

# Towards a DSL to Formalize Multimodal Requirements

Marcos Gomez-Vazquez

*Luxembourg Institute of Science and Technology*  
Esch-sur-Alzette, Luxembourg  
0000-0001-7176-0793

Jordi Cabot

*Luxembourg Institute of Science and Technology*  
*University of Luxembourg*  
Esch-sur-Alzette, Luxembourg  
0000-0003-2418-2489

**Abstract**—Multimodal systems, which process multiple input types such as text, audio, and images, are becoming increasingly prevalent in software systems, enabled by the huge advancements in Machine Learning. This triggers the need to easily define the requirements linked to these new types of user interactions, potentially involving more than one modality at the same time. This remains an open challenge due to the lack of languages and methods adapted to the diverse nature of multimodal interactions, with the risk of implementing AI-enhanced systems that do not properly satisfy the user needs.

In this sense, this paper presents MERLAN, a Domain-Specific Language (DSL) to specify the requirements for these new types of multimodal interfaces. We present the metamodel for such language together with a textual syntax implemented as an ANTLR grammar. A prototype tool enabling requirements engineers to write such requirements and automatically generate a possible implementation of a system compliant with them on top of an agentic framework is also provided.

**Index Terms**—Domain-Specific Languages, Requirements Engineering, Multimodal User Interfaces, Agents

## I. INTRODUCTION

With the explosion of Machine Learning and other AI techniques, software systems are quickly adopting new types of complex user interfaces that require processing new input modalities such as text, audio and images. Sometimes, more than one type at the same time. This type of interfaces are known as Multimodal User Interfaces (MUIs). MUIs provide an interface that bears the functionalities of human-human interface [1]. It is a broad field within Human-Computer Interaction (HCI) with different applications and definitions depending on the context. The term “modality” refers to the mode of communication used as input and output between the human and the computer, like text, audio and image, although other modalities are considered in other contexts (e.g., facial expression, gestures, etc.).

While building this type of AI-enhanced systems is becoming easier thanks to the constant influx of, for instance, new multimodal Large Language Models (LLMs) that facilitate the analysis of multimodal inputs, validating that the system is satisfying the actual needs of the user is becoming more and more complex. Indeed, we are missing proper requirements engineering languages and techniques to facilitate a precise

specification of the MUI conditions that should trigger a system response, the data (“entities” in the MUI terminology) from the multimodal input that should be collected to provide an adequate response and the actual response [2].

For the latter, current languages could be reused, but a new way to specify MUI-driven requirements is needed for the first two elements. Preliminary work in this area has focused on requirements for chatbots. In particular, to identify what user “intents” (i.e. expressions of users’ intentions or goals) the chatbot should respond to, and what entities or parameters are part of the user textual input matching to a specific intent [3]. However, this is not the case with other modalities (e.g., image or audio) and even less for requirements involving multiple inputs.

To solve this problem, this paper presents MERLAN: a Multimodal Environment Requirements Language<sup>1</sup>. MERLAN aims to standardize the definition of multimodal requirements, independently of the underlying technology being used to process multimodal data. MERLAN is a Domain-Specific Language (DSL) formalized with a language metamodel and supported by a textual concrete syntax to help the users easily define MUI requirements. Tool support is also available. It consists of the language support but also a proof-of-concept implementation of a transformation from the requirements to a Python-based agent able to trigger system responses when the requirements conditions are met.

The rest of the paper is structured as follows: Section II introduces a running example to motivate and illustrate our DSL; Section III presents the abstract syntax of the DSL while Section IV focuses on its concrete syntax; Section V shows the provided tool support; Section VI discusses the related work of this paper; The research roadmap is discussed in Section VII and, finally, Section VIII closes with the conclusions.

## II. RUNNING EXAMPLE

To motivate and illustrate our approach, this section introduces a running example that will be used throughout the paper.

Let’s imagine that we need to specify the requirements for a new home automation and security system, hereinafter referred to as house agent. The house agent has some input devices to

This project is supported by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

<sup>1</sup><https://github.com/BESSER-PEARL/merlan>

capture text, sound, video, temperature, light and movement. It also has output devices for text and audio, in addition to the capability of executing a set of predefined actions (i.e., actions such as making a call or triggering the alarm).

Regardless of the actual software components in charge of capturing information from the physical world (e.g., ML models to detect objects from camera inputs), the house agent needs to be provided with clear definitions of the entities that can be recognized from all the different input devices, and the rules that describe how those entities should be evaluated by the agent in order to decide whether to trigger a certain response.

For instance, regarding the image input devices, we could define concrete entities such as `person`, `dog`, `car`, `smoke` and `fire`. An audio entity could be `strong_sound`. Moreover, these entities can have attributes. The `person` entity could have `gender` and `ethnicity` attributes, which during the recognition process of the agent, should be filled with the right values. The system needs other kind of more abstract entities to be defined, such as “day or night” or “empty house”. They are more abstract in the sense that they must be inferred from parts of the input data instead of being directly identified as objects in it.

With all these entities defined, we can design the rules the agent will check to trigger some actions. For instance:

- 1) If smoke is detected: notify the house owner.
- 2) If (fire is detected) or (the house is empty and (some cars or some persons are detected)): Activate the alarm, notify the house owner and call the police.
- 3) If (strong sound) and (during night): turn on lights
- 4) If a car with unrecognized plate number is detected: notify the house owner

The first rule simply defines the presence of a concrete entity (smoke). The second consists of the composition of multiple concrete and abstract requirements with “and” and “or” operators. The third expects the detection of an audio concrete entity (strong sound) and an image abstract entity (night). The last requirement is satisfied when not only an entity (car) but also a specific attribute value (the license plate) are identified.

This set of rules and entities would compose the requirements of the house agent’s multimodal interface, describing what conditions and based on what data the system should react to, and the actions to be taken in case of requirements satisfaction. These requirements could be described using natural language, but without enough precision to implement them with confidence.

Therefore, we believe a DSL to enable the formal definition of MUI requirements in a more precise way is needed. The next section introduces MERLAN, the DSL we propose for this.

### III. DSL DESIGN

A Domain-Specific Language (DSL) is a specialized programming or modeling language designed to address problems within a specific domain. Unlike general-purpose languages,

DSLs provide higher-level abstractions tailored to the concepts and rules of the domain, making it easier to express models, transformations, and constraints [4]. A DSL is defined by an abstract syntax that specifies the DSL main concepts (and their interrelationships) and a concrete syntax that implements it (usually via a textual or graphical notation).

The abstract syntax of a DSL defines its structural representation via a metamodel specification independent of any concrete syntax or notation. This section presents the metamodel of the core abstract syntax of MERLAN for expressing MUI requirements, outlining its key elements and relationships.

Figure 1 illustrates this metamodel expressed using a UML class diagram formalism, as usual.

Next subsections describe in more detail the main elements of the metamodel. We split the explanation in two subsections, one covering the metaclasses focused on the definition of Multimodal Requirements and one focusing more on the definition of Entities referenced in those requirements.

#### A. Definition of Multimodal Requirements

A `MultimodalRequirement` defines the rules that need to be evaluated in a system with multimodal inputs. These rules provide formal definitions of conditions the system should match in order to trigger some actions.

The metamodel defines two kinds of requirements: simple and complex.

1) *Complex Requirements*: A `ComplexRequirement` is a composition of `MultimodalRequirements`, which can be themselves also complex or simple. Requirements are composed with boolean operators, namely AND (all requirements under it must be satisfied), OR (at least one requirement must be satisfied) and NOT (the requirement condition under it must not be satisfied).

2) *Simple Requirements*: A `SimpleRequirement` expresses a condition on a single entity. When defining a simple requirement, we are creating a rule that would be satisfied when an entity is detected. Simple requirements have a `confidence` attribute to indicate the minimum confidence that needs to be achieved for the requirement to be satisfied (i.e., confidence on the entity detection).

`SimpleRequirement` has `AbstractRequirement` and `ConcreteRequirement` to distinguish between simple requirements referencing concrete and abstract entities, respectively. The difference between them is that a `ConcreteRequirement` can define a cardinality to specific the number of instances of that entity that should be detected. The cardinality notation is based on the UML cardinalities, allowing to define specific values ( $[n]$ , exactly  $n$  instances), concrete intervals ( $[m..n]$ , where  $m < n$  and  $m \geq 0$ ) or unbounded intervals ( $[n..*]$ , where  $n \geq 0$ ). As we will see later, entities can have attributes which can also be used during the matching process.

A `SimpleRequirement` has a specific modality, indicating which modality should be considered to satisfy the requirement. For instance, a `SimpleRequirement` referencing a `Person` concrete entity with `Image` modality will be

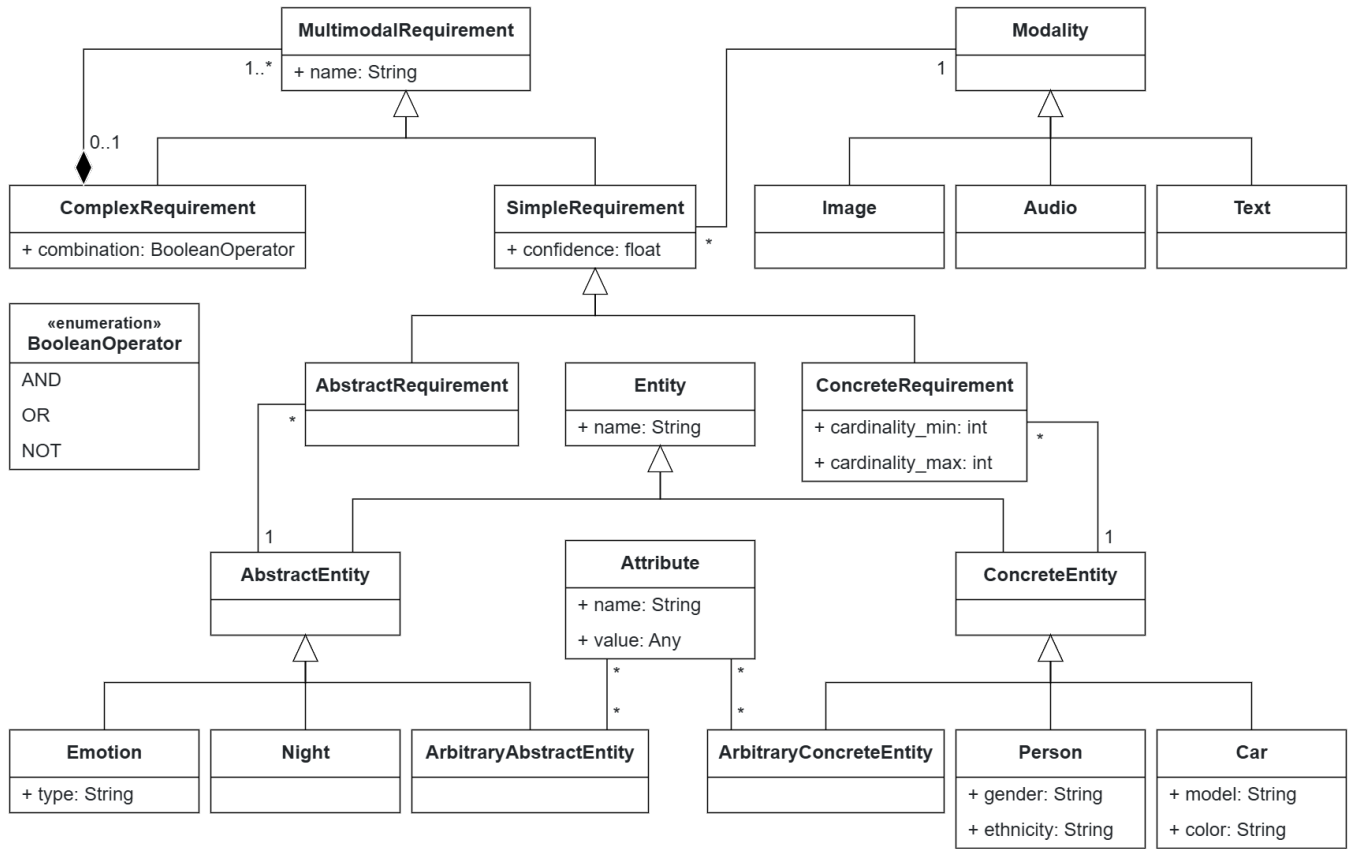


Fig. 1. MERLAN Metamodel

evaluated based on image inputs (i.e., looking for a person in an image), while the same requirement with `Audio` modality will analyze the audio input to find a person (i.e., if someone is speaking and the voice is detected). When evaluating the requirements, some attributes make sense only for specific modalities (e.g., the `color` of a `Car` entity only makes sense for image modality). This is something to consider when defining the entity attributes.

### B. Definition of Entities

An `Entity` represents a real-world concept, either material or abstract, that can be distinctly identified in some type of multimodal input. The simplest way to define entities is via a name and attributes that specify its structure and content. When defining an entity, we define its attributes, which may have specific values or empty ones (i.e., unknown values).

Setting an attribute value enforces the recognition engine to identify those entities with that exact attribute value. For instance, if we define a `Person` entity with `gender = "male"` attribute, we could use that entity within a requirement to restrict its applicability to men.

An attribute value can be left empty with the purpose of dynamically inferring its value during the recognition process. Following the `Person` entity example, a `gender` attribute with empty value should make the recognition engine identify

both men and women and fill the `gender` value of the recognized `Person` at runtime. Attributes with empty values also serve as placeholders that can be set at the requirement level. This mechanism provides flexibility by allowing the definition of generic entities and deferring context-specific details to individual requirements.

We consider two types of entities based on their nature, namely abstract and concrete entities.

1) *Concrete Entities*: A `ConcreteEntity` is a subclass of `Entity` that represents a concrete object or being within the multimodal environment that can be described in terms of its physical existence and referenced or interacted within the environment.

2) *Abstract Entities*: `AbstractEntity` is a subclass of `Entity` that does not have a clearly bounded physical existence but represents a concept, idea, property or classification that can be inferred from the environment. An example abstract entity could be `night`, indicating if an image is taken at night or not. A more abstract one could be `hazard`, indicating the level of hazard perceived from an image or audio (we could define some attributes for this property to describe our “hazard criteria”).

3) *Predefined and Arbitrary Entities*: The metamodel figure includes example instantiations of both concrete and abstract entities with default attributes. For instance, a simplified

Person concrete entity can be defined in terms of the attributes gender and ethnicity. Abstract and concrete entities have an `ArbitraryEntity` subclass that allows the definition of custom or domain-specific entities with arbitrary attributes.

To simplify the definition of requirements in a given domain we could predefine the most relevant entities in that domain based on existing ontologies for the domain. This is a trade-off between the size of the DSL and its ease of use.

#### IV. CONCRETE SYNTAX

This section introduces the concrete syntax of our DSL, which conforms to the previously detailed metamodel. The concrete syntax defines the rules ensuring a consistent and standardized use of the language. In this paper, we present a textual concrete syntax implemented with ANTLR, a powerful tool to generate language parsers from grammars.

We start by introducing the grammar and next an example MERLAN specification following the grammar. Listing 1 presents an excerpt of the DSL grammar.

Listing 1. MERLAN ANTLR grammar; main rules.

```
1 grammar MERLAN;
2
3 script
4   : entities?
5     requirements?
6   ;
7
8 entities
9   : ENTITIES NEWLINE
10      concrete_entities?
11      abstract_entities?
12   ;
13
14 concrete_entities
15   : CONCRETE NEWLINE concrete_entity*
16   ;
17
18 concrete_entity
19   : ID NEWLINE attribute*
20   ;
21
22 // Other rules omitted for brevity
23
24 requirement
25   : complex_requirement
26   | simple_requirement
27   ;
28
29 complex_requirement
30   : (AND | OR) NEWLINE requirement+
31   | NOT NEWLINE requirement
32   ;
33
34 simple_requirement
35   : abstract_requirement
36   | concrete_requirement
37   ;
38
39 concrete_requirement
40   : CONCRETE cardinality? NEWLINE attribute*
41   ;
```

42  
43 // Other rules omitted for brevity

The grammar defines the syntactic rules allowed in MERLAN language. At the first level, the `script` rule indicates that entities and requirements can be defined.

Some rules start with an uppercase keyword to divide the code into clear sections (e.g., see `ENTITIES`). The `entities` section can include concrete or abstract entities. For brevity, we excluded the `abstract_entities` rule (among others) in this paper, but it is analogous to `concrete_entities`. A concrete entity is defined by its name (i.e., the ID) and any number of attributes.

A requirement can be either complex or simple. A `complex_requirement` starts with a boolean operator (AND, OR or NOT) and a set of requirements (limited to 1 if the operator is NOT). A `simple_requirement` can be either abstract or concrete, where `concrete_requirement` can have a custom cardinality. Simple requirements contain a set of attributes, where some of them are mandatory. Mandatory attributes are not specified since the grammar's purpose is to ensure syntactical consistency. Evaluation on the missing attributes (as well as cardinality correctness) is done during the parser execution (i.e., when parsing a MERLAN code script). Mandatory attributes include `confidence`, `modality` and `entity` (those apply to simple requirements only) and `name`.

Listing 2 shows an example MERLAN code that defines entities and requirements following the previously proposed running example of a house agent in Section II.

Listing 2. Example MERLAN code.

```
1 ENTITIES:
2   CONCRETE:
3     person
4       - gender: ?
5       - ethnicity: ?
6     smoke
7     fire
8     dog
9       - breed: "labrador"
10    car:
11      - model: ?
12      - color: ?
13   ABSTRACT:
14     night
15       - description: "The image is taken at
16         night"
17     empty_house
18       - description: "The house is empty"
19 REQUIREMENTS:
20   requirement1:
21     CONCRETE
22       - entity: smoke
23       - name: "smoke"
24       - modality: "image"
25       - confidence: 0.5
26   requirement2
27     OR
28     CONCRETE
29       - entity: fire
30       - name: "fire"
31       - confidence: 0.5
```

```

31     - modality: "image"
32   AND
33   ABSTRACT
34     - entity: empty_house
35     - name: "empty_house"
36     - confidence: 0.3
37     - modality: "image"
38   OR
39   CONCRETE [1..*]
40     - entity: person
41     - name: "unknown_person"
42     - confidence: 0.7
43     - modality: "image"
44     - gender: "male"
45   CONCRETE [1..*]
46     - entity: car
47     - name: "unknown_car"
48     - confidence: 0.7
49     - modality: "image"

```

The first code block, identified with the `ENTITIES` keyword, contains all the entity definitions following the grammar rules. In this example, there are entities without attributes (see `smoke` and `fire`), an entity `dog` with an attribute with specific value (`breed: "labrador"`) and other entities (`person` and `car`) with attributes with empty values.

The requirements block, under the `REQUIREMENTS` keyword, contains the 2 example requirements defined in Section II. `requirement2` contains a composition of complex requirements where 2 of the inner simple requirements define cardinalities of `[1..*]` (i.e., minimum 1 instance). The requirement referencing the `person` entity shows how to set an entity attribute's value at the requirement level (see the `gender: "male"` attribute).

## V. TOOL SUPPORT

The full ANTLR grammar to use MERLAN is available in our open source repository in GitHub.

With the grammar, requirement engineers can describe full MERLAN specifications. But to make these specifications more actionable and offer a better Return On Investment (ROI) from them, we have also implemented a *prproof-of-concept* implementation to derive an agent implementation that understand and reacts to the requirement conditions effectively triggering a change on state based on detecting the satisfiability of the conditions on a multimodel input.

More specifically, we have implemented a transformation that, given a MERLAN specification creates an agent implemented on top of the BESSER Agentic Framework<sup>2</sup> where the MERLAN requirements have been transformed into a set of agent triggering conditions. Listing 3 shows the generated code for the example MERLAN code in Listing 2

The transformation is implemented via a custom implementation of a tree visitor pattern that traverses the abstract syntax tree generated by the MERLAN ANTLR parser. Note that this approach can be easily replicated for any language or agentic framework that supports similar requirements-based transitions.

<sup>2</sup><https://github.com/BESSER-PEARL/BESSER-Agentic-Framework>

The agentic framework we selected comes with integrated LLMs and Computer Vision models that capture information from a multimodal environment, which is used in real time to evaluate the requirements matching. The generated code includes multimodal entities, attached to the agent's entity database, and the requirements' conditions themselves. Based on this input processing component, the agent developer can complete the agent specification expressing the response behaviour to be executed in response of a triggering condition.

## VI. RELATED WORK

In this section, we discuss the work related to the specification of requirements for MUIs. We focus first on frameworks that explicitly address the creation of MUIs to then cover other approaches closer to the requirements engineering field, proposing languages to express requirements for some types of advanced interfaces.

### A. Multimodal User Interfaces

Recent work around MUIs is focusing on the exploitation of Machine Learning techniques to automatically extract patterns and insights from data [5].

A variety of frameworks have been proposed for this purpose. Xspeak [6] added a speech interface on top of X Window System, allowing to use words to interact with windows. Open Agent Architecture (OAA) [7] proposed a framework for multiagent systems with MUIs that included spoken language, handwriting and gesture. Openinterface [8] is another tool to design MUIs, providing its own runtime and IDE. The Squidy Interaction Library [9] proposed a library to reduce the efforts of designing MUIs, integrating different toolkits and frameworks in a common library integrating a GUI, hiding complexity by providing a visual language and a collection of devices and interaction techniques.

There is also a particular interest on MUIs in the robotics and healthcare domains. For instance, AMIR [10] is an assistive robot with voice and gesture-based interfaces, and FIRMA [11], a development framework for elderly-friendly interactive multimodal applications for assistive robots. MUIs also have a strong presence in smart home user interfaces [12].

Nevertheless, all these approaches focus on the development of the MUIs themselves not on the formalization of the requirements they are supposed to implement, which is exactly the purpose of our MERLAN proposal.

### B. Domain-Specific Languages

DSLs are used in many Machine Learning problems [13], including DSLs that focus on formalizing requirements [14]. An example is Impromptu [15] proposed a DSL to define structured prompts in a modular and tool-independent way. Other DSLs focus more on fairness aspects of Machine Learning, e.g. [16], [17].

Closer to our work, other DSLs cover multimodal aspects of software components. For instance, SEMKIS [18] focus on requirements engineering of datasets and neural networks to improve recognition skills. ViSaL [19] allows the programmer

Listing 3. Generated Python code.

```

1 # Entities
2 person = ConcreteEntity(name="person", attributes={"gender": None, "ethnicity": None})
3 smoke = ConcreteEntity(name="smoke", attributes={})
4 fire = ConcreteEntity(name="fire", attributes={})
5 dog = ConcreteEntity(name="dog", attributes={"breed": "labrador"})
6 car = ConcreteEntity(name="car", attributes={"brand": None, "model": None, "color": None})
7 night = AbstractEntity(name="night", attributes={"description": "The image is taken at
    night"})
8 empty_house = AbstractEntity(name="empty_house", attributes={"description": "The house is
    empty"})
9 # Requirements
10 requirement1 = RequirementDefinition("requirement1")
11 requirement1.set(ConcreteRequirement(name="smoke", concrete_entity=smoke,
    attributes={"modality": "image", "confidence": 0.5}))
12 requirement2 = RequirementDefinition("requirement2")
13 requirement2.set(
14     OR([
15         ConcreteRequirement(name="fire", concrete_entity=fire, attributes={"confidence": 0.5,
            "modality": "image"}),
16         AND([
17             AbstractRequirement(name="empty_house", abstract_entity=empty_house,
                attributes={"confidence": 0.3, "modality": "image"}),
18             OR([
19                 ConcreteRequirement(name="man", concrete_entity=person, attributes={"min": 1, "max":
                    0, "confidence": 0.7, "modality": "image", "gender": "male"}),
20                 ConcreteRequirement(name="unknown_car", concrete_entity=car, attributes={"min": 1,
                    "max": 0, "confidence": 0.7, "modality": "image"})
21             ])
22         ])
23     ])
24 )
25
26 # The following code is not automatically generated. It illustrates how to use a requirement
    to define transitions between the agent states (agent definition code ommited for brevity)
27 initial_state.when_requirement_matched_go_to(requirement1, smoke_state)

```

to express image quality detection rules for enforcing safety constraints, increasing trustworthiness in robot perception systems. FVision [20] designed as a Haskell library as a DSL to build and test visual tracking systems. The Midgar IoT platform was extended by adding a Computer Vision module to automate camera input analysis. Their approach detects only people, requiring model training for other objects. They propose developing a DSL to streamline the Computer Vision pipeline [21]. Note that there are existing resources to evaluate such systems, such as CLEVR [22], which proposed a dataset to benchmark visual reasoning in smart systems. More specific to the chatbot domain, [3] and [23] propose DSLs for intent matching and entity recognition in textual inputs.

Note that the above examples target specific types of inputs and, many of them, specific environments, while MERLAN aims to provide a multimodal requirements solution combining different types of requirements and entities (including concrete and abstract ones) and conditions on them, opening the door to the creation of powerful multimodal agents that satisfy the user requirements.

## VII. RESEARCH ROADMAP

To fully define a robust and expressive Domain-Specific Requirements Language for Multimodal User Interfaces, several key aspects must be addressed. In what follows we list some of them.

- Add a graphical notation to MERLAN, enhancing usability and making it easier for less-technical practitioners and engineers to specify and visualize MUI requirements. This should even include a visual modeling by example component where users could give images as example scenarios that should trigger an action in the system. Alternative, also the option to simply describe a requirement in natural language. Machine Learning can be used to support this task.
- Extend the language to cover also the specification of requirements on the behaviour to perform when the conditions are matched. For "traditional" actions, existing behavioural languages (e.g. UML specifications) could suffice but for multimodal responses, an extension to MERLAN, where multimodality is already a first-class element could be a better option.
- Time conditions to enable expressing temporal constraints



in the requirements, e.g., constraints on the duration of a certain object in a video before triggering an action

- Additionally, hierarchical modalities should be explored, where higher-level modalities (e.g., gestures or facial expressions) are derived from lower-level ones (e.g., images or video). This hierarchical structuring will enable more precise and flexible requirements definitions for complex multimodal interactions.
- Quality analysis to detect inconsistencies and conflicts among multimodal requirements. For example, certain attributes, such as color, are relevant only in image-based modalities and may not be applicable in textual or auditory contexts. Or more challenging to detect, two conditions in an image may be mutually exclusive, implying that such condition can never be satisfied.

## VIII. CONCLUSIONS

In this paper, we introduced MERLAN, a Domain-Specific Language designed to standardize the specification of requirements for Multimodal User Interfaces (MUIs). Our approach leverages a metamodel-based formalization and a textual concrete syntax to facilitate the specification process while also providing a proof-of-concept implementation showing a potential path to transform these requirements into actionable agent code to execute the requirements in real-time multimodal environments.

Future work will focus on expanding the capabilities of MERLAN by addressing the open challenges discussed above. Additionally, we aim to conduct empirical validation experiments and extend our tool support to foster adoption our language and infrastructure by researchers and practitioners in the field. As part of such tool extension, we plan to develop a library of predefined entities, based on existing ontologies, to be imported and reused when expressing new requirements.

## REFERENCES

- [1] M. Z. Baig and M. Kavakli, "Multimodal systems: Taxonomy, methods, and challenges," 2020. [Online]. Available: <https://arxiv.org/abs/2006.03813>
- [2] L. M. Reeves, J. Lai, J. A. Larson, S. Oviatt, T. S. Balaji, S. Buisine, P. Collings, P. Cohen, B. Kraal, J.-C. Martin, M. McTear, T. Raman, K. M. Stanney, H. Su, and Q. Y. Wang, "Guidelines for multimodal user interface design," *Commun. ACM*, vol. 47, no. 1, p. 57–59, Jan. 2004. [Online]. Available: <https://doi-org.proxy.bnl.lu/10.1145/962081.962106>
- [3] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: A multimodal low-code chatbot development framework," *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.
- [4] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, p. 316–344, Dec. 2005. [Online]. Available: <https://doi-org.proxy.bnl.lu/10.1145/1118890.1118892>
- [5] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2019.
- [6] C. Schmandt, M. Ackerman, and D. Hindus, "Augmenting a window system with speech input," *Computer*, vol. 23, no. 8, pp. 50–56, 1990.
- [7] D. B. Moran, A. J. Cheyer, L. E. Julia, D. L. Martin, and S. Park, "Multimodal user interfaces in the open agent architecture," in *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, ser. IUI '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 61–68. [Online]. Available: <https://doi-org.proxy.bnl.lu/10.1145/238218.238290>
- [8] M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef, "The openinterface framework: a tool for multimodal interaction," in *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 3501–3506. [Online]. Available: <https://doi.org/10.1145/1358628.1358881>
- [9] 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, Apr 2010. [Online]. Available: <https://doi.org/10.1007/s12193-010-0044-2>
- [10] D. Ryumin, I. Kagirow, A. Axyonov, N. Pavlyuk, A. Saveliev, I. Kipyatkova, M. Zelezny, I. Mporas, and A. Karpov, "A multimodal user interface for an assistive robotic shopping cart," *Electronics*, vol. 9, no. 12, 2020. [Online]. Available: <https://www.mdpi.com/2079-9292/9/12/2093>
- [11] N. Kazepis, M. Antona, and C. Stephanidis, "Firma: A development framework for elderly-friendly interactive multimodal applications for assistive robots," 04 2016.
- [12] M. Blumendorf and S. Albayrak, "Towards a framework for the development of adaptive multimodal user interfaces for ambient assisted living environments," in *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, C. Stephanidis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 150–159.
- [13] I. Portugal, P. Alencar, and D. Cowan, "A preliminary survey on domain-specific languages for machine learning in big data," in *2016 IEEE International Conference on Software Science, Technology and Engineering (SWSTE)*, 2016, pp. 108–110.
- [14] Z. Pei, L. Liu, C. Wang, and J. Wang, "Requirements engineering for machine learning: A review and reflection," in *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*, 2022, pp. 166–175.
- [15] S. Morales, R. Clarisó, and J. Cabot, "Impromptu: a framework for model-driven prompt engineering," *Software and Systems Modeling*, Jan 2025. [Online]. Available: <https://doi.org/10.1007/s10270-024-01235-4>
- [16] A. Yohannis and D. Kolovos, "Towards model-based bias mitigation in machine learning," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 143–153. [Online]. Available: <https://doi.org/10.1145/3550355.3552401>
- [17] J. Giner-Miguel, A. Gómez, and J. Cabot, "Describeml: A dataset description tool for machine learning," *Sci. Comput. Program.*, vol. 231, p. 103030, 2024. [Online]. Available: <https://doi.org/10.1016/j.scico.2023.103030>
- [18] B. Jahić, N. Guelfi, and B. Ries, "Semkis-dsl: A domain-specific language to support requirements engineering of datasets and neural network recognition," *Information*, vol. 14, no. 4, 2023. [Online]. Available: <https://www.mdpi.com/2078-2489/14/4/213>
- [19] J. T. M. Ingbergsson, D. Kraft, and U. P. Schultz, "Safety computer vision rules for improved sensor certification," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, 2017, pp. 89–92.
- [20] J. Peterson, P. Hudak, A. Reid, and G. Hager, "Fvision: A declarative language for visual tracking," in *Practical Aspects of Declarative Languages*, I. V. Ramakrishnan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 304–321.
- [21] C. González García, D. Meana-Llorián, B. C. Pelayo G-Bustelo, J. M. Cueva Lovelle, and N. Garcia-Fernandez, "Midgar: Detection of people through computer vision in the internet of things scenarios to improve the security in smart cities, smart towns, and smart homes," *Future Generation Computer Systems*, vol. 76, pp. 301–313, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X16308652>
- [22] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1988–1997.
- [23] S. Pérez-Soler, E. Guerra, and J. de Lara, "Model-driven chatbot development," in *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings*, ser. Lecture Notes in Computer Science, G. Dobbie, U. Frank, G. Kappel, S. W. Liddle, and H. C. Mayr, Eds., vol. 12400. Springer, 2020, pp. 207–222. [Online]. Available: [https://doi.org/10.1007/978-3-030-62522-1\\_15](https://doi.org/10.1007/978-3-030-62522-1_15)