraints.

One can also notice that the curricula identifies CS concepts to be taught, but does not enter into details. For example, *triggering* is mentioned, but not *filtering* by a condition; the concept of *event* appears, but no discussion of the time implications, on questions related to the *partial ordering* of sending and/or reception of events. As we shall see, this may have had implications on the further transposition of the knowledge-to-be-taught.

3.3 Defining the knowledge-to-be-taught: the case of pedagogical kits

If a number of decisions regarding choices among scholarly knowledge, such as focusing on certain concepts of event-driven programming, as well as a separation, at middle school, into Mathematics and Technology, were taken, as a result of an external transposition process between laity of the educational system such as the Academy of Science and political decision makers, on the one hand, and agents such as the Curricula Superior Council, on the other hand, the identified concepts remain described rather vaguely and a number of choices still need to be made. One such more detailed specification is provided by the IniRobot [6] pedagogical kit. With a version made available in 2014 [5], so somehow in parallel with the development of the curricula itself, mainly by actors which seem to be primarily CS researchers or school teachers, with interests in Education Sciences research, IniRobot has several declared objectives, among which "the acquisition and practical use of a number of fundamental concepts", such as, for example: "sensors", "actuators", "instruction", "algorithm" and "how to use basic concepts of event-based programming", by elementary and middle school pupils [6]. It seems therefore safe to assume the authors of IniRobot followed the principle of the *low-floor* (easy to get started) [18], and tried to avoid as much as possible known difficulties.

"Programming in explicitly event-driven models is very difficult" [19]. Several causes have been identified for this, among which we find particularly relevant for our discussion the fact that the interactive logic of a program is *fragmented* across multiple event handlers [14][19]. While sequential programming is structured as a single flow of control, with control structures such as branching (i.e. IF) and loops, event-driven programming requires a series of small callbacks/handlers. The control flow among these handlers is not obvious, because of the *inversion of the control* [19]: a program merely registers with the execution environment its interest to be resumed on certain events; it is the execution environment which dispatches the events to the event handlers; it is not the program which calls the handlers. Thus, the control flow among handlers is expressed only implicitly, through manipulation of shared state [15].

[4]https://www.epi.asso.fr/
[5]http://atief.fr/

It is maybe to avoid such difficulties related to understanding the fragmented and inverted control flow that the authors of IniRobot proposed the use of the word "IF" to introduce what they call an "event", as we have seen in Fig. 1. This may help "lowering the floor" [18], the entry point for pupils, when first encountering event-driven programming, by approaching the fragmented event-driven control flow to a sequential one. This seems the more likely as, in special (simple) cases, the control flow of an event-driven program may actually be sequential. Therefore, as long as the situations/exercises only require sequential code, pupils having a sequential representation of the event-driven programming does not show to be problematic. However, are they, in this case, really taught event-driven programming? Moreover, the similarity of using logical conditions: from the filtering of an event by conditions when *triggering/waiting/loop polling*, and, respectively, of the branch condition, increases the similarity between the 2 concepts.

Nonetheless, the semantics of *filtering of events by conditions* (let's denote it by the keyword WHEN) is different from that of a *branch* (IF). A filter has to be unique in a program, while branching conditions may appear multiple times. This becomes visible in some cases, such as that presented by the situation introduced by IniRobot in Mission 10, analyzed in more detail in Fig. 3.

The 2014 version of *IniRobot* has evolved, engendering several branches and multiple versions. We focus here on a 2016 version, *IniRobot for School* [20], the basis chosen for activities in PERSEVERONS. First change to notice is the dimension of the document: from 24 pages for IniRobot, to 80 pages for IniRobot for School. The authors of IniRobot for School are CS pedagogical counselors, whose declared objectives comprise, among others: "working numerous competencies related to mastering the language (oral and writing), mastering mathematical languages and mastering scientific languages" [20, p. 2]. While there still are important objectives related mainly to teaching CS concepts, IniRobot for School uses the activities with Thymio "to practice languages", "to practice scientific and technological procedures" and "to organize the personal work" [20, p.3] as well, in relation with other objectives of the French national curricula (defined for all school subjects) [1]. While IniRobot consists of 6 sessions containing a total of 14 missions, with a total estimated completion time of about 6 hours (possibly longer, depending on audience), IniRobot for School consists of 12 sessions, during which contents from IniRobot missions 1, 2, 3, 4, 5, 6, 7, 8, 10 are partially reused, reordered, developed and enriched, for a total of an estimated completion time of about 12 hours.

The authors of IniRobot for School have thus selected, according to their objectives, from IniRobot, contents related to CS concepts found in the curricula, excluded others (e.g. missions 11, 12, 13 from IniRobot) and added other knowledge (e.g. mission 1 on drawing Thymio), not necessarily related to CS, in order to augment the interdisciplinary content of their kit. Let's also note that, in IniRobot for School, while concepts such as *event* and *action* are mentioned several times and exemplified, no mention is made to *conditions* or *filtering/waiting*; which is, as we have seen, in accordance with their lack of mention in the curricula.

3.4 Defining the taught knowledge

At this phase in the transposition, it is the teachers' turn to intervene in the process. Based on the curricula, textbooks, pedagogical kits, other resources and the training they (are supposed to) have received, the French educational system expects teachers to define situations to be taught in the classroom, that introduce and argue the need of CS concepts. These are high expectations, difficult to meet, especially by teachers with no initial training in CS.

To support the teachers in this rather daunting task, the decision makers, most notably the Prime Minister and the Ministry of Education, have intervened through the e-FRAN[6] Spaces for training, research and digital organization call for projects. It has selected 22 projects, for a total of 19.5 Million Euros. The successful projects demonstrated that they federate schools, county local authorities, companies, research laboratories, and other actors, around an innovative project with objectives regarding digital objects - whether to use them as a pedagogical resource, or in relation to new skills to be acquired, or as a research object -, taking place on diverse territories which enable a precise monitoring and evaluation (especially of pupils involved in experiments).

As part of one such project, PERSEVERONS, we have the opportunity of observing and analyzing interactions between the numerous actors involved. These interactions have driven the transposition of CS knowledge-to-be-taught, as defined by the curricula, textbooks and pedagogical kits. We have observed several teachers and classrooms, keeping records in the form of notes, videos and photos, following an *observation*-based, *case study*, *document and artifact*-driven *content* analysis, *qualitative research method*[7] [21].

It should come as no surprise that, even though the teachers used the same main pedagogical kit, IniRobot for School, there were differences in the way the class time was organized, the emphasis on some explanations or situations, the general and CS-specific teacher experience etc. However, some conditions were fairly common, such as pupils working in groups of 2 or 3, with 1 computer and 1 Thymio, in half-classrooms of similar sizes, between around 10 and 15, for about 1 hour, in 1-2 sessions per week, for 4 - 10 weeks. This allowed observing some recurring phenomena across

[6] http://www.education.gouv.fr/cid94346/appel-a-projet-e-fran.html

[7] With its characteristic concern for context and meaning, of experiments in naturalistic occurring settings, in which the human investigator is the primary instrument for the gathering and analyzing of data, which is mainly descriptive, data obtained as a result of an emergent study design, and on which inductive analysis is applied.

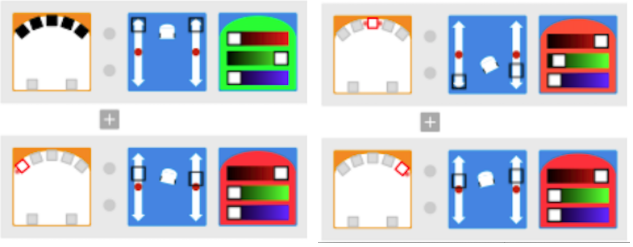

Fig. 3: Expected solution to programming Thymio's explorer behavior, from [5]

the different classes, among which the one related to the use of the IF keyword, described in more detail hereafter.

3.5 Defining the learned knowledge

One phenomenon, a recurrent pupils' "mistake" observed in several contexts and situations, can be exemplified by *Mission 10* from *IniRobot*, optionally part of *Session 10* of *IniRobot for School*. It asks pupils to program Thymio so that it moves about freely, avoiding all obstacles (corresponding to its *explorer* behavior, cf. Fig. 1); optionally, it asks to add instructions so that Thymio changes its color to red if it detects an obstacle, and to green if not. The expected solution (cf. Fig. 3 [5, p. 23]) consists of 4 instructions (grey blocks), each corresponding to an event (at the left of the ":" sign), and 2 actions (at the right ":"). In IniRobot's terminology, their respective semantics would be [5, p. 14]: if Thymio does not detect anything with its front sensors, advance and change color to green; if Thymio detects something at its left, turn right and change color to red; if Thymio detects something in front of it, back away while turning a little and change color to red; if Thymio detects something at its right, turn left and change color to red.

One "mistake" pupils make (cf. Fig. 4) is that, instead of adding a second action to the right of the ":" sign, for the same "event", they add a second instruction, with the same "event". This is signaled by the compiler as an error (in red at the top of Fig. 3) of the type: the events are the same; of course, for the program to be deterministic, the event callback/trigger has to be unique. While there may be several explanations for this "mistake", such as understanding that each "event" has only one corresponding action/instruction (while actually, at the right of the ":" sign, there is a block of instructions, which may contain an action of each type).

However, one possible explanation, on which we focus here, is that the pupils constructed a representation of the *event* concept which is close to a sequential one, and which therefore allows them to verify the same condition (**if** all of Thymio's frontal sensors do not detect anything, in the example of Fig. 3) several times. It would seem that constructing a representation of the *event* concept that includes time aspects (with a semantics of **when (each and every time)** all of Thymio's frontal sensors do not detect anything, in the given example), would help in understanding the uniqueness of the type of event and of its attached actions. While approaching the event-driven control flow to a sequential one may "lower the floor", a more appropriate representation needs to replace it as soon as possible, if "real" event-driven programming is to be taught.

While this is a fairly straightforward example, its analysis helps understanding difficulties encountered at each phase of the didactic transposition *co-construction* process, from (1) the esoterism of the scholarly knowledge on event-driven programming (in which time considerations are made using concepts such as *partial ordering*, and conditions are defined as part of a *filtering* of events, in a *waiting/triggering/loop polling* mechanism), defined by sometimes competing, sometimes continuing CS researches, passing through (2) the selection of concepts to-be-taught, selection influenced by political decision makers, academic institutions (laity in the sense of TDTK), national curricula definition bodies (which in our case study seem to leave out these time and conditions considerations), (3) the redefinition and reorganization into textbooks, pedagogical kits, or similar resources, by researchers, teachers and pedagogical counselors mainly concerned with education issues, with easing their understanding by pupils (and thus, in our case, seemingly approaching the event-driven flow of control to a sequential one by using keywords like "IF" instead of "WHEN"), (4) the development of classroom lessons by teachers and finally (5) the continuous evolution of pupils' representation of the taught concepts, who by "mistakes" experiment/test and construct this representation (in our case, e.g. assuming that conditions/events are not unique).

4. Conclusion

While most current research related to Computational Thinking (CT), especially in a K-12 context, has focused mainly on designing activities for teaching CT concepts, such pedagogical documents need to be analyzed and experimented in classrooms, so that they can be validated and improved. In this paper we adopted a qualitative approach, as a first step to identifying phenomena that appear recursively in classrooms. This allowed us to identify didactic obstacles to the learning of pupils. Further social and epistemological-driven analysis revealed causes of such didactic obstacles in

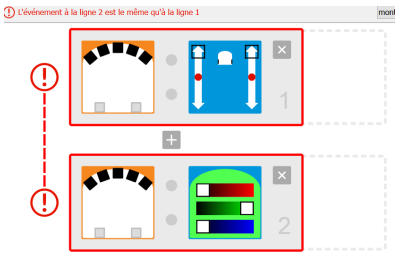

Fig. 4: Typical pupils' mistake

the complexity of the didactic transposition process involving numerous actors, with various constraints and objectives.

More specifically, we focused on event-driven programming, observing and analyzing during 4 years, how the *IniRobot for School* pedagogical kit using the *Thymio* robot, was used to teach and learn concepts related to the event-driven paradigm. We thus pinpointed that concepts of event-driven programming form the main focus of CS knowledge-to-be-taught in elementary and middle school (other fully arguable choices include robotics, intelligent (cooperative) (autonomous) agents, ethics etc). In an epistemological approach, we reviewed scholarly, research-introduced definitions of concepts that currently are considered to be related to what is called event-driven programming, showing their relations and underlining their complexity (e.g. external vs. internal events, messages vs. processes). We analyzed how some of these concepts were selected and partially redefined (e.g. event, action) in the French national curricula and identified how the lack of mention of certain concepts (e.g. filtering on conditions), may contribute to further difficulties. We found that these selections are indeed reflected in the pedagogical kits that implement the curricula, and are compounded with didactic and pedagogical concerns of "simplifying" the taught concepts (e.g. presenting the fragmented event control flow as a sequential one). This analysis (in particular the review of the scholarly knowledge) may serve as a basis in future designs of event-driven programming pedagogical proposals.

In this work, we mainly focus on *didactic obstacles*. While, for example, the difficulty of understanding the fragmented control flow of event-driven programming is an *epistemological obstacle*, the didactic choice of the "IF" keyword (while intended precisely to reduce this epistemological obstacle), with its unforeseen side effects, introducing *didactic obstacles* (sequential representation of the occurrence of events), created by the very redefinition (out of didactic concerns) of some event-driven concepts. While they are probably unavoidable in any didactic transposition process, it is important to recognize such obstacles, be aware of them, and try to address them further along the line.

This analysis can therefore be used to draw the attention, in particular, to curricula and pedagogical documents (textbooks, kits, etc.) designers, to pay singular attention to how they redefine scholarly concepts, as these "new" representations - the knowledge-to-be-taught, may introduce its own obstacles. Moreover, this analysis may serve researchers and analysts of the knowledge-to-be-taught (be it curricula, textbooks, pedagogical kits) as caution to drawing conclusions too hastily, and instead exercise a deeper consideration of the wider context and the complex interactions between the involved actors (be they agents of the education system or laity), which may influence decisions seeming, at a first glance, "weird" or simply "erroneous". It thus enlightens the fact that the "final" result of the taught knowledge is the product of a process of co-construction, in which many actors intervene at different stages.

References

[1] M. de l'Éducation Nationale de l'Enseignement Supérieur et de la Recherche MENSR, "Programmes d'enseignement du cycle des apprentissages fondamentaux (cycle 2), du cycle de consolidation (cycle 3) et du cycle des approfondissements (cycle 4)," *Bulletin Officiel no 11 du 26 novembre*, 2015.

[2] G.-L. Baron, B. Drot-Delange, M. Grandbastien, and F. Tort, "Computer science education in french secondary schools: Historical and didactical perspectives," *ACM Transactions on Computing Education (TOCE)*, vol. 14, no. 2, p. 11, 2014.

[3] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006.

[4] A. de Sciences AdS, "L'enseignement de l'informatique en france. il est urgent de ne plus attendre," 2013.

[5] T. Guitard, D. Roy, P.-Y. Oudeyer, and M. Chevalier, "IniRobot. Activités robotiques avec Thymio II pour l'initiation a l'informatique et a la robotique," 2014. [Online]. Available: https://dm1r.inria.fr/t/inirobot-les-documents-a-telecharger/141

[6] D. Roy, G. Gerber, S. Magnenat, F. Riedo, M. Chevalier, P.-Y. Oudeyer, and F. Mondada, "IniRobot : a pedagogical kit to initiate children to concepts of robotics and computer science," in *RIE 2015*, Yverdon-Les-Bains, Switzerland, May 2015.

[7] M. Bosch and J. Gascón, "Twenty-five years of the didactic transposition," *ICMI Bulletin*, vol. 58, pp. 51–65, 2006.

[8] Y. Chevallard, "On didactic transposition theory: Some introductory notes," in *International Symposium on Research and Development in Mathematics, Bratislava, Czechoslavakia*, 1988.

[9] ——, "Readjusting didactics to a changing epistemology," *European Educational Research Journal*, vol. 6, no. 2, pp. 131–134, 2007.

[10] ——, "A theoretical approach to curricula," *Journal fuer Mathematik-didaktik*, vol. 13, no. 2-3, pp. 215–230, 1992.

[11] D. Jusak and J. Hearne, "Language and runtime support for event programming in a distributed system," in *[1990] Proceedings. Second IEEE Workshop on Future Trends of Distributed Computing Systems*, Sep 1990, pp. 514–519.

[12] P.Caspi and N.Halbwachs, "An approach to real-time systems modeling," in *Int. Conference on Distributed Computing Systems*, 1982.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[14] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris, "Event-driven programming for robust software," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, ser. EW 10. ACM, 2002, pp. 186–189.

[15] B. Chin and T. Millstein, "Responders: Language support for interactive applications," in *Proceedings of the 20th European Conference on Object-Oriented Programming*, ser. ECOOP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 255–278.

[16] H. C. Lauer and R. M. Needham, "On the duality of operating system structures," *SIGOPS Oper. Syst. Rev.*, vol. 13, no. 2, pp. 3–19, 1979.

[17] R. von Behren, J. Condit, and E. Brewer, "Why events are a bad idea (for high-concurrency servers)," in *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*, ser. HOTOS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 4–4.

[18] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[19] P. Haller and M. Odersky, "Event-based programming without inversion of control," in *Proceedings of the 7th Joint Conference on Modular Programming Languages*, ser. JMLC'06. Springer-Verlag, 2006, pp. 4–22.

[20] J. Sagné, E. Page, and C. Lefrais, "Séquence IniRobot scolaire. «langages et robotique »," 2016. [Online]. Available: http://tice33.ac-bordeaux.fr/Ecolien/Langagesetrobotique/tabid/5953/language/fr-FR/Default.aspx

[21] D. Ary, L. C. Jacobs, C. K. S. Irvine, and D. Walker, *Introduction to research in education*. Cengage Learning, 2018.