Empirical Study: Comparing Hasselt with C# to describe multimodal dialogs

Fredy Cuenca, Jan Van den Bergh, Kris Luyten, Karin Coninx
Hasselt University - tUL - iMinds
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek
Belgium
Email: {fredy.cuencalucero,jan.vandenbergh,kris.luyten,karin.coninx}@uhasselt.be

***Abstract*—Previous research has proposed guidelines for creating domain-specific languages for modeling human-machine multimodal dialogs. One of these guidelines suggests the use of multiple levels of abstraction so that the descriptions of multimodal events can be separated from the human-machine dialog model. In line with this guideline, we implemented Hasselt, a domain-specific language that combines textual and visual models, each of them aiming at describing different aspects of the intended dialog system.**

We conducted a user study to measure whether the proposed language provides benefits over equivalent event-callback code. During the user study participants had to modify the Hasselt models and the equivalent C# code. The completion times obtained for C# were on average shorter, although the difference was not statiscally significant. Subjective responses were collected using standardized questionnaires and an interview, which both indicated that participants saw value in the proposed models. We provide possible explanations for the results and discuss some lessons learned regarding the design of the empirical study.

***Index Terms*—Multimodal systems, Human-machine dialog, Finite state machines, Dialog model, Domain-specific language.**

I. Introduction

Multimodal systems allow users to communicate through the coordinated use of multiple input modes, e.g. speech, gaze, and gestures. These systems have the potential to support a human-machine communication that is robust (e.g. multiple inputs can be combined to perform disambiguation), flexible (e.g. users can choose their preferred modality), and more natural than ever before.

However, implementing multimodal systems is still a difficult task. This is partly because of the complexity of multimodal interaction [1], [2], the absence of standardized methodology [2], and the mastering of different state of the art technologies required for their construction [3].

Several domain-specific languages have been proposed with the intention of simplifying the implementation of multimodal interfaces [3]–[9]. From an analysis of these languages, Dumas et al. [3] proposed many guidelines for developing future languages. One of these guidelines states that the specialized language must be such that the declaration of multimodal events can be separated from the description of the human-machine dialog (Figure 1). The present research has implemented this idea and measured how potential users can benefit from such an implementation.

Concretely, we created a language, called Hasselt, that allows notations for declaring multimodal events and human-machine dialogs separately. The multimodal events are textually declared as combinations of predefined user events (e.g. mouse clicks, speech inputs, etc.). The multimodal dialog is depicted as finite state machines (FSMs) whose arcs are labelled with multimodal event names.

In order to evaluate the benefits of such separation of concerns, a user study was conducted. Participants had to sequentially modify two equivalent implementations of a multimodal dialog system. In one case, both the code for handling the events and the code for handling the dialog were included in the same source file written in C#. In the other case, these were specified separately with the textual and visual notations provided by Hasselt.

II. Related work

A. Modeling multimodal dialogs as FSMs

When modeling human-machine dialogs as finite state machines (FSM), the nodes of the FSM represent the possible states of the dialog system, and its arcs represent the transitions in the dialog system's state. Many researchers have proposed FSM-based solutions for modeling unimodal human-machine dialogs, e.g. IOG [10], SwingStates [11], Schwarz's framework [12], and InterState[13], among others.

However, there are only few languages that allow modeling multimodal human-machine dialogs as FSMs. Some representative examples are listed in what follows.

We can consider MEngine [4] as a FSM-based language that allows modeling trivial multimodal dialogs, e.g. the system reponses are always the same for a given multimodal input. We can consider MEngine [4] as a FSM-based language that allows modeling trivial multimodal dialogs, e.g. the system reponses are always the same for a given multimodal input.

In NiMMiT [14], the dialog model is a state machine where each state represent a set of tasks that are available to the end user. NiMMiT is restricted to interactive virtual environments (IVE) since its presentation model has to be encoded in VRIXML [15]. In contrast, with our proposal, the presentation model can be implemented in any .NET language, which opens a wide assortment of possibilities beyond IVE.

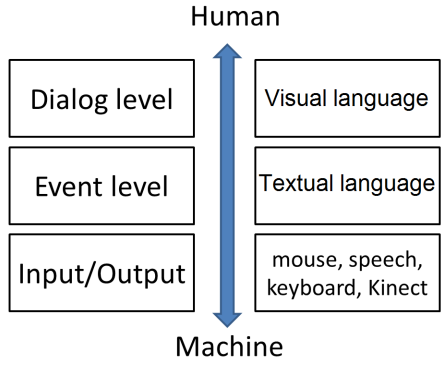


Fig. 1. On the left side of the diagram, one can see the different levels of abstraction proposed by Dumas et al. [3]. On the right side, one can see how our language follows the same framework: our visual language is at the dialog level whereas our textual notations are at the events level.

SMUIML [3] provides different notations for declaring the human-machine dialog and for combining user events. Unlike Hasselt, SMUIML does not include a symbol for defining iterative events. This reduces the space of multimodal events that can be specified with SMUIML in comparison with Hasselt. For instance, the *drag-and-drop*, which involves an arbitrary number of *mouse-move* events cannot be specified with SMUIML at the level of events. Another difference is that, unlike Hasselt, SMUIML does not support state variables or conditional transitions at the dialog level.

B. User studies. Interaction models vs. event-callback code

To the best of our knowledge, none of the abovementioned multimodal dialog modeling languages have been evaluated in user studies. Nonetheless, outside the multimodal domain, we found two user studies that guided us in the design of our experiments.

Oney et al. recruited 20 developers to evaluate the understandability of Interstate's visual notation. Each participant had to modify two systems (drag-and-drop and a thumbnail viewer) implemented in both RaphaelJS[1] and InterState. It was verified that InterState models are faster-to-modify than equivalent event-callback code written in RaphaelJS [13].

The creators of Proton++ carried out two experiments with 12 programmers. Each participant was shown a multitouch gesture specification and set of videos of a user performing gestures. Gestures may be specified as a regular expression, tablature, or with event-callback code and the participant had to match the specification with the video showing the described gesture. The results showed that the tablatures of Proton++ are easier-to-comprehend than equivalent regular expressions and event-callback code [16].

Since real-world scenarios require programmers not only to comprehend but to write programming code, we followed the schema of Oney et al [13]. We asked participants to perform modifications with our language and with equivalent event-callback code.

[1]http://raphaeljs.com/

III. HASSELT

Hasselt provides notations for creating executable specifications of multimodal human-machine dialogs. It comes with a complete User Interface Management System (UIMS) [17] that offers the editors, runtime environment, and debugging tools required to code, run, and test Hasselt specifications.

A. Running Example

In the remainder of the paper, we will show how to implement a simple multimodal dialog system with Hasselt. The front-end of our running example system is shown in Figure 3, BE. It allows end users to issue multimodal commands to create, move, and remove objects from a canvas that is initially empty. These commands may be enabled or disabled depending on the current context-of-use.

Users can create new objects by issuing voice commands like *'create green box here'* while clicking on the canvas to indicate the position of the new object. Boxes are reshuffled by issuing *'put that there'* while clicking on both the target object and its new position [18]. And the canvas can be cleared up in reaction to the voice command *'remove objects'*. To make the system responses depend on the context-of-use, we added two rules: The boxes can only be moved if there are more than three of them on the canvas; and the canvas can only be cleared up after the displacement of at least one object.

B. How to use Hasselt UIMS?

The steps required to create a multimodal dialog system with Hasselt UIMS are as follows.

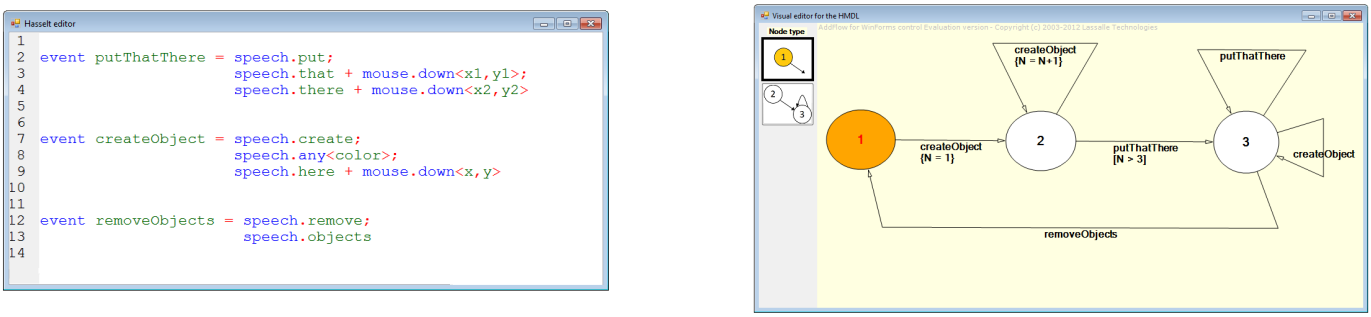
1) Implementing a back-end application: One must create an executable program implementing the front-end and the handling methods of the intended system. For the purpose of this work, such program will be referred to as back-end application. The back-end application can be implemented with any .NET programming language to be subsequently imported into Hasselt UIMS.

For the aforementioned running example, the back-end application implements the front-end shown in Figure 3, BE, and the methods for creating, moving, and removing virtual objects, i.e. CREATEOBJECT(COLOR,X,Y), PUTTHATTHERE(X1,Y1,X2,Y2), and REMOVEALLOBJECTS().

2) Declaring multimodal events: Hasselt allows combining multiple user events into one single abstraction [9], [19].

Programatically, user events can be combined through a set of event operators that can be used in a recursive manner. The operator $FOLLOWED\ BY(;)$ indicates sequentiality of events, the operator $OR(|)$ serves to specify alternative events, $AND(+)$ represents simultaneity of events, and $ITERATION(^*)$ is meant to specify repetitive events [9].

To describe the interactions to be supported by our running example system, we used these operators to declare the following multimodal events (Figure 2, a):



(a) (b)

Fig. 2. Hasselt UIMS during design time. (a) Textual editor for declaring multimodal events. It offers syntax highlighting, auto-completion popups, tooltip messages, and other features that facilitate the editing of code. (b) Visual editor for depicting human-machine dialogs. The arrows of the FSM are annotated with the multimodal events declared in (a). The arrows can include guard conditions.

$$event\ putThatThere = \begin{array}{l} speech.put\ ; \\ speech.that + mouse.down\langle x1, y1\rangle; \\ speech.there + mouse.down\langle x2, y2\rangle; \end{array} \quad (1)$$

$$event\ createObject = \begin{array}{l} speech.create\ ; \\ speech.any\langle color\rangle; \\ speech.here + mouse.down\langle x, y\rangle; \end{array} \quad (2)$$

$$event\ removeObjects = \begin{array}{l} speech.remove\ ; \\ speech.objects \end{array} \quad (3)$$

3) Binding multimodal events with event-handling callbacks: Each multimodal event must be bound to a method of the back-end application. At runtime, Hasselt UIMS will automatically launch these methods whenever their associated multimodal events are detected.

For our running example, one has to bind the method PUTTHATTHERE(X1,Y1,X2,Y2) to the event *putThatThere* shown in Equation 1. Similarly, methods CREATEOBJECT(COLOR,X,Y), and REMOVEALLOBJECTS() can be bound to the events declared in Equation 2 and Equation 3 respectively. The multimodal events do not have to have the same name as their associated callbacks.

We must highlight that the binding between multimodal events and callback functions is specified through a textual notation. With this notation, one can bind not only one but multiple callbacks to one single multimodal event, and to specify temporal and spatial constraints among the constituents of a multimodal event. The notations for binding multimodal events will not be presented herein and interested readers can refer to [19]. The focus of this paper is in the evaluation of the visual language that is used after the definition and binding of multimodal events.

4) Describing the human-machine dialog: The visual editor provided by Hasselt UIMS (Figure 2, b) enables programmers

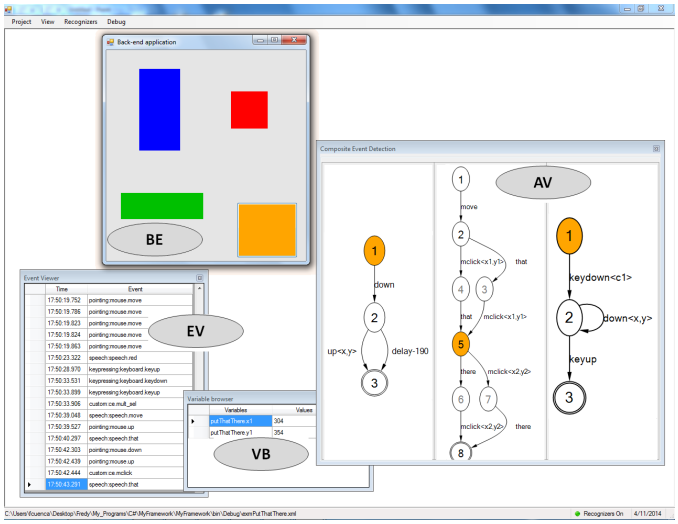


Fig. 3. Hasselt UIMS during runtime: The event viewer (EV) displays the user events as detected by the recognizers. The variable browser (VB) shows the event parameters values. The automata view (AV) presents animations showing the progressive detection of multimodal events. The back-end application (BE) the end user has to interact with was imported into Hasselt UIMS.

to describe human-machine dialogs as extended finite state machines [20], i.e. state machines augmented with state variables and guard conditions.

In a Hasselt visual model, the circles represent the potential states of the dialog system, and the arcs represent the system's state transitions. Each arc is annotated with a multimodal event whose occurrence causes the transition represented by the arc. Additionally, one can use state variables to encode quantitative aspects of the dialog, e.g. the number of times a state (transition) is visited (traversed). The statements required to maintain the state variables can be annotated in the arcs of the extended state machine. Finally, guard conditions can also be annotated in the arcs of a FSM to restrict their associated state transitions.

The visual model shown in Figure 2, b describes the dialog supported by our running example system. The circle labelled

as 1 represents the state where canvas is empty; the circle 2 represents the state where there is at least one object on the canvas; and the circle 3, the state where at least one object has been moved. The system moves from the initial state 1 to state 2 upon the creation of the first object. It also moves from state 2 to state 3 after the first displacement of an object. The variable N is used (a) to count the number of objects in the canvas –when this is relevant–, and (b) to condition the displacement of objects, which should only be possible if there are more than 3 objects on the canvas –notice the label $[N > 3]$. Finally, the removal of objects sets the system to its initial state: the circle labelled as 1.

It can be proved that if event-callback code were used to implement the running example system, the identification of the system's state would have required a series of nested if-else statements spread throughout a big portion of the whole program. Rather, Hasselt models have fewer and simpler conditional clauses that can be centralized in a FSM that provides a comprehensive overview of the human-machine dialog. There is one way to know whether these theoretical advantages are reflected into practical benefits for programmers, which is through a user study.

IV. User Study

The experiment aims at determining whether separating the declaration of events from the dialog model brings about benefits in favor of programmers.

A. Hypothesis

We hypothesize that the maintainance of a multimodal dialog can be performed faster and/or more easily with Hasselt, where the events can be described separately from the dialog model, than with C#, where the code for combining multimodal events is intermixed with the code for dialog management.

B. Method

1) Study Design: The participants were evaluated one by one after receiving a training session.

During the experiment, each participant was shown a multimodal system with which he had to interact according to the indications of the researcher. Once the participant was familiar with the functionality of the system, he was shown the source code/visual model of the system and asked to perform modifications in it. Each participant had to sequentially perform the changes in both Hasselt and C#. The changes to be performed were explained orally, but also written in a sheet that the participant can check during the experiment.

While the participant modifies the code/visual model, the researcher observes the changes made by the participant on a secondary monitor that replicates the screen in front of the participant. In this way, for each language, the researcher can measure the completion time of the task, count how many times the partial changes are tested in the runtime environment, and watch how the participant navigates trough the C# code or Hasselt visual model.

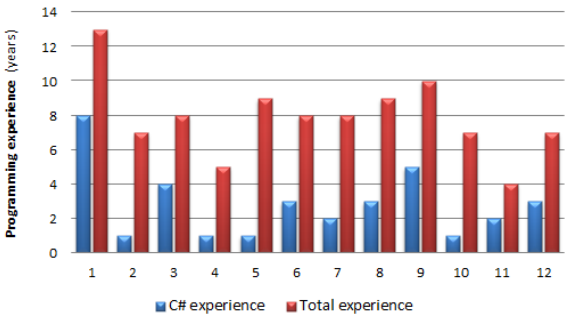


Fig. 4. Programming experience of the 12 participants

After the participant performs the requested changes with a language, he is asked to fill a post-task questionnaire. At the end of the whole experiment, i.e. after using Hasselt and C#, the participant is asked to evaluate the usability of Hasselt UIMS and interviewed by the researcher.

2) Participants: We recruited 12 participants, all of whom are male. The programming experience of the participants ranges from 4 to 13 years; their C# experience, between 1 and 8 years (Figure 4).

3) Procedure: Before the beginning of the experiment, each participant was given a 10-minutes tutorial about Hasselt. Participants had to describe a simple, Hello world-like multimodal interaction by following step-by-step instructions. The tutorial help participants get acquainted with the visual editor, debugging tools, and runtime environment of Hasselt UIMS.

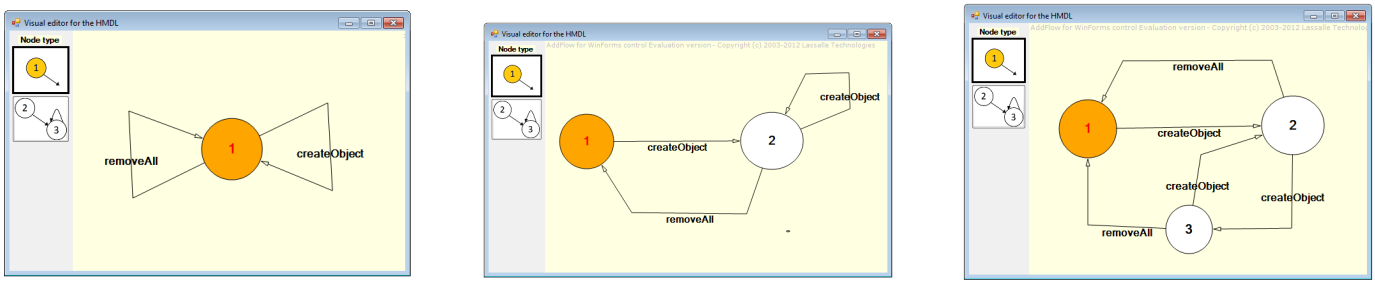
Since all participants had experience with C# and MS Visual Studio, there was no need for training in this respect.

For the experiment, the participant was presented with a system similar to the one herein used as a running example. It allowed users to create and remove virtual objects from a canvas in response to multimodal input. In the version given to participants, the objects could be created or removed at any time, after which the end user was acknowledged with voice feedback. Participants were asked to change the system so that it can handle two contexts-of-use: the command to remove objects must only be processed if there are objects on the screen; otherwise, it should be ignored.

The aforementioned system was described with both C# and Hasselt. Each participant had to modify both sources within a time limit of 30 minutes per language.

4) Solution of the modeling task: With Hasselt, the required changes can be made by modifying the human-machine dialog model only. Participants had to define different context-of-uses to distinguish whether the form is empty or has objects on it. Figure 5 shows two potential solutions.

As to the C# code, participants had to declare one variable for counting the number of objects on the form. This variable has to be updated every time a new object is created and whenever all the objects are removed from the form. It also has to be interrogated before proceeding to clear up the form. Although these four additions are easy to implement, they have to be included in the right place of a source code of 114 lines.



(a) Model given to participants (b) Most common solution (c) Outlier's solution

Fig. 5. The model shown in (a) was given to participants. Here the system is always in the same context-of-use and any interaction is available at any time. The model (b) was presented as a final solution by 11 participants. The model (c) was the final solution found by the outlier, who had no previous experience in FSM. Both types of solutions were correct.

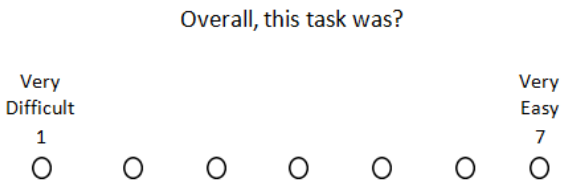


Fig. 6. Single Ease Question (SEQ) questonnaire.

(Actually, the full code contained 273 lines, but we hide the code for loading the speech recognizer, for hooking the mouse, and the back-end functions. This was to make the comparison as fair as possible. With Hasselt, the configuration code or the back-end code cannot be seen either. The former is within Hasselt UIMS; the latter, in a canned application imported into Hasselt UIMS.)

C. Measures

1) Observations: As the participant performs the required modifications with a certain language, the researcher monitors his working time and counts the number of times the code is tested.

2) Single Ease Question (SEQ) questionnaire: Right after completing the changes with each language, participants were asked to fill the Single Ease Question (SEQ) questionnaire, a 7-point rating scale (Figure 6) aimed to assess the perceived difficulty (or perceived ease, depending on one's perspective) of a task. The questionnaire has been proven to be reliable, sensitive, and valid while also being easy to respond and easy to score [21].

3) System Usability Scale (SUS) questionnaire: At the end of the experiment, participants had to fill the System Usability Scale (SUS) questionnaire [22] (Figure 7, a), which has become a well-known questionnaire for end-of-test subjective assessments of usability [23].

The SUS questionnaire consists of 10 items with 5-point scales numbered from 1 (anchored with "Strongly disagree") to 5 (anchored with "Strongly agree").

SUS test scores are normalized to values between 0 and 100. To have a benchmark to which one can compare SUS scores with, Lewis et al. shared historical information showing that the average and third quartile of 324 usability evaluations performed with SUS are 62.1 and 75.0 respectively [23].

Finally, according to a factor analysis performed by Lewis et al., the SUS questionnaire does not only measures usability. It also measures learnability, being $Q4$ and $Q10$ the questions that allow estimating the perceived learnability of the system under evaluation [23]. In the taxonomy proposed by Grossman et al. [24], this learnability falls within the category of initial learnability given that participants have been exposed to Hasselt for the first time during this experiment.

D. Interview Highlights

Based on the SEQ scores, a majority (7 out of 12 participants) considered that the modification of Hasselt visual models was easier than changing C# code. When asked for a reason, many of these participants referred to the overall view provided by the visual models: "You can see all the system in one screen" and "You do not have to browse code through multiple screens" were common answers.

One of the few participants who scored Hasselt as more difficult than C# was the outlier seen in Figure 8, a. He pointed out his total lack of knowledge in state machines as the cause of his poor performance. All other participants had, at least, pen-and-paper experience with state machines and thus, they could get more benefits from the training session.

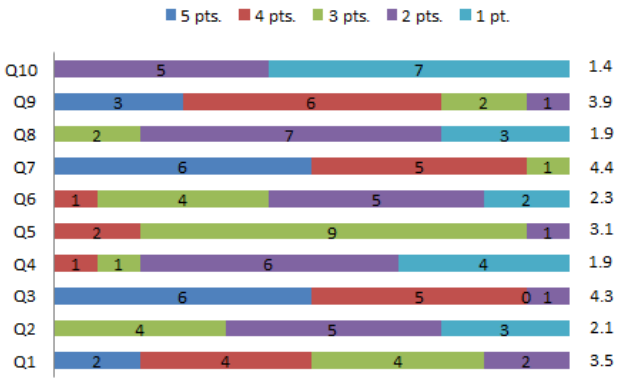
E. Results

All 12 participants could complete the changes with both languages Hasselt and C#. The data from observations and post-task questionnaires are synthesized in Figure 8. After inspecting the data, we decided to drop the only participant who had no previous experience with FSMs. He was an outlier in the plots (a) and (b) shown in Figure 8. Therefore, the following results are based on the remaining 11 participants.

1) Completion time: On average, changes made with Hasselt took 2.4 minutes in comparison with the 2.1 minutes when using C#. However, these results were not statistically significant. We could not reject the null hypothesis in favor

		strongly disagree 1	2	3	4	strongly agree 5
Q1	I think that I would like to use this system frequently	O	O	O	O	O
Q2	I found the system unnecessarily complex	O	O	O	O	O
Q3	I thought the system was easy to use	O	O	O	O	O
Q4	I think that I would need the support of a technical person to be able to use this system	O	O	O	O	O
Q5	I found the various functions in this system were well integrated	O	O	O	O	O
Q6	I thought there was too much inconsistency in this system	O	O	O	O	O
Q7	I would imagine that most people would learn to use this system very quickly	O	O	O	O	O
Q8	I found the system very cumbersome to use	O	O	O	O	O
Q9	I felt very confident using the system	O	O	O	O	O
Q10	I needed to learn a lot of things before I could get going with this system	O	O	O	O	O

(a) System Usability Scale (SUS) questionnaire



(b) Scores per question for Hasselt UIMS

Fig. 7. (a) SUS questionnaire that was filled by the 12 participants to evaluate Hasselt UIMS. (b) Participants' responses to the SUS questionnaire. Stacked barplots show the frequency of answers per question. The numbers at the right of the barplots indicate the average score per question.

of the alternative hypothesis that Hasselt completion times are higher than C# completion times: a Wilcoxon signed-rank test resulted in p-value = $0.1562 > 0.05$ (W = 12.5, Z = 1.3828).

2) Code testing effort: On average, programmers tested their code 1.2 times when using Hasselt and 1.4 times when using C#. But this result is not statistically significant either. We could not reject the null hypothesis in favor of the alternative hypothesis that the code testing effort is lower with Hasselt than with C#: a Wilcoxon signed-rank test resulted in p-value = 0.25 (W=0, Z = -1.4142).

3) Perceived ease of the task: The average SEQ scores for Hasselt and C# were 6.6 and 5.9 respectively. In this case, we found that this difference in favor of Hasselt was statistically significant. A Wilcoxon signed-rank test indicated that the alternative hypothesis that the SEQ scores are higher for Hasselt than for C# can be accepted (p-value = 0.0078, W=28, Z = 2.6153).

Note: The use of Wilcoxon signed-rank tests instead of paired t-tests responded to the fact that we could not guarantee the normality assumption required by the latter. The non-normality of the pair differences was observed in both normal Q-Q plots and Shapiro-Wilk normality tests. The data analysis was performed with the open source software R[2].

4) Results of the SUS questionnaire: The SUS questionnaire was only used to evaluate Hasselt UIMS. Comparing with the data repository provided by Lewis et al., the average SUS score of 73.96 that the participants gave to Hasselt UIMS indicates that its perceived usability is well above average but not higher than 75% of the 324 systems reported in [23].

The average scores obtained for Hasselt UIMS for each of the 10 items of the SUS questionnaires are observed in Figure 7, b.

F. Threats to validity

1) Construct validity: The general concept of validity was traditionally defined as the degree to which a test measures what it claims, or purports, to be measuring [25]. The construct validity of our empirical study coud have been affected as follows.

First, the code testing effort was quantified as the number of times the participant enters in the runtime environment. This means we assumed that participants have to run the program in order to test the correctness of the source code. This definition may not be complete since it ignores the effort made when the participant *'runs and tests the code inside his head'*.

Second, the SUS questionnaire may have been measured only certain aspects of the usability of Hasselt UIMS. An expert in empirical studies made us notice that usability also includes the long-term experience of using a software system, which is not considered in our study: all participants used Hasselt for the first and only time during the study. However, the initial learnability, which is another dimension of the SUS questionnaire, was correctly measured by $Q4$ and $Q10$, according to the same expert.

Construct validity is not the only type of validity that must be considered when designing empirical research. An empirical study is said to have internal validity when the impact of almost all influencing factors are excluded, so the study is performed in a highly controlled setting [26]. In contrast, external validity consists of allowing some influencing factors so that the experiment can emulate a real-world situation instead of an ideal one [26]. Whereas external validity increases the chances that results can be generalized to more realistic, everyday situations, internal validity allows researchers to pinpoint the reasons of improvement or degradation, but at the cost of generalizability.

2) Internal validity: We pursued for internal validity in the following way.

First, the order of the language to be used first (i.e. Hasselt or C#) was balanced over the participants so that the aggregated experience bias can be neutralized.

Besides, since the goal of the experiment was to measure the effort for describing multimodal dialogs, participants were

[2]https://www.r-project.org/

(a) (b) (c)

Fig. 8. Data collected from the 12 participants. (a) Completion times, (b) Number of times the code was tested. The plot whiskers are at the lowest datum still within 1.5 times the interquartile range (IQR) of the lower quartile, and the highest datum still within $1.5\times$ IQR of the upper quartile. (c) Barplots showing the frequency of each answer for the SEQ questionnaire.

restricted to this portion of the code/model only. With Hasselt, programmers were restricted to use the visual editor only. With C#, the code for configuring the speech recognizer and the application code (e.g. for creating, deleting objects) was hidden to programmers –we put this portion of the code in regions that were collapsed during the experiment.

On the other hand, offering participants a tutorial on Hasselt but no tutorial on creating multimodal dialogs using C# might affect the experiment's internal validity.

3) External validity: In order to confer our results with high external validity, we allow some 'freedom' to the experiments.

First, the pool of participants was quite varied. It includes master and PhD students, post-docs, and industry programmers, from different universities and countries, with and without background in finite state machines (FSMs).

Most importantly, participants were left free in the wild. This contrasts with other approaches commonly used in empirical studies, such as the *think-aloud* protocol and the *question-suggestion* protocol [24]. The former would require participants to speak out while programming in order to provide the researcher with insights about their programming logic. The latter would allow the researcher to give advice proactively to the participant. In our experiments, the researcher only interferes when participants ask for questions. In our opinion, this is a more realistic scenario that reflects the typical case of a programmer working by his own and eventually asking for advice to more expert programmers when he got stuck on a problem.

V. Discussion and Conclusion

A. Modeling with Hasselt and C#

We presented Hasselt, a language that provides notations for defining multimodal human-machine interaction dialogs. A dialog model in Hasselt is an extended finite state machine specified with a visual editor and whose arcs are annotated with multimodal events that are defined with a separate textual notation.

We expected to experience some benefits from separating the event definition code from the dialog management model. But this is what we found:

First, we found that the better-separated Hasselt models are not faster-to-modify than equivalent event-callback code where the instructions for event handling and for dialog management are intermixed. Although for our participants, the task of implementing a multimodal dialog was, on average, performed faster with C# than with Hasselt, these results were not statistically significant.

Second, our participants tested Hasselt models fewer times than equivalent C# code. Despite of this, completing changes with Hasselt took longer. Based on our observations, the reason for this may be that modifying visual models is more time-consuming than writing textual code.

Finally, the SEQ questionnaires revealed that participants perceived that performing the required changes with Hasselt was easier than with C#. Although these measurements turned out to be statistically significant, we cannot discard that some response bias played a role here. Participants gave higher scores to the language that led to longer completion times.

B. Perceived usability and initial learnability of Hasselt UIMS

Considering that odd-numbered questions are positively-worded, scores higher than 3 in these items reflect that participants agree (to a certain degree) that the evaluated system presents some good aspect/feature. In our study, all odd-numbered questions were scored with more than 3 points on average. From this group, $Q3$, i.e. "I thought the system was easy to use" and $Q7$, i.e. "I would imagine that most people would learn to use this system very quickly", received the highest scores.

Similarly, since even-numbered items are negatively-worded, scores lower than 3 would indicate that participants are disagreeing (to a certain degree) with some negative comment about the system. In our studies, all even-numbered questions were scored with less than 3 points on average. From this group, $Q10$, "I needed to learn a lot of things before I could get going with this system", $Q4$, i.e. "I think I would

need support of technical person to use this system", and $Q8$, i.e. "I found the system very cumbersome to use" received the lowest scores (which in this case it is something positive).

The salient scores obtained for $Q4$ and $Q10$, the two questions that define perceived initial learnability [23], indicate that, to a certain degree, participants consider that Hasselt UIMS is easy-to-learn.

C. Future work

We think that the main reason why no clear winner emerged from this study is that the task was too simple given the programming experience of the participants. Thus, we plan to repeat the experiment with more complex tasks.

Other minor changes refer to the functionalities of the visual editors. We want to minimize the effort involved in wiring the FSMs. We plan to add combination keys for creating nodes and links, not to allow resizing of the nodes, and allow jumping between the elements of a FSM with the TAB key.

Finally, we would like to gather objective cognitive load measurements [27] like heart rate or pupil dilatation. We expect to see some positive correlations between the perceived difficulty declared by participants in the questionnaires and their physiological reactions during the task.

D. Lessons learned

Based on this experience, we suggest some guidelines for others trying to design comparative studies between domain-specific languages and some mainstream language.

It is important that the training session can be supervised by the researcher and carried out right before the test. This makes all participants to start the experiment with a similar level of knowledge as long as they have similar backgrounds. Otherwise, some participants can benefit more from the training than others, which may cause the appearance of outliers.

It may not be a good idea to ask programmers working in the same research lab. Some may feel that one is going to evaluate their programming skills. From a research lab with more than 50 people, we could only recruit 5 participants. The remaining 7 participants were recruited from external institutions. Alternatively, one can also ask a person from an external institution to play the role of researcher so that participants do not feel observed by a acquaintance or colleague.

The complexity of the programming task must be appropriately calibrated. It has to be as high as to notice differences in the measurements; but not so high as to affect completion rates. In this matter, one must evaluate whether it is better to ask programmers to modify an existing program or to implement a new one from scratch.

References

[1] Y. A. Ameur and N. Kamel, "A generic formal specification of fusion of modalities in a multimodal hci," in *Building the Information Society*. Springer, 2004.

[2] W. Dargie, A. Strunk, M. Winkler, B. Mrohs, S. Thakar, and W. Enkelmann, "A model based approach for developing adaptive multimodal interactive systems." in *ICSOFT (PL/DPS/KE/MUSE)*, 2007, pp. 73–79.

[3] B. Dumas, D. Lalanne, and R. Ingold, "Description Languages for Multimodal Interaction: A Set of Guidelines and its Illustration with SMUIML," *Journal of multimodal user interfaces*, vol. 3, no. 3, pp. 237–247, 2010.

[4] M. Bourguet, "Designing and prototyping multimodal commands," in *Proceedings of INTERACT'03*, 2003, pp. 717–720.

[5] P. Dragicevic and J.-D. Fekete, "Support for input adaptability in the icon toolkit," in *Proceedings of the 6th ICMI'04*. New York, NY, USA: ACM, 2004, pp. 212–219. [Online]. Available: http://doi.acm.org/10.1145/1027933.1027969

[6] J. De Boeck, D. Vanacken, C. Raymaekers, and K. Coninx, "High level modeling of multimodal interaction techniques using NiMMiT," *Journal of Virtual Reality and Broadcasting*, vol. 4, no. 2, 2007.

[7] W. A. König, R. Rädle, and H. Reiterer, "Interactive design of multimodal user interfaces," *Journal on Multimodal User Interfaces*, vol. 3, no. 3, pp. 197–213, 2010.

[8] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq, "An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components," in *Proceedings of the EICS'09*. ACM, 2009, pp. 245–254.

[9] F. Cuenca, J. Van der Bergh, K. Luyten, and K. Coninx, "A domain-specific textual language for rapid prototyping of multimodal interactive systems," in *Proceedings of the 6th ACM SIGCHI symposium on Engineering interactive computing systems (EICS'14)*. ACM, 2014.

[10] D. A. Carr, "Specification of interface interaction objects," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1994, pp. 372–378.

[11] C. Appert and M. Beaudouin-Lafon, "Swingstates: Adding state machines to java and the swing toolkit," *Software: Practice and Experience*, vol. 38, no. 11, pp. 1149–1182, 2008.

[12] J. Schwarz, J. Mankoff, and S. Hudson, "Monte carlo methods for managing interactive state, action and feedback under uncertainty," in *Proceedings of the 24th annual ACM symposium on UIST*. ACM, 2011, pp. 235–244.

[13] S. Oney, B. Myers, and J. Brandt, "Interstate: Interaction-oriented language primitives for expressing gui behavior," in *Proc. of UIST'14*. ACM, 2014.

[14] J. De Boeck, C. Raymaekers, and K. Coninx, "A tool supporting model based user interface design in 3d virtual environments," in *Grapp 2008: proceedings of the third international conference on computer graphics theory and applications*, 2008, pp. 367–375.

[15] E. Cuppens, C. Raymaekers, and K. Coninx, "{VRIXML}: A user interface description language for virtual environments," 2004.

[16] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala, "Proton++: a customizable declarative multitouch framework," in *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST'12)*, 2012, pp. 477–486.

[17] M. Beaudouin-Lafon, "User interface management systems: Present and future," in *From object modelling to advanced visual communication*. Springer, 1994, pp. 197–223.

[18] R. Bolt, "Put-that-there: Voice and gesture at the graphics interface," in *Proceedings of the 7th annual conference on computer graphics and interactive techniques (SIGGRAPH' 80)*. ACM, 1980.

[19] F. Cuenca, J. Van der Bergh, K. Luyten, and K. Coninx, "Hasselt uims: a tool for describing multimodal interactions with composite events," in *Proceedings of EICS'15*, 2015.

[20] V. S. Alagar and K. Periyasamy, *Specification of software systems*. Springer Science & Business Media, 2011.

[21] J. Sauro and J. S. Dumas, "Comparison of three one-question, post-task usability questionnaires," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1599–1608.

[22] J. Brooke, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[23] J. R. Lewis and J. Sauro, "The factor structure of the system usability scale," in *Human Centered Design*. Springer, 2009, pp. 94–103.

[24] T. Grossman, G. Fitzmaurice, and R. Attar, "A survey of software learnability: metrics, methodologies and guidelines," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 649–658.

[25] J. D. Brown, *The elements of language curriculum: A systematic approach to program development.* ERIC, 1995.

[26] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in *Proceedings of the 37th International Conference on Software Engineering, ICSE 2015,(to appear)*, 2015.

[27] R. Brunken, J. L. Plass, and D. Leutner, "Direct measurement of cognitive load in multimedia learning," *Educational Psychologist*, vol. 38, no. 1, pp. 53–61, 2003.