Exploiting Semantic Search and Object-Oriented Programming to Ease Multimodal Interface Development

Septon, Thibaut; Villarreal Narvaez, Santiago; Devroey, Xavier; Dumas, Bruno

Published in:
EICS 2024 Companion - Companion of the 2024 ACM SIGCHI Symposium on Engineering Interactive Computing Systems

DOI:
10.1145/3660515.3664244

Publication date:
2024

Document Version
Peer reviewed version

Exploiting Semantic Search and Object-Oriented Programming to Ease Multimodal Interface Development

Thibaut Septon
thibaut.septon@unamur.be
Université de Namur
Namur Digital Institute
Namur, Belgium

Santiago Villarreal-Narvaez
santiago.villarreal@unamur.be
Université de Namur
Namur Digital Institute
Namur, Belgium

Xavier Devroey
xavier.devroey@unamur.be
Université de Namur
Namur Digital Institute
Namur, Belgium

Bruno Dumas
bruno.dumas@unamur.be
Université de Namur
Namur Digital Institute
Namur, Belgium

ABSTRACT

Multimodal interaction has been adopted across various platforms and devices, with supporting tools enhancing the developer experience in developing Multimodal Interfaces (MMI). While traditionally, these tools faced challenges balancing expressiveness and usability, recent progress in Natural Language Processing tends to mitigate this rule. However, adding multimodal interaction still remains challenging, especially when integrating the voice modality, and MMIs remain to be better integrated into today's applications. To address these challenges, we introduce a Unity tool-based system named Ummi. Ummi allows developers to use their knowledge in Object-Oriented Programming to handle the expert knowledge required to create a MMI while allowing end users great flexibility in the natural language they can use to interact. Our contributions are: 1) the presentation of Ummi's architecture and its inherent concepts along with its open-source implementation, and 2) a successful evaluation of its usability for describing MMIs through the System Usability Scale questionnaire with twelve participants.

CCS CONCEPTS

• **Human-centered computing** → **User interface toolkits**; • **Software and its engineering** → *Object oriented frameworks.*

KEYWORDS

Multimodal Interaction, Multimodal Interfaces, Fusion Engine Architecture

ACM Reference Format:
Thibaut Septon, Santiago Villarreal-Narvaez, Xavier Devroey, and Bruno Dumas. 2024. Exploiting Semantic Search and Object-Oriented Programming to Ease Multimodal Interface Development. In *Companion of the 16th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS Companion '24), June 24–28, 2024, Cagliari, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3660515.3664244

1 INTRODUCTION

Multimodal interaction tries to leverage human communication capabilities by using multiple modalities, such as speech, gestures, or facial expressions, to let users interact more naturally and efficiently with a machine [13, 21, 27]. Research has demonstrated numerous advantages of Multimodal Interfaces (MMI). These interfaces can stimulate human cognition, are less prone to user error, and adapt to different environments and diverse users. Additionally, integrating multiple modalities makes these interfaces more robust and accurate [13, 21]. MMI has been adopted through various systems, such as Extended Reality Headsets, Smartphones [19, 21], etc. Several tools have been developed to help designers and developers abstract the *fusion* of different modalities (i.e., extracting user intent through the various modalities). However, when designing multimodal interaction fusion engine tools, researchers need to find a trade-off between the tool's usability (easier for humans) and its expressiveness (easier for machines) [14]. While recent advances in Natural Language Processing (NLP) seem to bridge the gap between these two qualities when integrating the voice modality, it remains challenging to create MMIs [18, 23]. As the need for multimodal interaction expands, developers and designers face increasing difficulty in prototyping and developing MMIs.

We present Ummi[1], an open-source Unity tool tailored to create speech-centric multimodal interaction. Ummi allows developers to add multimodal commands easily into their applications using only basic Object-Oriented Programming (OO Programming) knowledge while using a state-of-the-art semantic search technique to conceal the expertise needed to make such interfaces. Moreover, it hugely reduces the work required to develop a MMI while allowing great flexibility in the end user's natural language. As the developer experience provided by software ultimately affects its adoption by developers, we also conducted a successful study with twelve participants to determine Ummi's usability score using the System Usability Scale (SUS) questionnaire [5].

Although research on *offline* multimodal fusion is actively pursued within the machine learning community, this work does not tackle the same challenges. Indeed, while *offline* multimodal fusion aims to analyze data, *online* multimodal fusion aims to enable human-machine communication. Thus, the considerations are not the same as the fusion needs to happen in a time and sometimes resource-constrained environment. Our contributions are the following: 1) Ummi, an early stage open-source multimodal engine toolkit realized for Unity that allows for usable, integrated, and expressive speech-centric MMI description, 2) A validation of Ummi's usability for declaring MMIs.

[1] https://github.com/tsepton/ummi

EICS Companion '24, June 24–28, 2024, Cagliari, Italy

Figure 1: Trade-off between usability and expressiveness for modeling MMI, adapted from [14].

2 BACKGROUND AND RELATED WORK

One of the first works to describe a MMI is the *put-that-there* application from Bolt [2] in 1980. In this pioneering work, users can interact with the system using a deictic language (i.e., where words have different meanings, depending on the current situation, e.g., *here* and *there*) and a pointing device. Since then, the field of Human-Computer Interaction (HCI) has long studied how multimodal interaction can be improved with key works such as the CARE properties [8], a framework to design multimodal systems, the ten myths of multimodal interaction [20], that deconstructs preconceived ideas about multimodal interactive systems, or the multimodal man-machine interaction loop [13]. However, despite the amount of work in the field, designing a MMI remains difficult as it demands a significant amount of work [23] and requires developers to handle the difficulty of multimodal fusion while maintaining control over the app's behavior [18]. Researchers have long worked to make the task easier, and as such, multiple tools have been created.

2.1 Fusion Engine Tools

There are a few specific examples that are noteworthy for the historical development of fusion engine tools: Bourguet proposes a graphical tool [4] named ***IMBuilder*** that uses a finite state machine to describe MMI and its attached fusion engine ***MEngine*** that dispatches events to the application, which proceeds to evaluate them. Cuenca et al. [9] use a similar approach with ***Hasselt UIMS*** but define their User Interface Description Language (UIDL), which permits the definition of atomic events (i.e., events coming from one modality) and the use of operators to compose them. Dumas et al. have developed ***HephaisTK*** [14], a graphical editor that uses SMUIML [11], an XML-based UIDL, to describe event-based MMIs and allow reacting to these events. The ***OpenInterface*** Framework [25] and its predecessor ***ICARE*** [3] created a component-based generic architecture, the idea being that components can be reused and linked together to prototype MMIs. Hoste et al. have proposed the ***Mudra*** Framework [17]. As previous multimodal architectures suffered from the wide variety of data highly linked to the different modalities, their solution allows for the centralization of any data representation from all modalities inside a centralized fact base, which serves for the fusion.

However, despite the growing number of tools to ease the development of MMI, these always offer a trade-off between their expressiveness and the usability they offer developers [14] (see Figure 1). This is especially true for the solutions that integrate the voice modality, as the developers have to choose between its expressiveness through verbosity (i.e., being exhaustive on the sentences the end users can employ), reducing the tool's usability, or keeping higher usability by reducing the modality expressiveness, thus restricting the language their end users can use, resulting in less natural communication.

In recent years, significant advancements in Large Language Model (LLM) and Natural Language Processing (NLP) solutions have greatly benefited multimodal interaction tools, as they allow to better understand users spoken language. These breakthroughs have made integrating and customizing the voice modality easier while providing better experiences for developers and users. One such tool is ***Geno*** [23], an IDE-based solution that uses NLP to integrate speech into web applications. It allows developers to add voice and GUI input in unimodal or multimodal interaction to existing web applications without requiring any knowledge of NLP. Geno asks developers to provide example utterances and label potential parameters within them. It then classifies users' utterances using an intent classifier model and executes the linked commands if the match is good enough. Another recent work by Li et al. is ***ReactGenie*** [18], a complete framework that aims to simplify the adoption of multimodal interaction within GUI applications. The framework delivers a programming experience comparable to current GUI frameworks, optimizes the utilization of pre-existing GUI code, and empowers developers to maintain complete control over the UI's appearance and app behavior through Object-Oriented (OO) state abstraction. While most of the older solutions previously cited [3, 4, 9, 11, 14, 25] were never widely adopted, partly because of the development workload imposed on their users, it remains to be seen whether Geno's or ReactGenie's approach will be successful.

As our objective is to enhance the accessibility of multimodal interaction for developers, we think that learning a dedicated framework (such as for ReactGenie [18]) or relying on a dedicated IDE (such as for Geno [23]) might not be optimal as it requires developers to learn these tools, and do not integrate within current development technologies, inducing increased context-switching resulting in an additional cognitive load [6, 7]. In reaction, we introduce Ummi, a Unity tool developers can integrate into their projects to build MMIs. A distinctive feature of Ummi lies in its compatibility with Unity's development workflow, offering a seamless integration experience. This approach contrasts with the one taken by ReactGenie [18], where the integration of React and Redux is replicated rather than complemented.

3 UMMI FUSION ENGINE

In this Section, we introduce Ummi, a toolkit for easier speech-centric MMI development for Unity. We first demonstrate how our tool works, then describe its architecture and how it integrates the voice modality, and finish with what this multimodal fusion approach offers.

3.1 Prototyping with Ummi

As Ummi is designed to be integrated within Unity, it uses the C# OO paradigm to abstract modalities' data and model the multimodal interaction. As such, to specify a MMI using Ummi, developers extend a "MonoBehaviour" extended class called "MMInterface" (see Listing 1 line 5), implement the "MonoBehaviour" inherited lifecycle method "Start" (see Listing 1 line 6) to reference user actions and reference the "MMInterface" to an imported prefab

```
1  using ummi.Runtime;
2  using Ummi.Runtime.Parser;
3  using UnityEngine;
4
5  public class MMInterfaceExample : MMInterface {
6    public override void Start() {
7      UserActions.Add(typeof(PutThatThereActions));
8    }
9
10   public static class PutThatThereActions {
11     [UserAction("Create a cube there")]
12     public static void CreateCubeThere(Vector3 there) {
13       GameObject go =
               GameObject.CreatePrimitive(PrimitiveType.Cube);
14       go.transform.position = there;
15     }
16
17     [UserAction("Put that there")]
18     public static void PutThatThere(GameObject that, Vector3
           there) {
19       that.transform.position = there;
20     }
21   }
22 }
```

Listing 1: Writing the typical *Put-that-there* using Ummi.

from the tool. If a developer wants to implement Bolt's put-that-there example [2], the code would be somewhat similar to Listing 1.

Ummi's approach to describing an interface is to write high-level actions that a user should be capable of doing. These actions are currently implemented using static methods regrouped inside a static class of an interface (see Listing 1 line 10). Developers specify what these methods need as parameters and write their bodies as usual. They then have to specify, using the "UserAction" property attribute, one or more typical sentences a user should be able to pronounce to invoke these actions (see Listing 1 lines 11 and 17). A single sentence should adequately enable Ummi to permit application users to employ similar sentences without requiring exhaustive details from the developer. As illustrated in Listing 1, the parameters of the methods are complementary to the example sentence and usually represent objects referenced with other modalities during the utterance of such a phrase.

3.2 Modalities Representation

Some previously developed tools allow the interaction to be designed directly by modeling events from the modalities. We believe this approach unnecessarily burdens developers, as they must consider all possible combinations of modalities to achieve a more natural language, trading usability for expressiveness. With Ummi, developers specify only the data needed to perform an action, leaving it up to the engine to use the best modality as they are often redundant [12]. Ummi employs OO Programming to create semantic representations of events from various modalities. As a result, in the put-that-there scenario (see Listing 1), developers do not need to specify all the modalities that could emit space-related data but instead indicate that they require space-related data. Ummi uses the architecture depicted in Figure 2 to use OO as an abstraction.

As Ummi needs to instantiate objects from modalities (Figure 2.1), the tool uses mediator scripts called modalities processors (Figure 2.2). Any added processor is in charge of interpreting a

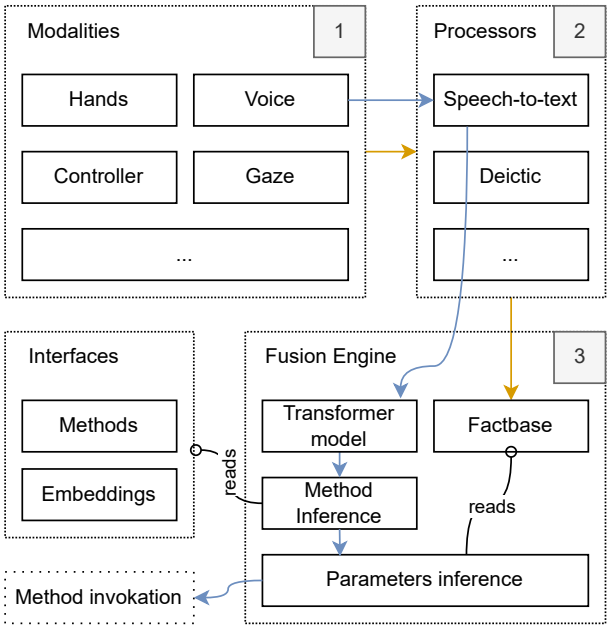

Figure 2: Ummi's simplified architecture: The blue arrow represents the data transformation flow from an utterance to its corresponding method, while the yellow arrow stands for the modalities processors mapping data into objects and writing them into the fact base.

modality by mapping its data into meaningful objects. For instance, matching a controller pointing in a direction to the instance of a class representing a three-dimensional ray. It should be noted that a processor can access multiple modalities at once to emit objects. Developers can either use the default provided processors or write their own, as it allows them to control how a modality should behave and integrate new modalities. To write a modality processor, developers can inherit the "*Processor*" abstract class, write the internal logic highly coupled to the modality and its input device, and call the "*WriteFact*" method to tell Ummi a modality event happened. An example processor for the mouse, provided by default with Ummi, is visible on Listing 2.

To centralize these objects, Ummi's MUDRA-inspired [17] architecture (see Figure 2) allows processors to append modalities instantiated objects to a data structure called the fact base. Its role is to centralize any processor-instantiated objects and to make them available to the Fusion Engine (Figure 2.3). The latter is the part of Ummi responsible for handling the multimodal fusion.

3.3 Speech-Centric Fusion

Ummi's fusion engine (Figure 2.3) can be decomposed into two different steps. The first is its ability to compare users' command utterances with the registered actions sentence examples (see Listing 1) and find the most appropriate method to be invoked. For the second, once the method has been found, the engine tries to fill in eventual parameters using simultaneous modalities events before invoking the method. If no method is found, Ummi stops the fusion.

```
using UnityEngine;

namespace ummi.Runtime.Processors {
  public class MouseProcessor : Processor {
    private void Update() {
      if (IsClicked()) OnClick(InputToRay());
    }

    private void OnClick(Ray ray) {
      if (Camera.main is null) return;
      int sequenceID = Time.frameCount;
      WriteFact(ray, sequenceID);
      if (Physics.Raycast(ray, out RaycastHit hit, 250)) {
        WriteFact(hit.collider.gameObject, sequenceID);
        WriteFact(hit.point, sequenceID);
      }
    }

    protected bool IsClicked() {
      return
          Input.GetMouseButtonDown((int)MouseButton.LeftClick);
    }

    protected Ray InputToRay() {
      return
          Camera.main!.ScreenPointToRay(Input.mousePosition);
    }
  }
}
```

Listing 2: An example implementation for a mouse processor, mapping clicks to Vector3, Ray and GameObject instances.

3.3.1 Finding the Most Appropriate Action. To compare user utterances with the actions provided by the developers (see Listing 1), Ummi uses a semantic search technique. With recent research in NLP, we saw the development of the BERT architecture [10]. Such models can compare sentences but are slow, as two sentences need to be forwarded simultaneously inside the model to be compared. To counter this, Reimers and Gurevych's [22] Sentence-Bert architecture maps each input (i.e., sentence) into a vector space where semantically similar sentences are close to one another. For Ummi's use case, this translates into transforming the actions sentences provided by the developers into meaningful sentence representations (i.e., vectors) when the application using Ummi starts using a pre-trained Sentence-Bert model. Then, when an end user wants to perform an action with the system at runtime, they can formulate their order appropriately using a deictic speech. Their utterance is then transformed into a sentence representation (i.e., a vector) using the same model as for the developer's provided sentences. Ummi can then perform the semantic search to retrieve the appropriate user action by computing the cosine similarities of the user sentence representation with the developer's provided sentence representations. This approach allows Ummi to find the corresponding closest method with no latency.

3.3.2 Completing Method Parameters. Once Ummi has found the method to be invoked, it must first check if it has any parameters to complete. If none, the method gets invoked. Otherwise, the engine has to infer the method parameters based on their type. This step reads the fact base to get previously instantiated objects from modalities processors. The current implementation uses a meaning-frame-based algorithm [28] that searches for the right objects by first keeping only the ones emitted during the timespan that covers the emission of the action by the user. The algorithm takes the first object in the fact base for each required parameter, which respects the required type. When iterating multiple times, any previously chosen object gets removed from the list of potential completion objects. This type-based fusion permits any type to be used within a MMI method signature as long as a modality processor emits it.

3.4 Desired Outcomes and Advantages

Using this semantic search approach (described in Section 3.3.1) for the fusion minimizes the required development time by relieving developers of the need to provide an exhaustive list of all permitted voice commands. This permits keeping a high expressiveness of the voice modality (i.e., offering end users significant flexibility in the natural language they can use) while keeping the tool's required verbosity at a minimum low, thus not reducing its usability.

Another advantage offered by Ummi's architecture is its ability to deconstruct MMI prototyping into two steps: the processing of modalities (e.g., Listing 2) and the MMI description (e.g., Listing 1). Such an approach permits leveraging the difficulty of writing MMIs and conceals the knowledge needed behind basic OO Programming concepts. We believe that modeling MMI through OO Programming may ultimately benefit programmers, as it abstracts advanced concepts related to the definition of MMIs under a well-known paradigm, clearly separating the processing of modalities and its inherent difficulties from the description of MMIs.

Decoupling the modalities from the MMIs description and its OO data representation has other benefits. First, creating new types to be handled by the Fusion Engine with no extra work on the fusion side is possible. Second, adding a new modality into Ummi only requires developers to write a new modality processor script to map its raw data into meaningful objects and has no impact over the already implemented MMIs. If the new modality emits objects that can be used inside this MMI, it will be taken into account automatically.

4 EVALUATION

This section describes the evaluation to assess Ummi's usability for describing speech-centric MMIs using the SUS questionnaire [5]. Participants are detailed, along with the evaluation itself. We then give and develop the results obtained and discuss potential threats to validity. A repository serving as part of this evaluation replication package is available online[2] and contains the data collected as well as the evaluation setup.

4.1 Participants

Twelve participants ($N = 12$) voluntarily took part in the study and did not receive any compensation. Eleven were between 20 and 25 years old, and one was between 26 and 30. All participants were enrolled in a computer science Master's degree, and the experiment occurred within the context of a HCI course. The experiment was not part of their course quotation. To understand the participants' existing knowledge concerning MMI and their familiarity with software engineering technologies, they were asked to rate their familiarity level on a 5-point Likert scale ranging from "No notion" to "Expert": $q1$) "How familiar are you with multimodal interfaces?", $q2$) "How would you define your own experience in

[2] https://github.com/tsepton/ummi_usability

multimodal interface development?", *q*3) "How familiar are you with Object-Oriented programming?", *q*4) "How familiar are you with strongly typed languages?", *q*5) "How familiar are you with the C# language?". Results show 67% were *novice* familiar with multimodal interfaces, and 50% had no prior experience. On the other hand, 50% of the participants felt proficient in object-oriented programming and strongly typed languages. When it came to the C# language, an equal number of participants (25%) were intermediates, proficients, or had no notion.

4.2 Setup

The experiment occurred across two sessions, with a one-week gap between them. The first session lasted for 40 minutes. It was dedicated to providing an overview of the challenges associated with designing MMIs and introduced Ummi's approach using the "put-that-there" example (same as seen in Listing 1). The session ended with the participants installing the necessary dependencies for the experiment on their computers. The second session took place the next week and lasted approximately 60 minutes. It was dedicated to the experiment itself. We began with a quick reminder of the content seen during the first session. Participants were given access to the experiment folder of the repository given above for the trial. All participants used Visual Studio Code, with the official C# Dev Kit extension, to get feedback from the IDE. The choice was not to use Unity with Ummi's real implementation for multiple reasons, all of which are discussed inside Section 4.4. Therefore, a mockup of Ummi was designed specifically for the experiment[3].

Participants were responsible for describing a MMI for an online sales Augmented Reality application. Their solution aims to visualize decorative items and other furniture directly at the user's location. The participants were then assigned the task of implementing an interface using Ummi (omitting the methods' body as they are not related to the declaration of the MMI and as they require Unity-related knowledge), that would allow users to perform the following actions: Display a menu to list the available items for sale, hide it, sort the items within the menu by name or by price, add a menu item to a given position inside the environment, invert the position of two environment items, move an environment item from one place to another, and delete an item from the environment. Finally, they were told that processors would be implemented by someone else. Still, the expected types they had at their disposal were the following: "Sort, "SortByPrice" and "SortByName" which extend "Sort", "Item" which represents any piece of furniture, and "Vector3" which represents a point in space.

Participants then worked alone and had a maximum of 60 minutes to perform the task. They could run Ummi's mockup implementation to get an output on what user actions were registered. Once they felt they had succeeded, they asked the experiment conductor for confirmation, who checked that the interface was correctly written and that all methods were correctly registered. Participants were then asked to fill in an online anonymous form containing the SUS questionnaire [5]. The SUS questionnaire was selected as it applies to a wide range of systems [24], was designed to be simple [5] and has proven to get reliable results even when applied to a small sample of users (8-12) [26]. An open field was also included

[3]Note that the mockup implementation is available inside the replication package.

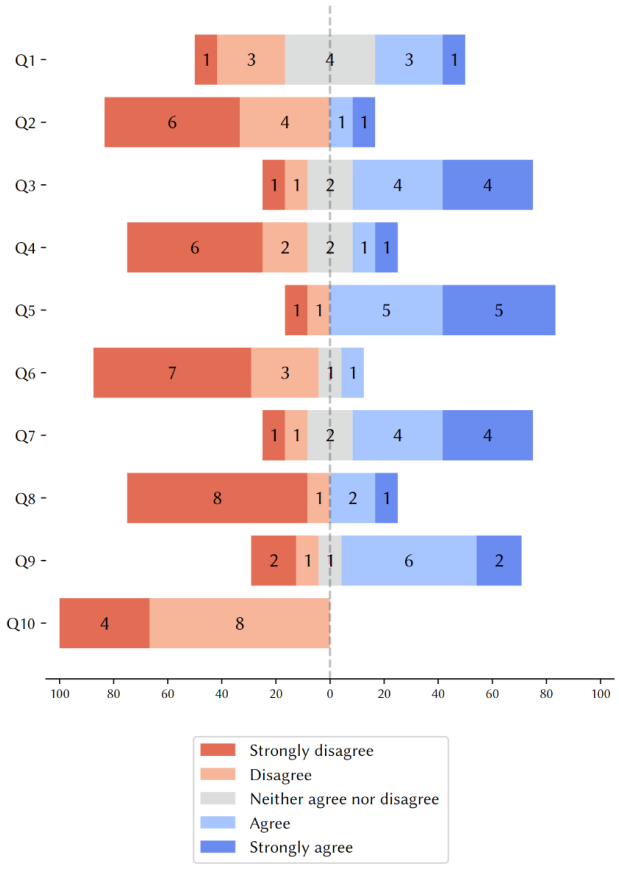

Figure 3: SUS questionnaire results.

to record the participants' general feelings about the experience. If the task were not correctly fulfilled, the experiment conductor would tell the participants something was missing and that they should reexamine the given documentation.

4.3 Results

All participants completed the task within 50 minutes, with six finishing in under 30 minutes. The experiment gave an overall SUS score of 71.67 for describing multimodal interfaces through Ummi. Figure 3 shows the percentage of responses for each SUS question. A common way to interpret SUS scores is to compare them to others SUS scores using percentiles according to Sauro [24]. Any SUS score is above average if > 68, and with an overall SUS score of 71.67 ($\sigma = 23.19$), Ummi is in the 60–64 percentile rank and is considered *good* result according to Bangor et al. [1] adjectives scale. The overall SUS score confirms the hypothesis that Ummi offers good developer usability for integrating speech-centric multimodal commands. Specifically, all participants except two expressed a positive feeling towards Ummi and an above average SUS score (except for *P*6, *P*8 and *P*9 with SUS scores of 65 (Fair), 20 (Awful) and 32.5 (Poor) respectively).

While checking if the different participants correctly realized the task, the experiment conductor took note of the problems that

occurred to the participants. First, one participant did not understand the idea behind the "UserAction" annotation, as they filled in the required string argument with the method's name. Also, six participants did not successfully register some actions, as they used types not emitted by the processors detailed inside the instructions. While they understood that the methods required arguments typed as objects emitted by the processors, all of them tried to use an "*ICollection<T>*" class from C# and used an accepted type as the generic argument (e.g., "*List<Item>*").

Results show that Ummi scored well for all questions except the first one, where results are more mitigated: 4 participants did not feel they would use the system frequently, while 4 neither disagreed nor agreed. This may be explained by the lack of multimodal interaction integration within current frameworks and libraries for desktop and mobile applications. Otherwise, Ummi performs well, with most participants (83%) finding the tool well integrated without inconsistencies or added complexity. Also, 67% agreed that the tool would be quick to learn to use, even though all participants agreed that there are not a lot of things to learn to start writing MMIs with Ummi.

4.4 Threats to Validity

One primary concern arises from employing students who are not accustomed to MMI development and are not representative of more advanced Unity developers. Nonetheless, utilizing students enabled us to establish a participant sample with consistent knowledge around using and creating MMI, allowing for meaningful comparisons. A second threat to validity may be related to the setup of the development environment. However, the decision to use a mockup environment for Ummi that did not rely on Unity is well-considered. Indeed, as participants may not have known Unity, and as the experiment was done to evaluate Ummi's ability to describe MMIs, discovering Unity for certain participants would have affected their SUS score. Also, as Ummi is still a work in progress, and its performances are still to be evaluated and improved, allowing participants to trial and error would not have been an appropriate solution, as a bad efficiency would inevitably have impacted the usability evaluation. Finally, allowing more time for participants and removing the presence of the experiment conductor could affect the experiment outcomes.

5 LIMITATION AND FUTURE WORK

First, while this work defined Ummi's way of declaring a MMI and focused on evaluating its usability, it should be noted that it is currently a prototype. In the near future, we will improve and evaluate its expressiveness through OO modeling. Another future work concerns the tool's efficiency in realizing the correct user action. As the fusion takes place in two stages (see Section 3.3), we first aim to evaluate and improve the performance of the semantic search before delving more into the parameters' completion algorithm. As for now, the frame-based algorithm used to complete methods parameters (see Section 3.3.2) handles the fusion badly when multiple objects correspond to the required parameter type inside the fact base. Using another algorithm, such as Rete [16] as it was proven to be efficient by Mudra [17], should overcome the problem. Also, while the evaluation shows successful results, it should be noted that the SUS questionnaire does not evaluate the resulting interface, but only the usability of Ummi. The user interfaces performance created with the tool is left to be evaluated.

Another future work is to be realized on the inner mechanism behind the fusion. Indeed, a current limitation of Ummi is that it is speech-centric: developers are only allowed to declare user actions that are invokable using the speech modality. However, a promising avenue for future research involves delving into multimodal machine learning, specifically exploring how data from other modalities can be transformed to be compared to text, as was realized by Fink et al. [15] for instance, which would allow other modalities to be used to find declared user actions. Also, while we are currently working on evaluating its efficiency and dedicated to improving it, this semantic search approach does not let end-users compose actions together and thus restricts the natural language they can use to interact with Ummi based applications.

Finally, while the Sentence-Bert models allow for quicker comparison of sentences at runtime than with a Bert architecture, this does come at the price of performance loss. To overcome this, performing an extra training step known as fine-tuning is common. This lets the model gain extra knowledge for domain-specific tasks and improves its ability to embed similar sentences closer to one another. However, this step requires data, takes time, and is computationally intensive. While Ummi allows to easily change the model used, it may not be possible to fine-tune the model in use, and thus, the intended user experience may suffer from it. We are fully aware of this and are exploring novel ways that do not require fine-tuning the model with each new version of a MMI.

6 CONCLUSION

Multimodal Interfaces (MMI) are becoming increasingly popular among users of different systems. As a result, developers and designers are dedicating significant efforts to designing and developing such interfaces [23], partly because it requires them to handle multimodal fusion while maintaining control over the app's behavior [18]. In this regard, we presented an open-source Unity tool called Ummi and its approach for easily creating speech-centric MMIs. Figure 2 simplifies how the modalities, processors, multimodal fusion, and MMIs fit together. Finally, we conducted a study to validate Ummi's ability to describe MMI, and the results were positive. All twelve participants successfully performed the given task, with a mean SUS score of 71.67. 75% of the participants gave an above-average SUS score (> 68), while 83.34% had an overall good opinion of the tool.

ACKNOWLEDGMENTS

The authors of this paper would like to express their gratitude to the participants of the usability studies. Likewise, we are very grateful to the anonymous reviewers, whose suggestions helped improve and clarify this manuscript. Authors are supported by the OPTIMIS project by Pôle MecaTech under Grant no. 8564.

REFERENCES

[1] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.

[2] Richard A. Bolt. 1980. "Put-that-there": Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph.* 14, 3, 262–270. https://doi.org/10.1145/965105.807503

[3] Jullien Bouchet, Laurence Nigay, and Thierry Ganille. 2004. ICARE software components for rapidly developing multimodal interfaces. In *Proceedings of the 6th international conference on Multimodal interfaces*. 251–258.

[4] Marie-Luce Bourguet. 2002. A toolkit for creating and testing multimodal interface designs. *companion proceedings of UIST* 2 (2002), 29–30.

[5] John Brooke. 1996. Sus: a "quick and dirty'usability. *Usability evaluation in industry* 189, 3 (1996), 189–194.

[6] Paul Chandler and John Sweller. 1992. The Split-Attention Effect as a Factor in the Design of Instruction. *British Journal of Educational Psychology* 62, 2 (1992), 233–246. https://doi.org/10.1111/j.2044-8279.1992.tb01017.x

[7] G. Convertino, J. Chen, Y. Ryu, C. North, and B. Yost. 2003. Exploring Context Switching and Cognition in Dual-View Coordinated Visualizations. In *International Conference on Coordinated and Multiple Views in Exploratory Visualization*. IEEE, 55. https://doi.org/10.1109/CMV.2003.1215003

[8] Joëlle Coutaz, Laurence Nigay, Daniel Salber, Ann Blandford, Jon May, and Richard M Young. 1995. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. *Human—Computer Interaction: Interact'95* (1995), 115–120.

[9] Fredy Cuenca, Jan Van den Bergh, Kris Luyten, and Karin Coninx. 2015. Hasselt uims: a tool for describing multimodal interactions with composite events. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 226–229.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[11] Bruno Dumas, Denis Lalanne, and Rolf Ingold. 2008. Prototyping multimodal interfaces with the SMUIML modeling language. In *CHI 2008 Workshop on User Interface Description Languages for Next Generation User Interfaces, CHI*.

[12] Bruno Dumas, Denis Lalanne, and Rolf Ingold. 2010. Description languages for multimodal interaction: a set of guidelines and its illustration with SMUIML. *Journal on multimodal user interfaces* 3 (2010), 237–247.

[13] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. 2009. *Multimodal Interfaces: A Survey of Principles, Models and Frameworks*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–26. https://doi.org/10.1007/978-3-642-00437-7_1

[14] Bruno Dumas, Beat Signer, and Denis Lalanne. 2011. A graphical uidl editor for multimodal interaction design based on smuiml. (2011).

[15] Jerome Fink, Pierre Poitier, Maxime André, Loup Meurice, Benoît Frénay, Anthony Cleve, Bruno Dumas, and Laurence Meurant. 2023. Sign Language to Text Dictionary with Lightweight Transformer Models. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023): AI for Social Good track*.

[16] Charles L Forgy. 1989. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases*. Elsevier, 547–559.

[17] Lode Hoste, Bruno Dumas, and Beat Signer. 2011. Mudra: a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces*. 97–104.

[18] Karina Li, Daniel Wan Rosli, Shuning Zhang, Yuhan Zhang, Monica S Lam, James A Landay, et al. 2023. ReactGenie: An Object-Oriented State Abstraction for Complex Multimodal Interactions Using Large Language Models. *arXiv preprint arXiv:2306.09649* (2023).

[19] Vivian Genaro Motti. 2020. *Wearable Interaction*. Springer International Publishing. https://doi.org/10.1007/978-3-030-27111-4

[20] Sharon Oviatt. 1999. Ten myths of multimodal interaction. *Commun. ACM* 42, 11 (1999), 74–81.

[21] Sharon Oviatt. 2022. *Multimodal Interaction, Interfaces, and Analytics*. Springer International Publishing, Cham, 1–29. https://doi.org/10.1007/978-3-319-27648-9_22-1

[22] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[23] Ritam Jyoti Sarmah, Yunpeng Ding, Di Wang, Cheuk Yin Phipson Lee, Toby Jia-Jun Li, and Xiang'Anthony' Chen. 2020. Geno: A Developer Tool for Authoring Multimodal Interaction on Existing Web Applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 1169–1181.

[24] Jeff Sauro. 2011. *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LLC.

[25] Marcos Serrano, Laurence Nigay, Jean-Yves L Lawson, Andrew Ramsay, Roderick Murray-Smith, and Sebastian Denef. 2008. The openinterface framework: A tool for multimodal interaction. In *CHI'08 Extended abstracts on human factors in computing systems*. 3501–3506.

[26] Jacqueline N Stetson and Thomas S Tullis. 2004. A comparison of questionnaires for assessing website usability. *UPA Presentation* (2004).

[27] Matthew Turk. 2014. Multimodal interaction: A review. *Pattern recognition letters* 36 (2014), 189–195.

[28] Minh Tue Vo and Cindy Wood. 1996. Building an application framework for speech and pen input integration in multimodal learning interfaces. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, Vol. 6. IEEE, 3545–3548.

Received February 16, 2024; accepted May 6, 2024