Human-Oriented Formal Modelling of Human-Computer Interaction: Practitioners' and Students' Perspectives

Antonio Cerone(✉)

Department of Computer Science, Nazarbayev University, Astana, Kazakhstan
antonio.cerone@nu.edu.kz

Abstract. Practitioners and students tend to have a negative inclination towards formal methods and consider them hard to learn and unusable in practice. In this paper we analyse the perspectives of practitioners, computer scientists and students to show that a notation developed for modelling interactive systems in previous work and its translations into rewriting logic and process algebra represent an appropriate compromise among such perspectives.

Keywords: Human computer interaction · Formal methods applications · Computer science education

1 Introduction

Formal methods experts are often so much focussed on the investigation of theoretical aspects of formal notations rather than on their applications to real problems, that they often neglect the needs of practitioners. As a result, they produce methods of little use in practice and have to resort to simplified, unrealistic, too abstract versions of application-domain problems while they are also biased in choosing the data that best illustrate the features and potential of their favourite formal languages and analysis techniques [5]. Instead of focusing on usability, formal analysis and the tools that realise it are more and more evolving towards efficient, automatic rather than human-oriented proofs (theorem-proving) [1] or the checking of rich extensions of temporal logic, which are hard to understand by humans (model checking). Nothing of this is of any interest for a practitioner. In addition, formal methods are often presented to students through dry syntax and involved semantics rather than seen in a lively applicative context through lab sessions that allow students to use appropriate, usable tools to experiment both with learning-oriented, fun-making examples and, when familiarity is acquired, with real world case studies [4].

In this paper we consider three human-oriented perspectives in which formal methods can be used to model human-computer interaction.

First, the development of new approaches to the use of formal methods and tools within a specific application domain should address the perspective

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 232–241, 2016.
DOI: 10.1007/978-3-319-50230-4_17

of and provide an effective support to the practitioner (who is normally a domain expert) and should be tested on real case studies from the application domain. This aspect is dealt with in Sect. 2, where modelling and analysis goals and objectives, as well as description level, are established from the perspective of a practitioner, who is, in our context, a cognitive psychologist/scientist. An intuitive but unambiguous notation to model cognitive processes, subset of a more extensive notation presented in previous work [3], is illustrated in the context of the practical use of the practitioner.

Second, the intuitive description of the problem given from a domain expert perspective has to be translated or, technically speaking, implemented into an appropriate formal language, which, on the one hand, is equipped with powerful tools that support the accomplishment of the modelling and analysis objectives of the domain expert, and, on the other hand, is close enough to the intuitive modelling notation used by the domain expert. Section 3 provides another notation, also from our previous work [3], with simple primitives to describe the behaviour of interfaces, which reflects, in a simplified form, the formal methods perspective of modelling a system in terms of transitions between states. Section 4 merges the notations corresponding to the two perspectives into a modelling language for interactive systems, which has been implemented using two distinct formal methods, rewriting logic and process algebra.

Third, it essential to address learners' and practitioners' negative inclination towards formal methods and provide the appropriate educational tools to allow learners, as well as practitioners, to overcome the prejudice that "formal methods are hard to learn and to use". This should be done at the root, by making learning formal methods motivating, appealing and involving for students. Therefore, in Sect. 5 we discuss the role of our modelling language and its translations into rewriting logic and process algebra in the context of an PhD course on applied formal methods.

2 A Perspective from Cognitive Science

Cognitive science is an interdisciplinary field that comprises various research disciplines, including psychology, artificial intelligence, philosophy, neuroscience, linguistics, and anthropology [13]. Moreover, it adopts the so-called *information processing* approach, whereby human cognitive processes are modelled as processing activities that make use of input-output channels, to interact with the external environment, and three main kinds of memory to store information: *sensory memory*, where information perceived through the senses persists for a very short time; *short-term memory (STM)*, where the information that is needed for processing activities is temporary stored; *long-term memory (LTM)*, where information is organised in structured ways for long-term use [6]. In this sense computer scientists and cognitive scientists share the view of a processing system with components for input, output and storage of information, and information streams flowing between different components (*computer analogy*).

In spite of this important commonality, the perspective of a cognitive scientist tends to be very different from the perspective of a computer scientist, especially

if the computer scientist is a formal methods expert. In particular, a cognitive scientist tends to see a model as conceptual rather than formal and give it representations that are visual rather than mathematical and/or symbolic. Even when a mathematical representation of the model is conceived, this is envisaged as an operational tool to be used only for description or simulation purposes. Furthermore, even cognitive scientists who work in the area of human-computer interaction are not keen to use formal descriptions, but prefer to adopt instead the scientific method and analyse and evaluate systems using empirical and measurable evidence, systematic observation and usability experiments.

In terms of general goals a domain expert aims at

1. describing domain phenomena with a notation that represents them in an intuitive way by providing a direct representation of the basic components and processes of the domain;
2. using tools that can:
   (a) automatically manipulate such notation to generate simulations of the domain phenomena and map the results on the structure of the domain components;
   (b) extract global information and general properties from an extensive set of simulations.

Therefore, the respective *objective* of a cognitive scientist are

1. a notation to define cognitive processes in terms of how the different components of human cognition (perception, attention, memory, reasoning and action) cooperate to process information, possibly with the support of a digital device or system through its interface, to accomplish specific goals of human behaviour;
2. the availability of tools that:
   (a) allow a simulation mapped on the various cognitive and non cognitive components;
   (b) provide analysis features to conduct in-silico experiments to overcome a major difficulty in cognitive science field studies and lab experiments, that is, that human behaviour, the main object of study, is characterised by multiple aspects, such as unpredictability, ethical issues, individual and cultural diversity, inaccessibility of introspective processes and slow evolution, that hinder the design of the research plan and the validity of the results.

In order to address these human-oriented objectives and link the two distinct perspectives of a cognitive scientist and a computer scientist, we devised an intuitive, formal notation to describe the components of a cognitive system and the information flow among them [3]. We consider only STM as a dynamic memory, that is, supporting both storage and retrieval of information; LTM is, instead, implicitly seen as a container of all knowledge needed for the processing activities, as already given rather than dynamically constructed, with only retrieval and no storage (i.e. no transfer between STM and LTM); sensory memory is not represented at all.

2.1 A Formal Notation for Human Cognition and Behaviour

Input and output occur in humans through senses. We give a general representation of input channels in term of *perceptions*, with little or no details about the specific senses involved in the perception. We represent output channels in term of *actions*. Actions are performed in response to perceptions.

Human behaviour is driven by *goals*. In order to accomplish a goal in a specific domain of action, the human has to carry out a *task*, that is, an operation to manipulate the concepts of the domain. This is done by performing actions. In an interactive context, namely while interacting with an interface, each action is normally executed as an automatic response to a specific perception (*automatic control* or *automaticity*). For example, automaticity is essential in driving a car: the driver is aware of the high-level tasks that are carried out, such as driving to office, turning to the right and waiting at a traffic light, but is not aware about low-level details such as changing gear, using the indicator and the colour of the traffic light, amber or red, while stopping at a traffic light. A goal is associated with a top-level task. A top-level task can be decomposed in a hierarchy of tasks until reaching basic tasks, which cannot be further decomposed. We model a basic task as a quadruple

$$info_i \uparrow perc_h \Longrightarrow act_h \downarrow info_j$$

where perception $perc_h$ triggers the retrieval of information $info_i$ from the STM, the execution of action act_h and the storage of information $info_j$ in the STM. We formally denote by *none* when there is no information to retrieve from or store in the STM.

Information is kept promptly available, while it is needed to perform the current top-level task, by storing it in the STM. For the purpose of our work we consider only two kinds of information that can be stored in the STM: *task goal*, represented as the action that directly accomplishes the goal, and *action reference*, which refers to a future action to be performed. A task goal is formally modelled as $goal(act)$ where act is the action that directly accomplishes the goal.

As an example, a simple Automatic Teller Machine (ATM) task, in which the user has only the goal to withdraw cash, is modelled by the following four basic tasks

1. $none \uparrow cardR \Longrightarrow cardI \downarrow cardB$
2. $none \uparrow pinR \Longrightarrow pinI \downarrow none$
3. $none \uparrow cashO \Longrightarrow cashC \downarrow none$
4. $cardB \uparrow cardO \Longrightarrow cardC \downarrow none$

where: $cardR$ denotes the perception that the ATM is ready to receive the card, $pinR$ that it has requested the pin, $cashO$ that it has delivered the cash and $cardO$ that it has delivered the card; $cardI$ denotes the action of inserting the card, $pinI$ inserting the pin, $cashC$ collecting the cash and $cardC$ collecting the card; $cardB$ is the action reference used as a memory for the card collection (it refers to action $cardC$). The goal ("to withdraw cash") is identified with the act of collecting cash (action $cashC$) and is formally modelled as $goal(cashC)$.

3 A Perspective from Formal Methods

We consider one possible formal methods perspective in which the system behaviour is seen as a discrete sequence of state changes. We apply this perspective to the context of a user interacting with an interface. Normally an interface provides an output to the user and waits for the user action (i.e. reaction), which is seen as an input that triggers a change of state. In some cases the current state $state_h$ is associated with a timeout: if user's reaction act_h occurs before the timeout expires, then it triggers the change to state $state_k$, otherwise, at the expiration of the timeout, the state changes to state $state_r$, which may be distinct from $state_k$. In order to associate timeouts with interface states, we decorate interface states as follows.

$state!0$ state not associated with a timeout;
$state!1$ state associated with a timeout that is not expired;
$state!2$ state associated with a timeout that has already expired.

Thus we model a state change as a triple

$$state_h!m \xrightarrow{act_h} state_k!n$$

where interface state $state_h$, with possible timeout characterised by m, triggers the execution of action act_h with a change to state $state_k$, whose possible timeout is charcterised by n. The initial state of the interface is normally an idling state (the interface is available for an interaction), thus it is not associated with a timeout ($state!0$). If we have

$$state_h!1 \xrightarrow{act_h} state_k!n_k$$

and the timeout associated with $state_h$ expires, than $state_h!1$ changes to $state_h!2$ and the state change that occurs at the timeout expiration is modelled by

$$state_h!2 \longrightarrow state_r!n_r$$

where the absence of action denotes that there is no interaction with the user, thus describing an autonomous action of the interface.

4 A Common Perspective

We have seen in Sect. 3 that an action act is performed through a cooperation between the human (the subject performing the action) and the interface (which changes its internal state as a consequence of the human action). Therefore, an action belongs to both a task and an interface transition and represents the basic form of interaction. In the context of an interactive system, a user perception refers to a stimulus produced by an output of the interface with which the human is interacting. We can thus identify the perception with such an output. Moreover, since the output of the interface is associated with the interface

state that results from producing that output, we can take a step forward and identify the user perception with the interface state associated with the output that produced that perception. For example, the interface state associated with the interface of a vending machine giving a change is identified with the perception (sound of falling coins or sight of the coins) produced. Thus, in our notation, interface state and corresponding human perception are denoted by the same formal entity (which, assuming the cognitive scientist's perspective, we call "perception" rather than "state"). In this way our formal notation meets Objective 2 presented in Sect. 2.

Identifying interface state and corresponding human perception allows us to merge the two notations presented in Sects. 2 and 3 and attain a modelling language for interactive systems. A state change is thus modelled as

$$perc_h!m \xrightarrow{act_h} perc_k!n$$

where $perc_h$ is the perception that triggers the user to perform action act_h, which causes the interface to change to the state corresponding to perception $perc_k$. As an additional link between the two merged notations, we keep track of the human action act_h, if any, that produced the state $perc_k$ by defining an interface state as a pair $act_h \gg perc_k!n$. The initial state becomes then $\gg perc!0$.

With reference to the ATM example introduced in Sect. 2.1, we model an old interface that sequentially requests a card, requests a pin, delivers the cash and returns the card, and a new interface that returns the card before delivering the cash. The two interface models are as follows.

Old ATM: transitions	**New ATM: transitions**
1. $cardR!0 \xrightarrow{cardI} pinR!1$	1. $cardR!0 \xrightarrow{cardI} pinR!1$
2. $pinR!1 \xrightarrow{pinI} cashO!1$	2. $pinR!1 \xrightarrow{pinI} cardO!1$
3. $cashO!1 \xrightarrow{cashC} cardO!1$	3. $cardO!1 \xrightarrow{cardC} cashO!1$
4. $cardO!1 \xrightarrow{cardC} cardR!0$	4. $cashO!1 \xrightarrow{cashC} cardR!0$
5. $pinR!2 \longrightarrow cardO!1$	5. $pinR!2 \longrightarrow cardR!0$
6. $cashO!2 \longrightarrow cardO!1$	6. $cashO!2 \longrightarrow cardR!0$
7. $cardO!2 \longrightarrow cardR!0$	7. $cardO!2 \longrightarrow cardR!0$

For both interfaces the initial state is $\gg cardR!0$. In both interfaces, transitions 1–4 model the normal sequences of interactions for the specific design (old or new).

The last three transitions model interface autonomous actions. If the timeout expires after requesting a pin (transitions 5), then in the old ATM the card is returned, whereas in the new ATM the control goes back to the initial state, implicitly modelling that the card is confiscated by the ATM, and in both cases the cash delivery is inhibited. If the timeout expires after delivering the cash (transitions 6), then in the old ATM the card is returned, whereas in the new ATM the control goes back to the initial state, so inhibiting a cash collection action in both cases and implicitly modelling that the cash is taken back by the ATM. Finally, in both interfaces, if the timeout expires after returning the

card, then the control goes back to the initial state, so inhibiting a card collection action and, as a result, implicitly modelling that the card is confiscated (transitions 7), obviously, with no cash delivery in the new ATM.

Our modelling language for interactive systems has been translated into rewriting logic [3] and implemented using the MAUDE rewrite system[1], and into the CSP (Communicating Sequential Processes) process algebra [2] and implemented using the Process Analysis Toolkit (PAT)[2]. Both tools, MAUDE and PAT, are equipped with model checkers, thus featuring the potential for meeting Objective 2 from Sect. 2. In reality, the simulators and model-checkers of the two tools produce results that refer to the low-level structures that implement the modelling language with no mechanisms to present the effect of such results on the high-level cognitive and non cognitive components, which the practitioner is familiar with. Therefore, implementing such mechanisms, such as domain specific visualisations [9], would be necessary to accomplish Objective 2.

5 Students' Perspective

There is an ongoing debate on the importance of formal methods to computer science education. This debate links with the wider debate on the centrality of mathematics and logic in computer science curricula: on the one side the claim that rigorous mathematical knowledge is not necessary for computer science practitioners [7] and, on the other side, the belief [14,15] and the empirical evidences [10–12] that learning rigorous discrete mathematics and formal methods has an important impact on problem-solving and programming skills and is perceived by students as useful in practical problems and helpful in improving their mental processes [16].

We agree with the latter position but, in addition, we believe [4] that:

1. instead of tediously going through the semantics of each construct in a formal language, students should be allowed to experiment with an appropriate tool to discover the semantics by themselves;
2. tools for simulation visualisation are essential to allow students to understand the behaviour associated with their models.

Moreover, in order to motivate students, formal methods should be presented in a variety of realistic, applied contexts, not at all limited to computer science and software engineering, and including, why not, examples that can bring some fun [4] in an apparently very serious area. The recent application of formal methods to several disciplines such as biology and cognitive science provides heaps of interesting and motivating examples.

The rewrite systems and CSP translations of our modelling language were presented during a course on "Formal Methods for Interactive Systems", which was held at the IMT School for Advanced Studies Lucca in May 2015 and delivered to four first year PhD students. The double aim of the course, teaching

[1] http://sysma.imtlucca.it/cognitive-framework-maude-hofm-2016/.
[2] http://sysma.imtlucca.it/cognitive-framework-csp-hofm-2016/.

formal methods and provide an approach for their application to interactive systems, was realised through the use of our practitioner-oriented formal notation and its translations in MAUDE and CSP.

After introducing the two translations but before introducing the tools, the students were asked three questions:

1. "In which of the two approaches did you find easier to get the model right?"
2. "Which of the two translations is more elegant?"
3. "In which of the two approaches the resultant behaviour is easier to guess?"

The PhD students unanimously answered "the rewriting logic approach" to Questions 1 and 3, and "the process algebra approach" to Question 2. It is interesting to notice that, in spite of finding the process algebra approach more difficult, the student unanimously agreed that it is more elegant. These answers, as well as further remarks and opinions that emerged in an open discussion that followed, are an indicator that students have a strong interest for solutions that are concise, elegant and abstract, and that they are happy to tackle challenging problems in order to look for elegant rather than easy solutions. Given the small number of students and the absence of research design we cannot draw empirical conclusions from the students' answers and remarks, although these appear to be in line with the results of previous research [16].

In terms of tools, from the perspective of a student learning formal methods, it is important to see simulation and model-checking results directly on the low-level semantic structures underlying high-level domain structures. This perspective is very different from that of a practitioner, who prefers tools that hide the formal semantic structures underlying domain structures. Moreover, in the case of students' perspective, the presentation of results must aim at highlighting relations between behaviour and semantics and using under-approximation [8], the capability to output only relevant states and/or events, as well as stimulating and developing their abstraction and problem solving skills.

MAUDE and PAT are somehow complementary in terms of presentation of results, also due to the different characteristics of the formal methods on which they are based. MAUDE does not support any form of graphical representation but supports a form of under-approximation, by filtering the output through additional rewrite rules, and allows the designer to easily track which rewrite rule is applied and check the content of all data structures, thus tracking the behaviour back to the architectural view of the designer. PAT facilitates the visual representations of the global behaviour in terms of finite state machines, but the form of under-approximation introduced by the CSP hiding operator is not very effective due to the possible introduction of nondeterminism, while the represented behaviour does not reflect the structure, in terms of concurrent components and synchronisations, from which the global behaviour has been attained. However, the use of both these tools in our course has allowed students to make use of all needed presentation features, visualisation from PAT, under-approximation and behaviour tracking from MAUDE. Moreover, in our class discussions, students showed the perception that the fact that the two tools are based on two distinct modelling paradigms contributed to stimulate and develop their abstraction and problem solving skills.

6 Conclusion and Future Work

We have discussed to which extent the modelling language developed in previous work [3] for modelling interactive systems may represent an appropriate compromise between the perspectives of an HCI practitioner (meets Objective 2 from Sect. 2) and a formal methods expert (can be translated into formal methods and undergo formal analysis). We noted that in order to accomplish Objective 2 from Sect. 2 it would be necessary to implement mechanisms to effectively present the results of simulation and model checking on the high-level cognitive and non cognitive components, for example through domain specific visualisations.

Instead, for students learning formal methods, the presentation of both the rewriting logic translation and the CSP translation and both respective tools, MAUDE and PAT, was perceived by the students themselves as beneficial for their abstraction and problem solving skills. In our future work, we plan to systematically investigate empirical evidence of such student perception.

References

1. Beckert, B., Grebing, S., Böhl, F.: A usability evaluation of interactive theorem provers using focus groups. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 3–19. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15201-1_1
2. Cerone, A.: Closure, attention activation in human automatic behaviour: A framework for the formal analysis of interactive systems. In: Proceedings of FMIS 2011, Electronic Communications of the EASST, vol. 45 (2011)
3. Cerone, A.: A cognitive framework based on rewriting logic for the analysis of interactive systems. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 287–303. Springer, Heidelberg (2016). doi:10.1007/978-3-319-41591-8_20
4. Cerone, A., Roggenbach, M., Schlingloff, B.-H., Schneider, G., Shaikh, S.: Teaching formal methods for software engineering – ten principles. Informatica Didactica **9** (2015). https://www.informaticadidactica.de/index.php?page=Schlinghoff2015
5. Cerone, A., Scotti, M.: Research challenges in modelling ecosystems. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 276–293. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15201-1_18
6. Dix, A., Finlay, J., Abowd, G., Beale, R.: Human-Computer Interaction. Pearson Education, Upper Saddle River (1998)
7. Glass, R.L.: A new answer to "how important is mathematics to the software practitioner?". IEEE Softw. **17**(6), 136–136 (2000)
8. Idani, A., Stouls, N.: When a formal model rhymes with a graphical notation. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 54–68. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15201-1_4
9. Ladenberger, L., Dobrikov, I., Leuschel, M.: An approach for creating domain specific visualisations of CSP models. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 20–35. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15201-1_2
10. Page, R.L.: Software in discrete mathematics. In: Proceedings of ICFP 2003, ACM SIGPLAN Notices, vol. 38, pp. 79–86. ACM (2003)
11. Sobel, A.E.K., Clarkson, M.R.: Formal methods application: an empirical tale of software development. IEEE Trans. Softw. Eng. **28**(3), 308–320 (2002)

12. Sobel, A.E.K., Clarkson, M.R.: Response on "Comments on 'Formal methods application: an empirical tale of software development"'. IEEE Trans. Softw. Eng. **29**(6), 572–575 (2003)
13. Thagard, P.: Cognitive science. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Stanford University (2008)
14. Wing, J.M.: Teaching mathematics to software engineers. In: Alagar, V.S., Nivat, M. (eds.) AMAST 1995. LNCS, vol. 936, pp. 18–40. Springer, Heidelberg (1995). doi:10.1007/3-540-60043-4_44
15. Wing, J.M.: Invited talk: weaving formal methods into the undergraduate computer science curriculum (Extended Abstract). In: Rus, T. (ed.) AMAST 2000. LNCS, vol. 1816, pp. 2–7. Springer, Heidelberg (2000). doi:10.1007/3-540-45499-3_2
16. Zamansky, A., Farchi, E.: Exploring the role of logic and formal methods in information systems education. In: Bianculli, D., Calinescu, R., Rumpe, B. (eds.) SEFM 2015. LNCS, vol. 9509, pp. 68–74. Springer, Heidelberg (2015). doi:10.1007/978-3-662-49224-6_7