Multimodal Technologies and Interaction

MDPI

Article

Semantic Fusion for Natural Multimodal Interfaces using Concurrent Augmented Transition Networks

Chris Zimmerer, Martin Fischbach and Marc Erich Latoschik *

Chair for Human–Computer Interaction, University of Würzburg, Am Hubland, 97074 Würzburg, Germany; chris.zimmerer@uni-wuerzburg.de (C.Z.); martin.fischbach@uni-wuerzburg.de (M.F.)

* Correspondence: marc.latoschik@uni-wuerzburg.de; Tel.: +49-(0)931-31-85871

Received: 14 October 2018; Accepted: 4 December 2018; Published: 6 December 2018

Abstract: Semantic fusion is a central requirement of many multimodal interfaces. Procedural methods like finite-state transducers and augmented transition networks have proven to be beneficial to implement semantic fusion. They are compliant with rapid development cycles that are common for the development of user interfaces, in contrast to machine-learning approaches that require time-costly training and optimization. We identify seven fundamental requirements for the implementation of semantic fusion: Action derivation, continuous feedback, context-sensitivity, temporal relation support, access to the interaction context, as well as the support of chronologically unsorted and probabilistic input. A subsequent analysis reveals, however, that there is currently no solution for fulfilling the latter two requirements. As the main contribution of this article, we thus present the Concurrent Cursor concept to compensate these shortcomings. In addition, we showcase a reference implementation, the *Concurrent Augmented Transition Network* (CATN), that validates the concept's feasibility in a series of proof of concept demonstrations as well as through a comparative benchmark. The CATN fulfills all identified requirements and fills the lack amongst previous solutions. It supports the rapid prototyping of multimodal interfaces by means of five concrete traits: Its declarative nature, the recursiveness of the underlying transition network, the network abstraction constructs of its description language, the utilized semantic queries, and an abstraction layer for lexical information. Our reference implementation was and is used in various student projects, theses, as well as master-level courses. It is openly available and showcases that non-experts can effectively implement multimodal interfaces, even for non-trivial applications in mixed and virtual reality.

Keywords: multimodal fusion; multimodal interface; semantic fusion; procedural fusion methods; natural interfaces; human-computer interaction

1. Introduction

Multimodal interfaces (MMIs) implement human-computer interaction paradigms that center around users' natural behavior and communication capabilities [1]. Such interfaces combine at least two modalities potentially operating simultaneously [2]. During intentional interactions, the specification of parameters for required system actions can thus be distributed among adequate modalities in a synergistic fashion [3]. For instance, using speech to specify an interaction and deictic gestures to specify operands and target locations (see Figure 1). Potential benefits of such multimodal interfaces include increased expressiveness, flexibility, reliability, and efficiency [4–6].

Figure 1. Intentional multimodal interaction (MMI) in three application areas: *Space Tentacle*, a fully-immersive virtual reality adventure game ([7] left). A mixed reality real-time strategy game on an interactive surface ([8] middle). *SiXton's Curse*, a semi-immersive adventure game played in front of a power wall ([9] right).

Natural (human) behavior or rather interpersonal communication is context dependent. Prior communication contents and especially the surrounding environment have to be taken into account when interpreting multimodal utterances. The benefits of MMIs thus become especially apparent if interactions are spatially and temporally grounded with an environment in which the user is (physically) situated [10]. For example, it is easier to point at an object as part of a speech-gestural instruction than to provide similar information via a keyboard, because it uses the natural spatial referencing of interpersonal communication. Such situated human-computer interaction environments range from physical spaces to so called *virtual environments*. Respective application areas range from smart homes [11] and human-robot interaction [12] to mixed reality [8] and virtual reality [7].

If MMIs are to be supported, input modalities have to be jointly analyzed at some point of processing to derive a conjoint meaning [13,14]. This analysis has been given different names, ranging from *combining* [15] to *multimodal integration* [16]. More widely the term *fusion* is used, e.g., *multimodal input fusion* [17] or the more recently used *multimodal fusion* [18–20]. Multimodal fusion is applied for a wide variety of scenarios. They range from analyzing intentionally performed communication based on speech and gesture [7,8,12,21,22] to drawing conclusions on one's intentions and feelings based on unintentional (social) signals [23–26], like eye movements and changes in body posture.

The concrete mechanism of applying fusion is denoted as *fusion method*. There is a wide variety of methods used for multimodal fusion. They range from frame-based [27,28], over unification-based [29,30], symbolic-statistical [31–33], and procedural methods [15,16,34,35] to machine learning approaches [36–38]. An elaborated overview is given by [14,39]. Fusion methods can be categorized within a spectrum that extends from early fusion, comprising low-level data stream-oriented approaches, to late fusion, comprising high-level semantic inference-based approaches [4,40,41].

Early fusion requires tightly synchronized input, like audio-visual speech recognition [42]. Here, machine learning approaches exhibit great potential. It has been shown that deep learning approaches significantly out-perform traditional approaches in challenging tasks, such as computer vision, natural language processing, robotics, and information retrieval [43]. When building multimodal systems, these approaches are predominantly used as a black box to interpret each modality individually, i.e., transforming raw data to features (see Figure 2, left gray box).

For example, an infrared tracking system fuses gray scaled images and outputs positions and orientations of reflective markers. These numerical features can be fused with other features, e.g., angles obtained from a data glove, to provide symbolic features, like the type of performed gestures.

On the other end of this spectrum, a decision has to be made on what concrete system reactions are to be derived from intentional multimodal commands (see Figure 2, right black box). This late fusion process leads to the explicit combination of primarily semantic features from multiple modalities. It is denoted as *semantic fusion* [44]. In contrast to early fusion, modalities at this level are typically loosely coupled with respect to their temporal occurrence, e.g., detected gestures and speech tokens.

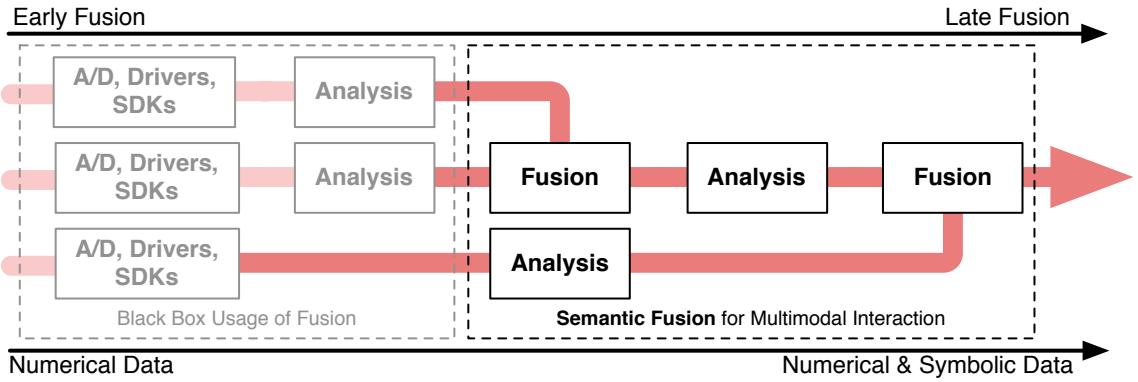

Figure 2. Multimodal fusion categorized in early and late fusion. Early fusion methods are usually used as black boxes for MMIs. Late fusion is mostly realized by unification-based or procedural methods.

Semantic fusion methods started to be influenced by machine learning as well. In human–robot interaction a suitably trained bayesian network has been successfully used as fusion method [12]. Support vector machines have been used to combine gaze movement and brain activity to provide an interface for implicit selection in a graphical user interface [45]. It has been demonstrated that a fusion method based on hidden Markov models (HMM) is more accurate than a classic frame-based fusion algorithm by adapting to the user [33]. The *members to teams to committee* technique [31] increases the overall robustness of semantic fusion methods by offering a preprocessing step based on the empirical posterior distribution of co-occurring modality estimates.

However, machine learning is not without drawbacks. Respective approaches involve careful tuning of learning parameters and model hyperparameters [46]. They also require the selection of relevant features and oftentimes large corpora for training [47,48]. Building these corpora and optimizing the algorithms requires a lot of effort and time. This, however, contradicts iterative prototyping and evaluation commonly applied in interface design to produce suitable interfaces [49].

In contrast, descriptive fusion methods pose general-purpose approaches. In addition, they can be altered and adapted more easily without the need for time costly training and optimization procedures. This compatibility with typical interface development processes makes them the predominantly utilized approach for semantic fusion. Moreover, they are open for supplementation with machine learning, as shown by [12,31]. Descriptive fusion methods are thus also compatible with a possible future increase in machine learning approaches for multimodal fusion.

The two most prominent classes thereof are unification-based and procedural methods [14]. Unification-based approaches, like [30], are highly expressive and support many interface types. However, their computational complexity and their potential for mutual compensation amongst modalities render procedural methods the better choice [34]. Traditional procedural methods, like the augmented transition network (ATN) [50] and the finite-state transducer, are used but not designed for multimodal fusion. Because of their simple comprehensibility, they have been extended to meet more requirements of multimodal fusion. For instance, reference [16] present a temporal Augmented Transition Network (tATN) that facilitates the consideration of temporal relations between input. However, there is currently no solution on how to fulfill two fundamental semantic fusion requirements (see Table 1): Handling probabilistic and chronologically unsorted input [14,16].

1.1. Research Question

The problem addressed by this article is thus formulated as follows.

How can a procedural fusion method satisfy all fundamental requirements for performing semantic fusion, while maintaining the support of rapid MMI development processes?

Table 1. Finite State Automata (FSAs), Finite State Transducers (FSTs), Augmented Transition Networks (ATNs), and Temporal Augmented Transition Networks (tATNs) rated with respect to fundamental requirements (R). The Concurrent Augmented Transition Network (cATN) represents the contribution of this article. Yellow *Yes* cells denote that the rated method allows the fulfillment of the requirement, however there is no explicit support. Yellow *No* cells denote that the rated method theoretically allows the fulfillment of the requirement, however a respective implementation is highly impractical. The red frame highlights the identified shortcomings. Green *Yes* cells denote that the rated method fulfills the requirement. Red *No* cells denote that the rated method does not fulfill the requirement. Refer to Section 2 for details.

		FSA	FST	ATN	tATN	cATN
Action Derivation	R_{act}	Yes	Yes	Yes	Yes	Yes
Continuous Feedback	R_{fb}	No	Yes	Yes	Yes	Yes
Context-sensitive	R_{syn}	No	No	Yes	Yes	Yes
Temporal Relation Support	R_{tmp}	No	No	Yes	Yes	Yes
Access to Interaction Context	R_{sem}	No	No	Yes	Yes	Yes
Chronologically Unsorted Input	R_{uns}	No	No	No	No	Yes
Probabilistic Input	R_{prob}	No	No	No	No	Yes

1.2. Contribution

Targeting semantic fusion methods for natural and intentional multimodal interactions that are compliant with rapid development cycles, we identify seven fundamental requirements: Action derivation, continuous feedback, context-sensitivity, temporal relation support, access to the interaction context, as well as the support of chronologically unsorted and probabilistic input. Our main contribution, the Concurrent Cursor concept, provides a solution for fulfilling the latter two requirements with procedural methods. In addition, we showcase a reference implementation, the *Concurrent Augmented Transition Network* (CATN), that validates the concept's feasibility in a series of proof of concept demonstrations as well as through a comparative performance benchmark. The CATN fulfills all identified requirements and fills a lack amongst previous solutions. It supports the rapid prototyping of multimodal interfaces by means of five concrete traits: Its declarative nature, the recursiveness of the underlying transition network, the network abstraction constructs of its description language, the utilized semantic queries, and an abstraction layer for lexical information. Our reference implementation is available for the research community [51].

1.3. Structure of the Paper

The remainder of this article is structured as follows. Section 2 introduces an interaction use case and presents a requirement analysis for semantic fusion on the basis of a comprehensive review of associated contributions. Existing procedural fusion methods are evaluated with regard to the identified fundamental requirements. Section 3 presents our results: The Concurrent Cursor concept, a reference implementation, as well as the comparative benchmark. Our validation activities by means of proof of concept demonstration are reported in Section 4, followed by a discussion of results, implications, and limitations. Section 5 concludes this article and points out promising future work.

2. Requirement Engineering

This section reports on a comprehensive review of associated contributions that identifies seven fundamental requirements for semantic fusion. For that purpose, we introduce and analyze an example use case. A review of procedural fusion methods reveals that there is no solution which fulfills all of these requirements.

2.1. Use Case

A user furnishes a virtual room in an architectural modeling application. The user can give intentional instructions to create, modify, and delete virtual furniture by means of combined speech and gesture commands (see Figure 3). The commands are in the style of the pioneering *Put that there* demonstration [21]. In contrast to Bolt's work, this is an interface to a real-time interactive virtual environment. That is, the user is (virtually) situated within the environment by means of a head mounted display, not sitting in front of it. An exemplary interaction involving speech and a deictic gesture looks like the following: *"Put [deictic gesture] that green chair near [deictic gesture] this table."*

Figure 3. The user is fully immersed into a real-time interactive virtual environment. A multimodal, speech and gesture, interface allows the user to furnish a virtual room.

2.2. Analysis

In order to create such an application, two primary tasks have to be fulfilled: The implementation of the virtual environment and of the MMI. Real-time interactive systems (RISs) facilitate technical realizations of such situated interaction environments [10]. These systems typically consist of subsystems to capture and process relevant user behavior and physical environment changes, to simulate a virtual environment, and to provide coherent rendering output to the user within soft real-time constraints. In addition, performance is crucial to guarantee low system response times and hence usability, immersion, and security. Conceptually, RIS applications comprise state and behavior. The **application state** is made up of all meaningful objects including their properties. For example, the object `Table` has a property `Transformation` representing its position, orientation, and scale in the virtual environment. **Actions** describe operations that are invokable by the user via the interface and that alter the application state in a way that can be perceived by the user. They are usually triggered by user interactions or virtual agents. A possible action in our use case is placing the virtual object `Chair` next to the `Table`.

Interface-wise the user's communicative utterances, e.g., posture and spoken word, have to be captured and processed. A typical approach to realize a respective MMI is as follows. Sensors provide information via drivers or software development kits (SDKs) in different abstraction levels. A microphone captures the user's voice and an automated speech recognizer provides a text representation of spoken words. The positions and orientations of a user's head and hand movement is tracked by means of a head-mounted display and controller setup. A trained neural network analyzes the user's motion patterns and detects gestures. The output data of both the speech and the gesture recognizer are jointly analyzed by the fusion method implementation to derive a conjoint meaning. A developer has to implement a semantic fusion method and configure it in a way it recognizes desired interactions. To this end, the developer has to implement a multimodal processing pipeline that communicates data from one processing step to the next. For instance, from a tracking system

to a gesture recognizer to a fusion method and finally to the application. Events or a shared state are common mechanisms to communicate such data.

Considering this (real-time interactive) system setting, the following analysis focusses on three essential aspects of semantic fusion: Input, context, and output. The fusion method has to analyze data from different sources, i.e., input from a speech and a gesture recognition system. The user's interactions are highly dependent on the application's context, like which chair has to be placed next to what table in our use case. Further, the fusion method has to communicate feedback and its recognition result. Based on its output, respective actions within the application have to be invoked.

2.2.1. Input

Semantic fusion implementations receive input from early fusion systems, e.g., from a speech and gesture recognizer. This input is commonly probabilistic, mostly due to the utilization of machine learning approaches. This probabilistic character is quantified by *confidence values* typically ranging from 0 to 1, e.g., obtained from the activation value of output layer neurons of a neural network. In addition, early fusion systems often output several hypotheses for each analyzed time frame. These sets of hypotheses are commonly called *n-best guesses*, where *n* denotes the number of alternatives a system provides. For instance, a gesture recognizer outputs two guesses for one analyzed user motion pattern and a speech recognizer outputs two guesses for one analyzed utterance (see Table 2). This data is communicated to and analyzed by a semantic fusion system.

Table 2. Exemplary output of a speech and gesture recognizer. Both recognizer provide probabilistic output in form of n-best guesses with a confidence value between 0 and 1. Simply combining the best guess of each modality might not always produce the best result. The best guess of the speech recognizer "*Put that green chair*" indicates an accompanying pointing gesture over a rotation.

	Speech Recognizer	**Gesture Recognizer**
Best Guess	Put that green chair (0.9)	rotating (0.8)
Second Best Guess	Rotate the green chair (0.2)	pointing (0.7)
Third Best Guess	...	...

The most simple way of analyzing the probabilistic input is to convert it to binary input by picking only the most confident guess for each modality during semantic fusion. However, it has been shown how the recognition results of one modality can *"pull up"* the confidence of other results during multimodal fusion [3]. Simply fusing the best guesses "*Put that green chair*" with the *rotating* gesture does not yield a semantically valid result (see Table 2). The semantics of the speech recognizer's best guess suggests that the user in fact performed a pointing gesture instead of a rotating gesture. For that reason it is necessary that a fusion method is capable of processing probabilistic input. In other words, it has to be able to process multiple hypothesis for each modality and check different possible combinations between modalities to provide the most probable and semantically valid output.

Another common task is to define chronological sequences of possible inputs. This is straightforward in unimodal interfaces. Input is received via only one channel in a clear order. During multimodal interaction it is not always obvious which input precedes another. For example, a deictic word occurs during or after the gesture in 97% of the cases [52]. It is thus particularly important that a fusion method supports temporal relations between input, e.g., after, during, and before.

However, this presupposes that the fusion method processes input in the same order it has been performed by the user. This might not always be the case since input does not necessarily arrive at the fusion method implementation in that order. On the one hand, concurrently running recognizers may take different processing times across modalities. For example, recognizing a gesture might take longer than recognizing a spoken word. On the other hand, probabilistic output, as described before, causes an issue as well. During a user interaction a recognizer can formulate different hypotheses

about the input. Alternative hypotheses about preceding parts of an interaction might occur and be communicated to the fusion method. For instance, current speech recognizers like the *Microsoft Speech Recognition API* provide hypotheses for preceding words even after the user uttered additional words. This contributes to the problem of chronologically unsorted input. To this end, it is necessary to sort inputs based on their time of occurrence before processing them.

In summary, a fusion method has to be able to **analyze probabilistic input**. If recognizers provide output in form of n-best guesses, **all hypotheses** and **all temporally valid combinations of hypotheses** have to be analyzed as well. The fusion method has to **support different temporal relations between inputs**, and **handle chronologically unsorted input**. Issues concerning both time and probabilistic input have been also identified by [14] and [16] before that. While [16] describe it as the capability to process parallel incoming percepts and support for temporal relations, reference [14] postulate the two requirements: multiple and temporal combinations & probabilistic inputs. However, both scientific contributions do not address the issue of parsing n-best guesses foreach input, i.e., how to handle different hypotheses.

2.2.2. Context

Besides time and confidence another important aspect is context. Putting a recognized user command into context, i.e., performing *semantic integration*, can be accomplished during or after the multimodal fusion. For instance, reference [53] describes an approach for the latter. A fusion method checks for syntactic correctness of multimodal utterances. Parse results include the syntactic structure of valid utterances, e.g., the subject "*chair*" and the verb "*to put*" as well as pointing directions mapped to corresponding demonstratives. They are communicated to a semantic integration implementation that is a subsystem of the RIS realizing the virtual environment. This subsystem has to check the semantic correctness of the command and execute the respective action. For the introduced use case "*Put [deictic gesture] that green chair near [deictic gesture] this table.*" this entails the following steps: It has to determine if the object pointed at (deictic gesture) really denotes a green chair or table (speech). The user command's corresponding action has to be identified and retrieved. Before executing the action, it's parameters and preconditions have to be checked, e.g., is the object `Chair` movable. It has to correctly parametrize the retrieved action and execute it if the preconditions are fulfilled.

However, performing semantic integration after fusion holds two disadvantages. Firstly, linguistic knowledge about the syntactic structure of valid utterances is required, to resolve application specific dependencies. For instance, how does the deictic gesture relate to the verb "*to put*" and the subject "*chair*". This information is already present in the fusion method implementation that performs the syntactic check in the first place. Implementing semantic fusion as a subsystem of a RIS avoids this redundancy and potentially decreases the end-to-end latency for action execution and feedback provision. Secondly, semantic integration allows the fusion subsystem to check the semantic validity of an interaction during fusion. For example, if there is no green chair present in the user's pointing direction, the fusion can discard this guess. This does not only improve the reliability of other guesses but increases performance as well.

Either way, a suitable interface to state and behavior is required. The architecture of this interface is highly dependent on whether a fusion method is integrated into a RIS platform or stand alone. The latter include frameworks like *OpenInterface* [54], *Mudra* [40], or *SSI* [55]. They provide functionality for integrating sensors and performing fusion with the goal to foster modularity and reusability. However, they either rely on a time costly synchronization mechanism with a RIS, e.g., via sockets, or have no access to an application's context at all. Hence, they are less suited to perform semantic integration. On the contrary, platforms with an integrated MMI framework, like *SGIM* and *virtuelle Werkstatt* [56], *ICare* [57], and *Simulator X* [20], commonly provide a uniform access scheme to the application's context.

However, conceptualizing and implementing an appropriate interface is not without challenge. This particularly effects the area of RIS where multiple subsystems have to collectively maintain

a real-time simulation of a coherent application state. On the one hand, this interdependencies between individual state representations, mutual state access, overall synchronization, and flow of control imply a conceptual close coupling. On the other hand, software quality asks for a decoupling to develop maintainable solutions. This contradiction negatively effects the availability of these frameworks [20,58]. This challenge is exacerbated by the integration of subsystems that reflect the overall state and behavior, like a semantic fusion subsystem which performs semantic integration [22,59,60].

An interface for performing semantic integration has to be able to access the application state and behavior. Consider the first part of the introduced interaction: "*Put [deictic gesture] that green chair ...*". Words like "put", "chair", and "green" map to concrete actions, objects, and properties. Transformations and directions are important to interpret the deictic gesture. The interface has to support access by means of both symbolic and numerical values, e.g., to resolve a concrete entity reference.

The interface has to fulfill another requirement. Commonly the application state of a RIS is subject to dynamic changes caused by, e.g., physic simulations. An application state history is required to cope with the delay between performing an input and analyzing it. This delay is exacerbated by chronologically unsorted input and general latency. For example, until a deictic gesture is recognized and used to perform semantic integration a certain amount of time has passed. It is not ensured that the chair has the same position as it did when the climax of the deictic gesture occurred. For that reason, relevant application state histories shall be accessible.

In conclusion, context is essential when performing fusion to check semantic validity and to derive actions. This has also been identified by [14] who state that multimodal commands can be interpreted differently depending on the context of their use, e.g., in a car, at home or at work. Performing semantic integration directly during fusion promises potential benefits in term of performance and the reliability of recognition results. However, providing the fusion subsystem with a suitable **access to the context** of an application is not without challenges. A well defined interface between fusion subsystem and the application is required. For example, reference [16] describes a latching mechanism into real-time applications to support for application context integration. The interface has to provide access to the **application state**, executable **actions**, and **application state history**.

2.2.3. Output

A fusion method's output can be categorized in actions and feedback. On the one hand, the intention of a multimodal interaction is to trigger a certain action that ultimately alters the application state, especially in instruction-based interfaces. To this end, the output of a multimodal fusion method implementation has to be matched to a respective action. On the other hand, appropriate feedback is crucial for all interfaces [61]. Multimodal interfaces introduce another layer of uncertainty for the user, if an application does not behave in an expected fashion in response to an interaction. Input does does not correspond to unambiguous events, like mouse clicks or keyboard presses, but is inherently probabilistic. The fusion method should thus provide feedback about its current state after each processed input. Action derivation and feedback provision can be implemented in two ways. A fusion method implementation can communicate its parsing results to a semantic integration subsystem, which performs actions and feedback. This can be achieved more efficiently, if the fusion method is implemented as a RIS subsystem and can directly access state and behavior.

If a semantic fusion implementation shall properly consider input in form of n-best guesses, providing feedback requires special consideration. The parsing result then can consist of n-best guesses as well. The best guess can change upon new input. As a consequence, previously provided feedback has to be revertible. Feedback can be presented to the developer or user in different ways. For example, presented in form of simple debug print lines to the console. Especially for a user, situated in a virtual environment, such feedback is not very helpful. More diegetic feedback grounded within the virtual environment is desirable. For example, after the first half of the interaction "*Put [deictic gesture] that green chair ...*" the respective entity could simply be visually highlighted. Even more natural

ways of feedback are conceivable, e.g., when interacting with a virtual (humanoid) agent. The agent could move his head and look towards the referenced chair or affirmatively nod his head whenever a dedicated part of a command has been recognized. With respect to changing best guesses of the fusion method, highlighted objects might be unhighlighted or a virtual agent might change its gaze.

In conclusion, it is important that the fusion method is capable to **provide feedback after each processed input** and **trigger actions**. This is a prerequisite to keep the user informed about the current state of the semantic fusion and provide adequate insight in why a certain command has not been successfully recognized. Lastly, the fusion method has to be able **to revert previously given feedback** to perform proper error handling. This becomes essential if dealing with chronologically unsorted input and multiple probabilistic hypotheses for each input.

2.3. Requirement Specification

The analysis derives fundamental requirements that a fusion method has to fulfill to perform semantic fusion for intentional multimodal interactions (written bold in Section 2.2). In the following, we summarize these requirements:

R_{act} The fusion method shall be able to trigger actions.

R_{fb} The fusion method shall be able to provide feedback after each processed input.

- If multiple hypothesis are analyzed, previously given feedback shall be revertible.

R_{syn} The fusion method shall analyze multimodal input with respect to a context-sensitive syntax.

R_{tmp} The fusion method shall support arbitrary temporal relations between input.

R_{sem} The fusion method shall be able to access the interaction context for semantic integration.

- The application state shall be accessible.
- Relevant application state histories shall be accessible.
- Actions shall be accessible.

R_{prob} The fusion method shall be able to analyze probabilistic input.

- If n-best guesses are provided by a recognizer, all hypothesis shall be analyzed.
- If multiple hypothesis are analyzed, all temporally valid combinations shall be considered.

R_{uns} The fusion method shall be able to analyze chronologically unsorted input.

2.4. Procedural Fusion Methods

As argued in Section 1, procedural fusion methods are the most suitable method to perform semantic fusion in RIS. In the following we analyze four commonly used procedural fusion methods with regard to the introduced requirements: Finite-State Automata, Finite-State Transducers, Augmented Transition Networks and the temporal Augmented Transition Networks.

Initially, finite-state automata have been used to parse natural language [62]. Later, they have been adapted for multimodal fusion [63]. A finite state automaton consists of a finite set of states, a transition function and a finite alphabet of input letters [64]. Usually a finite-state automaton is visually represented as a transition graph. Figure 4 depicts an exemplary transition graph. There is one dedicated start state (*Start*), at least one end state (*End*) and always one active state. States are connected with each other by arcs. For example, state S_1 is connected with S_2 by arc B. A finite-state automaton parses a finite string of symbols stored on a memory tape, i.e., input tape. The finite set of symbols that can be parsed by an automaton is called the alphabet. The automaton sequentially parses each symbol on the input tape. According to the transition function the automaton will transition from state to state. If the end state becomes active (after the last symbol on the tape has been processed) the input string is successfully parsed. A finite-state automaton can only parse context free

grammars whereas natural language, a commonly used modality in multimodal interfaces, consists of a context sensitive grammar (R_{syn}). Input is only processed in the order in which it is placed on the tape (R_{uns}) and only sequential temporal relations can be considered (R_{tmp}). It is not possible to process multiple hypothesis (R_{prob}) since only one state can be active at any given time. Creating new symbols, each annotated with every possible confidence value, for each symbol in the initial alphabet could solve this issue. However, since confidence values are conceptually element of $\mathbb{R}$, this leads to an infinite number of states and is thus disregarded. Lastly, the interaction context (R_{sem}) can also only be considered, if it is encoded in the automaton's alphabet in a similar fashion. A respective implementation however is highly impractical. There is no possibility to provide feedback (R_{fb}) since finite state automaton only have one memory tape used for input. Altogether, finite-state automata are less suited for semantic fusion (see Table 1).

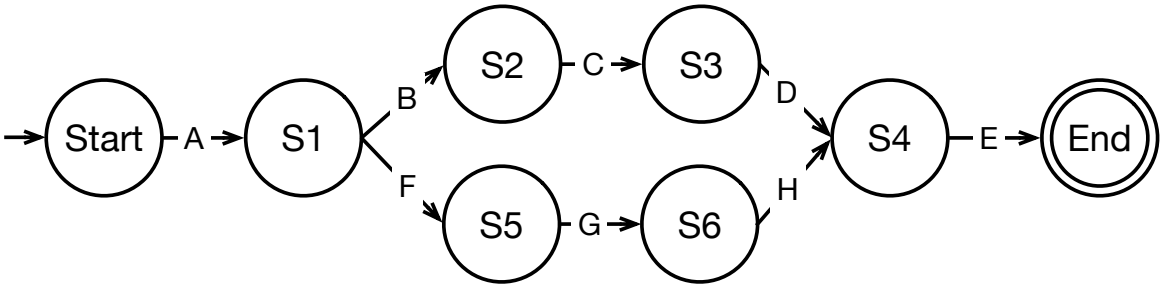

Figure 4. Illustration of a transition graph. The graph is composed of states (`S1`, `S2`, `S3`, etc.) and arcs (`A`, `B`, `C`, etc). A dedicated start state *Start* represents the starting point while a dedicated end state `End` represents a completely parsed input.

In contrast, a finite-state transducer is a finite-state automaton with two memory tapes: An input and an output tape. It can map symbols of one alphabet to symbols of a second alphabet and store them on its output tape. Thus a finite-state transducer is capable of providing continuous feedback (R_{fb}) in contrast to a finite-state automaton. Finite-state transducer have been used for multimodal fusion in the past [34,35].

Augmented Transition Networks have been initially introduced to analyze natural language [50] and later to perform multimodal fusion [15]. They inherently foresee data storage by introducing state specific *Registers*. Primarily, registers were designed to move input throughout the graph in order to be able to parse context-sensitive grammars (R_{syn}). Arcs can be guarded with additional conditions and can perform dedicated functionality when traversed. Arc functions can be used to provide feedback after each processed input (R_{fb}). They can also be used to implement semantic integration (R_{sem}). Since, the ATN was not initially part of a RIS, it does not explicitly support it. Conditions and registers are means to support different temporal relations between input (R_{tmp}), by storing timestamps in registers and checking them with conditions. However, similar to the finite-state automaton and transducer, there can only be one state active at once. Hence, it does not support probabilistic input (R_{prob}). Additionally, there is no possibility to handle chronologically unsorted input (R_{uns}).

The temporal Augmented Transition Network [16] has been proposed as an extension to the ATN. As such it fulfills the same requirements as the ATN. It has been introduced specifically for RIS and as such implements an interface for performing semantic integration during fusion (R_{sem}). In addition, it provides explicit support for defining temporal relations between inputs (R_{tmp}). However, it still does not handle chronologically unsorted input (R_{uns}) or supports multiple hypothesis (R_{prob}).

In conclusion, we identify a lack amongst procedural fusion methods with regard to handling probabilistic and chronologically unsorted input (R_{prob} & R_{uns}) which is highlighted with a red boarder in Table 1.

3. Results

Our results are twofold. First, we present the Concurrent Cursor concept. It provides a solution for the two major shortcomings of procedural fusion methods: the analysis of probabilistic (R_{prob}) and chronologically unsorted input (R_{uns}). These requirements presuppose that each input is annotated with a respective confidence and timestamp value. Our concept also provides a solution to effectively provide feedback (R_{fb}) despite the challenging consequences of this kind of input. Second, we present a reference implementation, the *Concurrent Augmented Transition Network* (CATN). It takes four design decisions that are left open by the Concurrent Cursor concept: the choice of a procedural method, the choice of a description language, a measure for handling temporal relations between input, and an access scheme to the interaction context. Altogether, the CATN thus fulfills all identified requirements. Finally, we show how the CATN utilizes the Concurrent Cursor concept to perform semantic fusion using the example of the introduced use case and present a comparative benchmark.

3.1. Concurrent Cursors

The Concurrent Cursor concept introduces so called *cursors*. A cursor is a pointer indicating a currently active state in a procedural fusion method. For example, C_1 in Figure 5 represents a cursor pointing to the currently active start state *Start*. Cursors do not move from one state to another, but spawn a child that is positioned on the target state. The original cursor remains on its current state. For instance, if a transition over A is performed, the child C_{1-1} of cursor C_1 is created. The child is positioned on state S_1 while C_1 remains on *Start*. As a result, multiple states can be active at once and strands of cursors are created while processing input. A strand is interpreted as one viable path throughout the graph. All possible strands form a tree representing all possible parsings. Its leafs represent guesses of the fusion method. This way of copying cursors lays a foundation for interpreting hypotheses and different combinations of hypotheses, since multiple states can be active at the same time. In order to sort these n-best guesses by their confidence, every cursor has to store its own confidence. This value is based on the confidences of a cursor's processed inputs. Different mathematical calculations can be used to calculate a cursor's confidence, e.g., the median value of the processed inputs' confidences. In summary, each cursor is copied to the target state when performing a transition and has its own confidence.

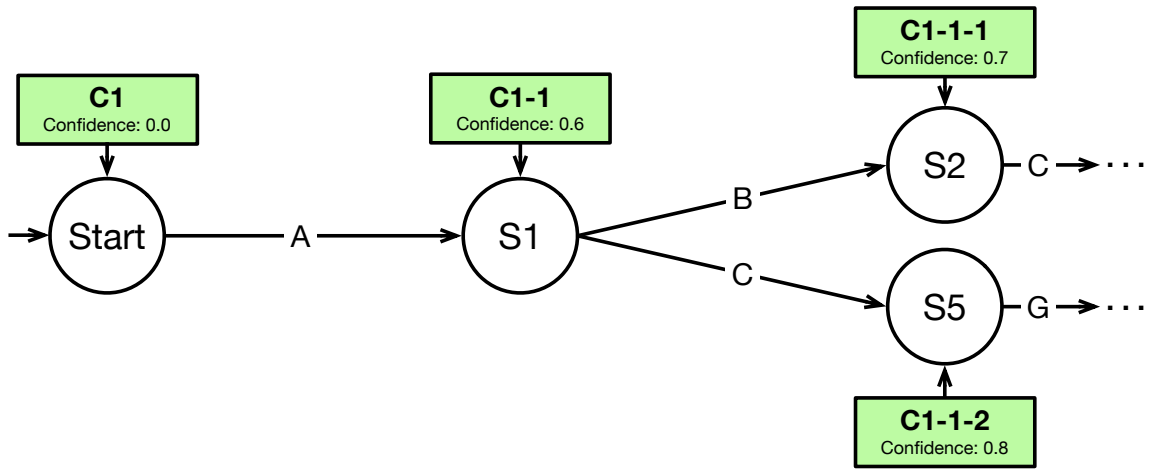

Figure 5. C_1, C_{1-1}, C_{1-1-1}, and C_{1-1-2} are cursor that each point to an active state. They hold their own confidence. A cursor's confidence depends on the input it has processed. In the depicted graph, two cursor strands have been formed: C_{1-1-1} and C_{1-1-2}. Each strand represents a possible recognition result. They can be sorted by their confidence to get n-best guesses as output.

To illustrate the concept, consider the following example (see Figure 5): Four states (*Start*, S_1, S_2, and S_5) are active and two cursor strands have been formed: C_{1-1-1} represents the strand from

Start over S_1 to S_2 with a confidence of 0.7. C_{1-1-2} represents the strand from *Start* over S_1 to S_5 with a confidence of 0.8. Both cursors are sorted by their confidence. As a result, the most confident recognition result would be the inputs processed by C_{1-1-2}.

In the following we describe how the Concurrent Cursor concept allows to process chronologically unsorted input and consider temporally valid combinations of hypotheses. To this end, cursors use an extended *two dimensional input tape*. Recognizers usually provide timestamps indicating when a specific event took place. For example, when a user uttered a word or performed a gesture. Input is sorted on the tape depending on this timestamp instead of its arrival at the fusion method. In contrast to traditional one dimensional input tapes, which only store input sequentially, our input tape can store multiple inputs for one *time cell* in parallel to represent alternative hypotheses. Time cells represent a specified time interval that further discretize timestamps to cluster hypothesis.

Figure 6 depicts such an input tape. At time cell T_0 no input has been received by the fusion method implementation while input I_1 occurred at T_1. Two hypothesis I_2 and I_3 have been recognized at T_2 and sorted into the tape. Cursors keep track of which inputs they already analyzed by pointing to respective time cells on the input tape. For example, in Figure 6 cursor C_{1-1} points to state S_1 and processed input I_1 at time cell T_1. Cursor C_{1-1-1} points to S_2 and processed I_1 at T_1 and I_2 at T_2. If a new input I_4 is sorted into the tape at T_1 (see Figure 6, red color) all cursors that already processed input at or later then T_1 do not have to process I_4. For them it is not temporally valid to process input which, from their point of view, occurred in the past. The only eligible cursor to process I_4 is C_1. If I_4 passes transition `A`'s conditions, a new child cursor C_{1-2} is created. C_{1-2} points to state S_1 in the graph and to I_4 at T_1 on the tape. After processing this input the cursor has to check if newer input is available on the tape which could form another valid multimodal command. In the example of Figure 6, two newer inputs are available for the newly created cursor C_{1-2}: I_2 and I_3 at T_2. If these inputs lead to new transitions, two new children are created from cursor C_{1-2}: C_{1-2-1} and C_{1-2-2}.

The described copying pattern of cursors in combination with the two dimensional input tape provides the means to process probabilistic and chronologically unsorted input (R_{prob} & R_{uns}). In addition, it reduces the computational complexity by reducing the number of triggered transitions for each new input by means of the described temporal validity check.

Providing proper feedback (R_{fb}) is exacerbated since new hypotheses may cause the current best guess to change. To cope with this issue, the concept defines dedicated feedback functions for each arc (see Figure 7, green arrow). The arc's condition determines if a transition will be performed while the corresponding function is executed when it is performed. The feedback function will be executed after a transition has been performed. Besides the time of execution, feedback and function differ in one central aspect. Arcs might be traversed several times by different cursors resulting in multiple function calls. The user would receive a large number of feedback about every possible guess a fusion method is considering at the moment, if this function type is used for feedback. In contrast, feedback is only executed if the arc is traversed by the current most confident cursor, i.e., the cursor with the highest confidence. If the most confident cursor changes during processing previously executed feedback has to be revertible (R_{fb}). Reverting feedback can not be automatically performed by the fusion method, since arbitrary functionality can be implemented in the feedback function. The developer has to specify dedicated undo feedback functions which negate the respective feedback (see Figure 7, red arrow). All transitions made by the old most confident cursor have to be traced back until the first common parent between new and old most confident cursor. The feedback of these transitions has to be undone by executing the respective undo feedback functions. Upon reaching the first common parent, the feedback can be sequentially executed from the parent to the new most confident cursor.

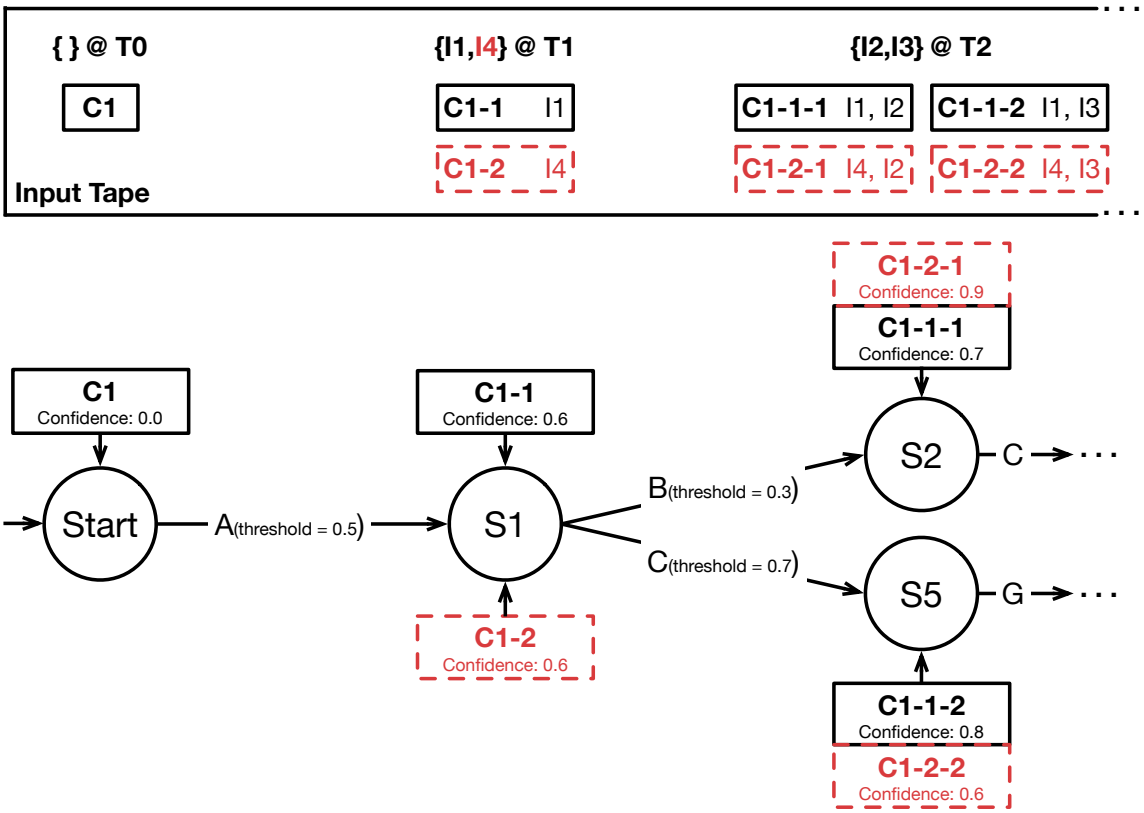

Figure 6. The extended two dimensional *Input Tape* (top) and a transition graph (bottom). The input tape does not only store input in the chronological order in which it occurred, but is also capable of storing multiple hypotheses for a single time cell. For example, a recognizer may have provided two different hypothesis I_2 and I_3 for one input that is stored on the tape at time cell T_2. Besides pointing to a state in the transition graph, cursor point to time cells on the tape and store which input they already processed. For instance, cursor C_{1-1-1} points to state `S2` and to T_2 on the input tape. It already processed input I_1 and I_2.

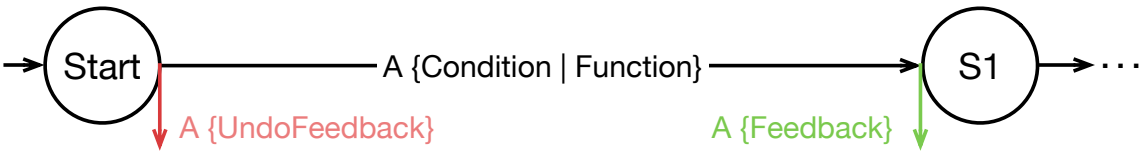

Figure 7. Two states *Start* and S_1 which are connected by arc A. The arc contains a guarding `Condition` which determines if a certain input leads to a transition and a `Function` which is called during the transition. The Concurrent Cursor concept introduces two additional functions for each arc: `Feedback` and `UndoFeedback`. In contrast to `Condition` and `Function`, `Feedback` is not executed for every transition but only for transitions made by the most confident cursor. This makes it especially suitable to provide feedback. `UndoFeedback` can be used to revert previously given feedback.

Figure 8 depicts such a procedure. A new input has been processed by the fusion method implementation. It leads to a change of the most confident cursor from C_{1-1-1} to C_{1-1-2}. Their common parent is C_{1-1}. First, the undo feedback function of transition B is triggered. Second, the transition C's feedback function is executed for cursor C_{1-1-2}.

In summary, the Concurrent Cursor concept provides effective means to analyze probabilistic and chronologically unsorted input and satisfies R_{prob} and R_{uns}. Cursors allow for multiple states to be active at the same time and contain their own confidence. This is essential for the fusion method to represent sorted n-best recognition results caused by analyzing multiple hypotheses and their

combinations. Cursors keep a reference to their processed input on a two dimensional, chronologically sorted input tape. This enables cursors to properly parse new input in the correct order, independent of their time of arrival. Providing proper feedback is supported by the Concurrent Cursor concept, since the fusion methods best guess can be communicated to the application continuously and be reverted (R_{fb}), if the best guess changes.

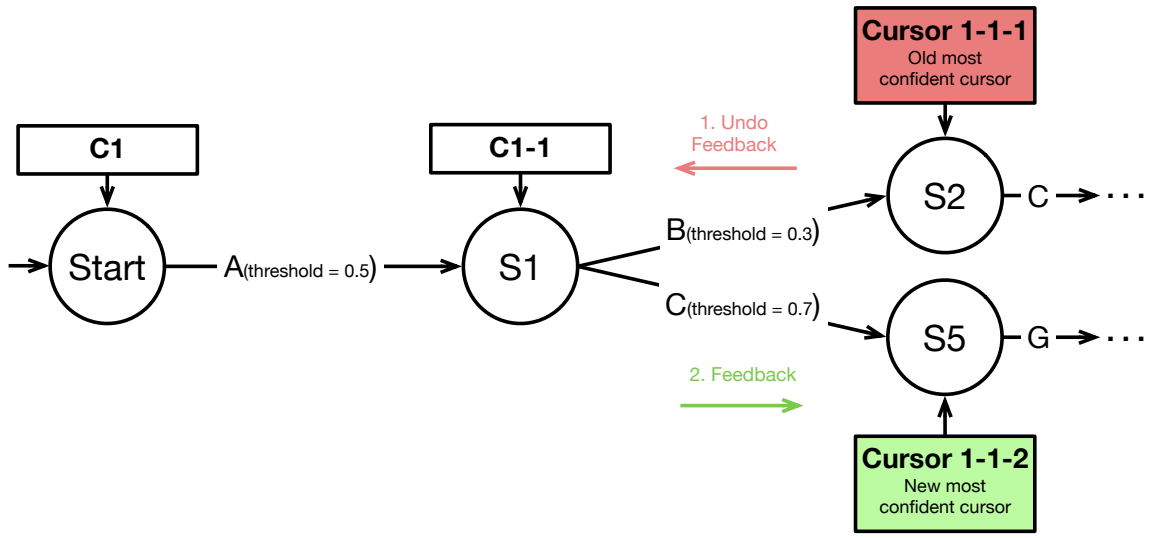

Figure 8. Two cursors in a transition graph: C_{1-1-1} and C_{1-1-2}. The fusion method's most confident cursor changes from C_{1-1-1} (red) to C_{1-1-2} (green), after processing an input. To this end the previously provided feedback of C_{1-1-1} is reverted by means of the `UndoFeedback` function, and the feedback of C_{1-1-2} is provided.

3.2. Reference Implementation

Our reference implementation is called *Concurrent Augmented Transition Network* (CATN). In the following, we describe the implementation of the Concurrent Cursor concept and detail the four design decisions that are left open by the Concurrent Cursor concept: The choice of a procedural method, the choice of a description language, a measure for handling temporal relations between input, and an access scheme to the interaction context. Altogether, the CATN thus fulfills all identified requirements. Since the Concurrent Cursor concept is an extension to a procedural fusion method, we first elaborate this decision.

3.2.1. Design Decision: Augmented Transition Network

We chose to base the CATN on an augmented transition network which receives input by means of events. An ATN foresees data storage in form of state-specific registers. The CATN implements registers cursor-specific to comply with the Concurrent Cursor concept. That is, each cursor comprises its own set of registers. The content of a cursor's register is copied to the register of its children. A cursor's registers can thus be used to fulfill two essential requirements. On the one hand, they can store information about the syntactic structure of each particular guess, enabling the parsing of context-sensitive grammars (R_{syn}). On the other hand, they can store state and behavior references to permit semantic integration during fusion (R_{sem}).

A further benefit of basing on an ATN is its inherent recursiveness. Reoccurring recognition tasks, e.g., parsing a noun phrase, can be externalized into separate networks. These networks can be integrated into the main network of the ATN by means of dedicated subroutine calls. A concrete example of this functionality is provided in Section 3.3. Dividing a large network into several small ones increases reusability and modifiability.

Besides normal and sub arcs, we implemented dedicated arcs for semantic integration called `EpsilonArcs`. An epsilon arc is not triggered with new input, but is automatically triggered if a cursor reaches its point of origin. While semantic integration can be performed in any arc, encapsulating this functionality in dedicated reusable arcs proofed to be easier to maintain.

3.2.2. Concurrent Cursor Implementation

Both cursors and the two dimensional input tape are implemented according to the Concurrent Cursor concept and utilized in the CATN's parser. The parser, as well as noteworthy optimizations, are described in the following. Figure 9 (top flow chart) depicts a flow chart illustrating the designed algorithm. One processing loop is triggered by an input event, which is first sorted in the input tape depending on its timestamp.

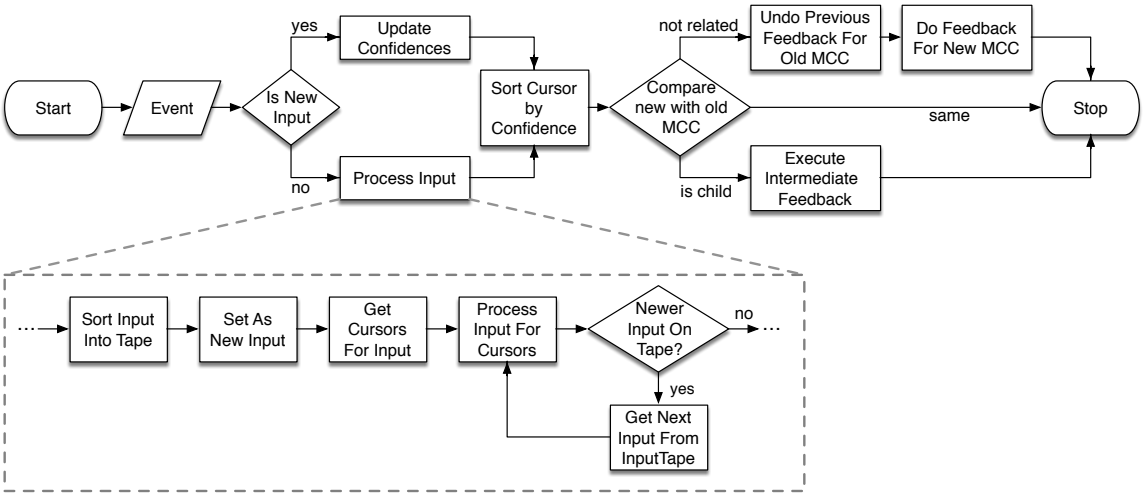

Figure 9. An illustration of the concurrent Augmented Transition Network (CATN) parser's algorithm. The upper flow chart outlines the behavior of the CATN's parser upon receiving input. The bottom flow chart is a more in depth description of the `Process Input` step of the upper flow chart. The abbreviation MCC stands for most confident cursor. For a more detailed description of the algorithm refer to the text.

The parser checks if the respective time cell already contains an input with equal value. That is, the confidence of a previously parsed input changed, e.g., the automated speech recognizer updated the confidence for having recognized "*chair*" at a certain timestamp. If this is the case, the confidence of all cursors that already processed this input is updated with the new input's confidence. This step is optional but aids to optimize performance by not unnecessarily introducing new cursors to the CATN. It is based on the premise that later hypotheses are more reliable.

If the input is new, it is being processed. This may lead to the creation of multiple new cursors. If new cursors are created, the new most confident cursor will be calculated. Three different possibilities are considered. Firstly, the old most confident cursor remains the new most confident cursor, in which case no feedback has to be provided. Secondly, the new most confident cursor is a child of the old one. In this case, the feedback function of all arcs between the old most confident cursor's state and the new one's has to be executed. Thirdly, the new most confident cursor is not a child of the old one. In this case the previously provided feedback has to be undone by executing the dedicated undo-feedback functions and the new feedback has to be provided.

As an additional optimization, the CATN can be reset, e.g, after a successfully parsed command has been executed. A reset deletes all existing cursors and spawns a new cursor on the start state. The parser waits for the next input. On top of that we implemented an auto reset for the CATN. The auto reset is triggered if the CATN does not receive input in a configurable time frame. It avoids that the CATN's parser gets stuck in an incomplete command.

Figure 9 (bottom) illustrates the *Process Input* step in detail. The new input is sorted into the input tape. Next, all eligible cursor are identified. A cursor is deemed eligible if its position on the input tape precedes the new input. Additionally, we implemented a max time delta for how far in the past cursors can be eligible. This is an optimization and not part of the concept. However, it reduces the number of active cursors in practice which decreases processing time. It is based on the assumption that natural input occurs within a certain time interval to each other. Eligible cursors process the new input by checking the conditions of outgoing arcs from the states to which they are pointing to. If a condition is satisfied, the arc's function is executed. Its feedback functions will be collected in the parser and maybe provided later depending on the new most confident cursor calculation step. A child of the cursor is created by cloning it. The child's confidence is calculated and its pointers updated to the new state and position on the tape. Finally, the parser checks for newer input on the tape. If newer input exists it will be processed by the newly created children, if not the *Process Input* step is finished.

3.2.3. Design Decision: The Programming Language Scala

The *Scala* programming language has been chosen as a description language for the CATN. Scala combines both object-oriented and functional paradigms. This fosters the definition of concise application programming interfaces (APIs), especially comprising callbacks, and is beneficial for addressing a large number of developers. In addition, Scala's syntax is very flexible which greatly facilitates the use of a description languages. For example, Scala allows to omit the dot operator or the parentheses in certain cases. Listing 1 showcases our Scala based description language for defining the CATN's transition graph. Line 1–3 define the transition graph's composition. A dedicated start state *Start* is connected with arc A to the state S_1 (line 1). S_1 is connected with arc B to the end state *End* (line 2). The end state is defined in line 3 while line 4 depicts the definition of the arc A. It's condition `isA`, function `doA` and feedback `feedbackA` are Scala functions which can directly be passed to the description language. This description language fosters rapid prototyping and helps non-experts to effectively design multimodal interfaces.

Listing 1. Example of a concurrent Augmented Transition Network (CATN) defined with our description language. By omitting the dot operator and parentheses, the CATN can be defined in an easily readable manner. For example, the `StartState` method of the object `create` in line 1 accepts a string as parameter and yields an object with a method `withArc`.

```
1 create StartState "Start" withArc "A" toTargetState "S1"
2 create State      "S1"    withArc "B" toTargetState "End"
3 create EndState   "End"
4 create Arc "A" withCondition isA andFunction doA provideFeedback feedbackA
5 create Arc "B" withCondition isB andFunction doB provideFeedback feedbackB
```

3.2.4. Design Decision: Network Abstraction Constructs

We include network abstraction constructs in the definition language to provide a convenient measure for handling temporal relations between input. An ATN's conditions and registers already provide means to support this requirement, by storing timestamps in registers and checking them with conditions. However, the repeated definition of such constructs is cumbersome. Thus we utilize constructs in our description language to abstract this kind of reoccurring network configurations.

Split-Merge is an example of an network abstraction construct in our description language. It allows the definition of more complex temporal relations between input (R_{tmp}). Special *Split* and *Merge* states can be defined. A split state has to be connected to a merge state with at least two transitions in parallel. In order for a cursor to transition to the merge state, all transitions from split to merge have to be performed. These transitions have to satisfy an additional *Merge-Condition*, e.g., a temporal condition. Listing 2 illustrates the description language for defining a split-merge. Both arcs `A` and

`B` have to be traversed within the `maxTimeDelta(500L)` condition. In other words, the CATN has to receive appropriate input within 500 milliseconds next to each other. A dedicated `merge` function can be freely implemented (line 4). The `merge` function is automatically executed if both `A` and `B` are traversed in the `maxTimeDelta(500L)` condition. This function can be used, for example, to clean the cursor's registers or perform semantic integration. The temporal condition for the split-merge can be freely chosen. Allen's interval algebra for representing relation between time intervals [65] provides helpful guidelines for defining suitable conditions. The CATN processes a split merge by mapping this high level definition to the low level functionality of the ATN.

Listing 2. An example code excerpt showcasing our network abstraction design decision *Split-Merge*. A dedicated split state `Split` is defined in line 1. Two arcs, `A` and `B`, connect `Split` with state `Merge` in line 2 and 3. Line 4 defines the dedicated merge state `Merge` with the on merge function `merge`.

```
1 create SplitState "Split" withCondition maxTimeDelta(500L)
2 andArc "A" toTargetState "Merge"
3 andArc "B" toTargetState "Merge"
4 create MergeState "Merge" withOnMergeFunction merge
```

Figure 10 illustrates this mapping. The left transition graph represents the in Listing 2 defined network. In a preprocessing step it is mapped to the transition graph on the right. Two new states $AS1$ and $AS2$ are automatically generated by the CATN and placed between `Split` and `Merge`. All permutations of the two transitions A and B are thus represented. The time condition is added to the conditions of all arcs between `Split` and `Merge`. Similarly, a function is added to the respective arcs that automatically stores the inputs' timestamps in the cursors' registers. These timestamps can then be evaluated by the time condition to conclude a successful split-merge. Utilizing the network abstraction design decision the *Split-Merge* construct provides a solution for defining more complex temporal relations between input (R_{tmp}).

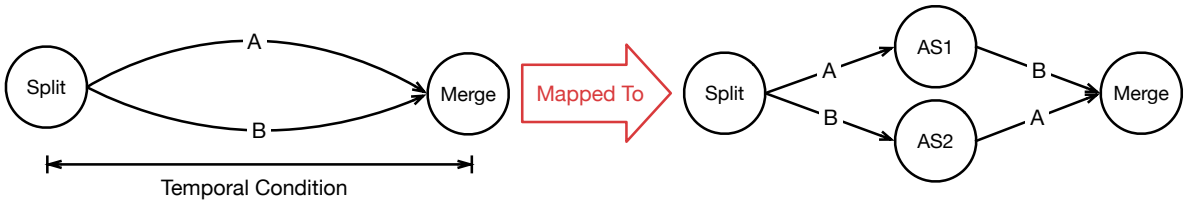

Figure 10. The left transition graph visualizes the network abstraction for supporting different temporal relations between input. A temporal condition checks if both transitions A and B are performed within a predefined temporal relation between each other. The CATN maps this abstraction to a parsable transition graph depicted on the right.

3.2.5. Design Decision: Semantics-based State- and Behavior-Management Techniques

We utilize semantics-based state- and behavior-management techniques [10,20] to realize our access scheme to the interaction context. Entities are used as state representing objects containing a set of numerical and symbolic properties that are semantically grounded and can dynamically change during simulation. The chair in the running use case, for example, would be represented as such a *semantic entity* [20,56], containing grounded properties like type, color, position, and orientation. The user and its position in the virtual environment or environmental information, such as the surrounding light level, can be represented with semantic entities as well. Most importantly however, these techniques incorporate a concept to semantically describe behavior as complement to the application state representation. State and behavior queries can be performed by means of semantic descriptions called *semantic queries*. They support the resolution of references to (virtual) objects by permitting to query the application state for entities by a logical combination of desired present and

past properties and property values. After "*... [deictic gesture] that green chair ...*" has been analyzed an entity could be queried for that is a chair, has a color property that equals *green*, and is located within a certain area. Such a query would yield the above described semantic entity representing the chair in the virtual environment. This entity can also be altered by the CATN during input analysis if required. Semantic queries likewise foster semantic integration. Since the goal of many multimodal user interactions is to trigger a certain action the application can perform, semantic-level fusion has to match uttered multimodal commands to available actions. Behavior queries support in exactly this task and yield so called *grounded actions* that consist of a set of preconditions, a set of parameters, and a set of effects. A semantic validation can be performed using the action's meta data by checking if the information contained in the multimodal utterance fulfills the action's preconditions and covers the action's parameters. In the use case, the analysis of the utterance "*Put ...*" could query a respective grounded action *collocate* that takes an object as well as a destination (parameters) and requires the object to be moveable (preconditions).

We propose to integrate this semantics-based state- and behavior-management into procedural fusion as follows: Registers are realized as semantically grounded property sets just like semantic entities. Thus transition-conditions and -functions can exploit introspection. Semantic entities, grounded actions, as well as semantic queries complement the means for the implementation of transition. They are highly beneficial for resolving references to (virtual) objects and for realizing semantic integration. Transition functions can map input to semantic entities, grounded actions, and semantic queries to store them in their register.

Listing 3. An example code excerpt for an arc's function which performs semantic integration after the user uttered the word "*chair*" with a preceding deictic gesture. A semantic query is used for identifying and subsequent accessing entities in the application state. In this example, the query returns an entity, if one exists, which possess a `Semantics` property with value `Chair` and intersects with the user's pointing `Ray` at a specified time. Grounded properties are highlighted in blue.

```
1  def resolveNP(input: Event, register: SValSet) {
2  val ray = register.get(Ray).value
3  val timestamp = register.get(Timestamp).value
4  val word = input.get(Token).value // "chair"
5  val noun = Lexicon.nouns(word)
6  val pointedAtChair  = HasProperty(Semantics(noun.assocEntity)) and
7  PointedAt(ray) at timestamp
8  val entity = Get the pointedAtChair
9  if(entity.isDefined) register.put(entity)
10 }
```

Listing 3 depicts an arc's function which resolves a noun phrase accompanied with a deictic gesture using a semantic query. The goal of this function is to find the semantic entity `Chair`. Both the direction `ray` of the pointing gesture and the `timestamp` at which the user performed the gesture can be retrieved from the register (line 2–3). The uttered word *"chair"* is retrieved from the event that triggered the transition (`input`) (line 4). We use a `Lexicon` [66] to map the word *"chair"* to a respective `noun` which is associated with a type of entity (line 5). This data is used to parameterize the query (line 7–8) which is then executed (line 9). The timestamp is of particular importance for identifying the respective entity in a dynamically changing environment. The `PointedAt(ray) at timestamp` part of the query performs a lookup in the application state history to check if the ray intersected an entity at `timestamp`. If the query yields a result, the entity is stored in the register (line 10) and can later be used to invoke a resolved action.

The semantics-based state- and behavior-management design decision allows multimodal systems to use entities and actions without requiring explicit references (R_{sem}). This decouples in terms of

data sinks and sources as well as of utilization of application and system functionality. It facilitates the reuse of fusion method configurations in other contexts or applications, as long as the required application state and behavior representation elements exist. Thus an universal multimodal interface, comprised of basic commands like creation, collocation, and deletion, could be rapidly added to any application that is realized with an interactive system supporting this design decision.

3.3. Use Case Implementation

In this section, we present a concrete CATN configuration capable of recognizing the example interaction introduced in Section 2.1: "*Put [deictic gesture] that green chair near [deictic gesture] this table.*" Figure 11 illustrates the respective CATN's transition graphs. To represent parts of speech we use the Penn Treebank POS tagset [67]. The primary transition graph (see Figure 11, top transition graph) recognizes a verb (VB), a noun-phrase (NP), a preposition (IN), and another noun-phrase (NP) in a sequential order. A transition of the final arc `CMD` implies that the command has been successfully recognized and executes the respective action. The NP arc is a sub arc which points to the secondary transition graph (see Figure 11, bottom transition graph). In order for a cursor to traverse from S_1 to S_2 or S_3 to S_4, it has to traverse through the entire secondary graph first. A noun phrase starts with a split-merge consisting of a demonstrative (DT) and a deictic gesture (dG). The merge is followed by a noun (NN) and finally a dedicated arc for semantic integration ($resolveNP$). $resolveNP$ provides feedback by visually highlighting the resolved entity. Additionally the arc ADJ allows for an optional number (0-n) of adjectives.

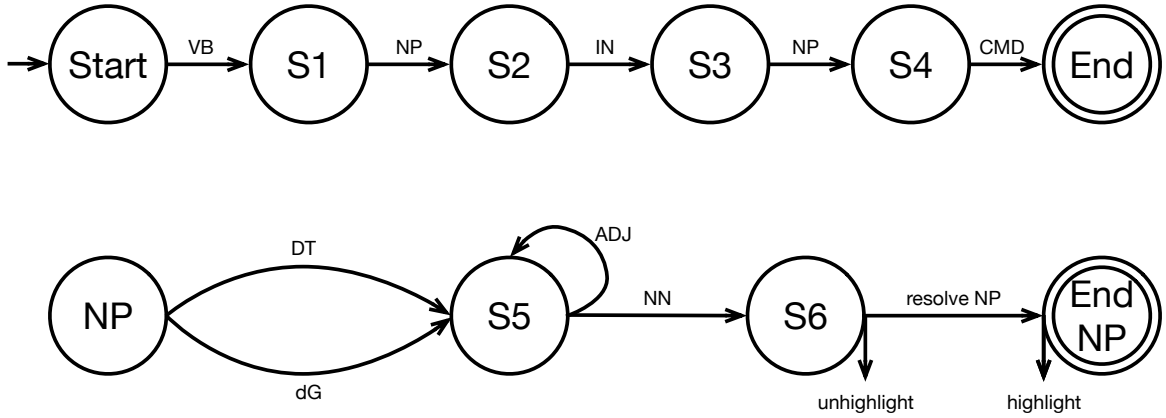

Figure 11. Two CATN transition graphs for recognizing the multimodal command introduced in Section 2.1. The upper graph defines the overall structure of the command. The lower graph is a dedicated sub graph for recognizing noun phrases accompanied by a deictic gesture. The lower graph is utilized twice in the upper graph, between S_1 and S_2, and S_3 and S_4. Arcs are named after the input they process and include a respective condition and function, e.g., VB accepts a verb. The `resolveNP` arc performs semantic integration and provides revertible feedback in form of a visual highlighting effect.

Listing 4 showcases the description language code to define the introduced CATN configuration. Line 1–6 define the upper graph in Figure 11, while line 8–15 define the lower graph. The Arc `VB` is created in line 17–18. Since the description language is internal, Scala functions can be directly passed to it. `VB`'s condition `isPartOfSpeech[Verb]` firstly verifies whether the input is an input from the speech recognizer. Secondly, it checks if the recognized token is of a certain part of speech, i.e., a verb. The arcs function `saveAs[Verb]` stores the verb in the cursor's register for future reference. Arcs for recognizing the remaining speech tokens, i.e., `VB`, `IN`, `DT`, `ADJ`, and `NN` are all defined similarly. Line 21. illustrates how the `dG` arc for parsing a pointing gesture is defined. Its condition `isGesture[Pointing]` verifies whether the input is a pointing gesture. The `saveAs[Pointing]` function stores the pointing ray semantically annotated in the cursors register.

If a cursor reaches `S6` in the `NP` graph (line 14), a noun phrase accompanied with a gesture has been successfully recognized. Relevant information is stored in the cursor's registers and can be retrieved from it in the `resolveNP` epsilon arc. Its condition `entityExists` (line 24) checks if an entity described by the parsed noun-phrase and accompanying deictic gesture exists using semantic queries. If such an entity exists in the application state, the arc's function `resolveNP` (line 25) stores a reference to the entity in the cursor's register. Finally, the `resolveNP` arc's feedback function (line 26) highlights the entity if the cursor currently is the most confident one. An additional undo feedback is defined in case the most confident cursor changes during fusion and the wrong entity has been highlighted (line 27). The *CMD* arc (line 29) is implemented as an epsilon arc as well. It does not need a condition or function. If a cursor reaches state `S4` (line 5) a command has been successfully recognized and all necessary information is stored in the cursor's registers. It retrieves the appropriate action, i.e., `colocate` from the application's behavior by means of a semantic query. The action is parameterized with the content of the cursor's register, i.e., the semantic entity `Chair` and `Table`, and is executed.

This generalized approach to multimodal interface design leads to easy extensibility. Simply adding more determiner, adjectives, nouns and verbs to the lexicon is sufficient to recognize a multitude of commands, e.g., "*Move [deictic gesture] that big vase to [deictic gesture] this counter.*" or "*Place [deictic gesture] that small yellow thing under [deictic gesture] that table.*".

Listing 4. A description language excerpt for creating the CATN depicted in Figure 11.

```
1  create StartState "Start" withArc        "VB"  toTargetState "S1"
2  create State      "S1"    withSubArc     "NP"  toTargetState "S2"
3  create State      "S2"    withArc        "IN"  toTargetState "S3"
4  create State      "S3"    withSubArc     "NP"  toTargetState "S4"
5  create State      "S4"    withEpsilonArc "CMD" toTargetState "End"
6  create EndState   "End"
7
8  create SplitState "NP"    withCondition maxTimeDelta(500L)
9  andArc  "DT"  toTargetState "S5"
10 andArc  "dG"  toTargetState "S5"
11 create MergeState "S5"    withOnMergeFunction merge
12 withArc "NN"  toTargetState "S6"
13 andArc  "ADJ" toTargetState "S5"
14 create State      "S6"    withEpsilonArc "resolveNP" toTargetState "EndNP"
15 create EndState   "EndNP"
16
17 create Arc "VB" withCondition isPartOfSpeech[Verb]
18 andFunction saveAs[Verb]
19 /*...*/
20
21 create Arc "dG" withCondition isGesture[Pointing]
22 andFunction saveAs[Pointing]
23
24 create Arc "resolveNP" withCondition entityExists
25 andFunction resolveNP
26 provideFeedback highlight
27 withUndoFeedback unhighlight
28
29 create ARC "CMD" withFeeedback executeCommand
```

3.4. Benchmark

The Concurrent Cursor concept provides an extension to procedural semantic fusion methods. It fulfills the two fundamental requirements handling probabilistic (R_{prob}) and chronologically unsorted input (R_{uns}). The concept introduces concurrent cursors and a two dimensional input tape that could introduce a performance overhead. This section presents a fundamental comparative benchmark that provides basic insights into the impact of the Concurrent Cursor concept on the performance of ATNs and thus into the practicability of our contribution.

3.4.1. Concept

The benchmark measures and compares the mean processing time of both the CATN and a state of the art ATN-based approach implemented without the Concurrent Cursor concept (GENERIC ATN) on a fabricated dataset D. D consists of two different types of inputs A and B, e.g., speech tokens and recognized gestures. The benchmark generates n guesses for each input, e.g., A_1, A_2 as two guesses for A, to consider probabilistic input (R_{prob}). Timestamp t_a is assigned to inputs of type A and t_b to B, with $t_a < t_b$. Additionally, a randomly generated confidence is assigned to each input.

The benchmark samples the mean processing time of the CATN and GENERIC ATN for $n \in [1, 2, ..., 25]$. For $n = 2$, for example, the following input dataset is constructed: $D_2 = [A_1, A_2, B_1, B_2]$. The input dataset is randomized before it is passed to the fusion methods, to simulate chronologically unsorted input (R_{uns}). The CATN natively supports this kind of input. The GENERIC ATN approach is required to chronologically sort the passed list again before processing.

Figure 12 illustrates the networks that the benchmark uses for parsing D. The main difference between the CATN and the GENERIC ATN approach is that the CATN can process all D_n using one configuration due to the Concurrent Cursor concept. A single GENERIC ATN, however, conceptually provides only one active state and is thus incapable of considering chronologically plausible combinations of guesses. To be nevertheless able to consider probabilistic input, a set of artificially constructed networks is generated: $[(A_1, B_1), (A_1, B_2), ..., (A_n, B_n)]$.

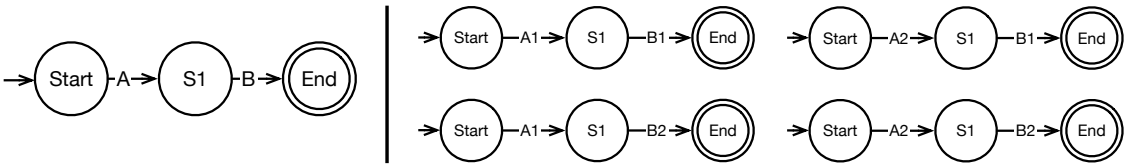

Figure 12. The CATN utilized for all D_n is illustrated on the left. The GENERIC ATN approach requires to create a network for each permutation of guesses. The four GENERIC ATNs that are generated to process $D_2 = [A_1, A_2, B_1, B_2]$ are depicted on the right.

Each network (CATN and GENERIC ATN) consists of three states and two arcs. The arcs' conditions are setup to accept the guesses, while the arcs' function store the value, timestamp, and confidence of each input in their registers. In the case of the CATN, this information about all parsed inputs are thus available in registers that belong to cursors on the end state. To mimic this behavior with the GENERIC ATN, the GENERIC ATN's arc functions have to retrieve value, timestamp, and confidence from the previous state's register and add them to the current state's register.

The benchmark has been performed on a *Windows 10* workstation with an *Intel Core I7-8700k @ 3700GHz* processor, *16GB DDR3* ram, and an *Nvidia Geforce GTX 1080Ti* graphics card. Both fusion methods have been implemented with the programming language *Scala* v. 2.11.8 and ran in the *Java Virtual Machine*(JVM) v. 1.8.0_181. For each $n \in [1, 2, ..., 25]$ 50,000 warm up cycles have been run before a total of 10,000 measurement cycles. The warm up cycles reduce the impact of the JVM's runtime optimizations on the measurement cycles.

3.4.2. Results

The obtained results are presented in Figure 13. They show that the CATN is comparable in performance to state of the art ATN-based approaches until $n = 12$, with a peek difference at $n = 8$ of 0.099 milliseconds. For high numbers of n the CATN outperforms its comparison partner. At $n = 25$ the CATN is in average 2.830 milliseconds faster than the ATN.

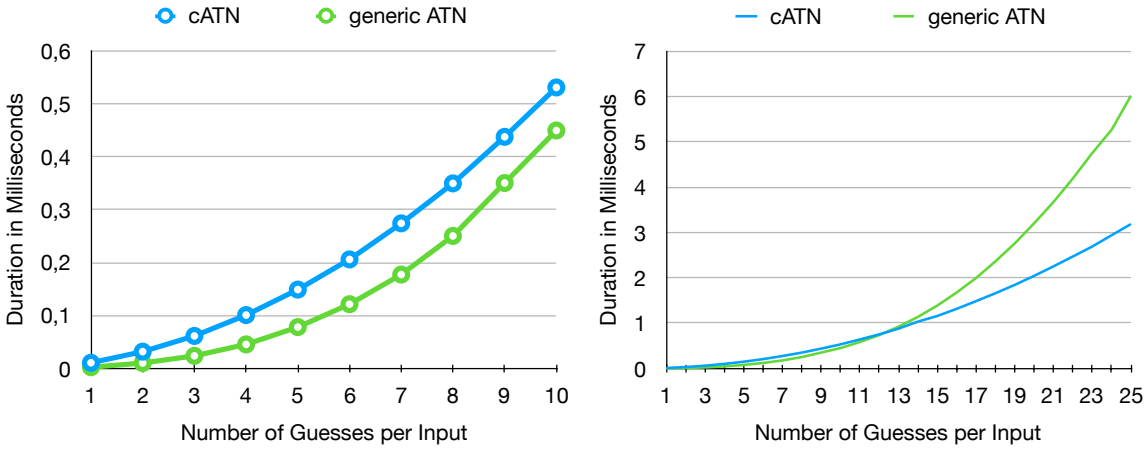

Figure 13. The results of the comparative benchmark. The graphs on the left depict the mean processing duration for $n \in [1, 2, ..., 10]$ guesses per input type A and B. The graphs on the right depict the mean processing duration for $n \in [1, 2, ..., 25]$ guesses per input type A and B. At $n = 13$ guesses per input the CATN outperforms the GENERIC ATN approach.

These results are in line with our expectations. The Concurrent Cursor concept introduces a small performance overhead, which can be observed for $n <= 12$. Eligible cursors have to be identified before they can process the input and children of cursors have to be created, while the GENERIC ATN can directly process all inputs. However, as n increases this overhead is negligible in comparison to the number of arcs tested and transitions made by the GENERIC ATN. Before any input has been processed there are n^2 active states for the GENERIC ATN approach, since there are constantly n^2 networks. In the CATN there is only one active cursor. After D_n has been processed, there are still n^2 active states for the GENERIC ATN approach. In the CATN there are $\sum_{i=0}^{2} n^i$ cursors, since one cursor remains at the start state, n cursors represent A_i with $i \in [1, 2, ..., n]$ at t_a, and at t_b all combinations of $A_i \times B_i$ with $i \in [1, 2, ..., n]$ are considered. In total, both fusion approaches exhibit a quadratic growth of active states ($\mathcal{O}(n^2)$). However, in terms of processing time the GENERIC ATN tests more arcs and makes more transitions per input. Each input triggers a test for all n^2 active states right from the beginning. The CATN only processes input for eligible cursors. The number of such cursors starts with one and increases depending on the randomization of inputs.

In conclusion, the presented benchmark results support the fundamental practicability of our contribution. To make measurements comparable the GENERIC ATN approach sorts chronologically unsorted input before the actual processing, artificially constructs a set of single ATNs, and copies register contents to the end state. The benchmark showcases that the GENERIC ATN approach can not handle chronologically unsorted dynamically, since it has to wait until all guesses are provided. Further, the processing of probabilistic input is merely possible due to the artificial construction of networks, which impacts the clarity and conciseness of network definitions in realistic use cases.

4. Discussion

The Concurrent Cursor concept enables to process probabilistic (R_{prob}) and chronologically unsorted input (R_{uns}) typically provided to semantic-level fusion methods. At the same time it supports the provision of continuous and revertible feedback to inform the user about the current state of the semantic fusion and to provide insight in why a certain command has not been successfully recognized

(R_{fb}). Unification as used in *Quickset* [30] also meets R_{prob} and R_{uns}. However, the system has to wait for the user to finish the interaction before analyzing it. This restriction renders the fusion method unable to provide continuous feedback during interaction. Our reference implementation, the CATN, builds upon four design decisions to jointly fulfill all identified fundamental requirements: An ATN, the Scala programming language, network abstraction constructs, and semantics-based state- and behavior-management techniques.

The CATN is set out to facilitate rapid prototyping of MMIs. Five characteristics concretely contribute to this trait. (1) The declarative nature of the CATN's procedural approach avoids the time costly training required by machine-learning based approaches. (2) The recursiveness of the underlying transition network permits the reuse of substructures, like the dedicated subgraph for recognizing noun phrases shown in the use case implementation. (3) The network abstraction constructs of our description language complement this reuse by abstracting repeatedly required network configurations. Finally, the concrete state and behavior elements of the application are decoupled from (4) the semantic integration process, by means of semantic queries, and from (5) lexical information, i.e., from the tokens of the interface's multimodal grammar, by means of the lexicon. Altogether, the options of reuse save time and thus foster rapid development cycles. Moreover, the achieved level of decoupling puts an „universal multimodal interface", comprised of basic commands that can be readily added to any application, close at hand. Ultimately, it shows how a procedural fusion method can satisfy all fundamental requirements for performing semantic fusion, while maintaining the support of rapid MMI development processes and thus answers the research question.

This facilitation of development and the completeness of captured requirements has been validated by a series of proof of concept demonstrations. Their implementation guided the concept development process and provided a basis for assessment and improvements. In the following, the most relevant proof of concept demonstrations for our CATN are briefly presented (see Figure 14). They comprise various student projects, theses, as well as results of a master-level *Multimodal Interface* course. It shows that non-experts can effectively implement MMIs even for non-trivial application areas like virtual and mixed reality. This is in line with the general benefits of procedural fusion methods identified in literature, i.e., simple comprehensibility and compatibility with typical interface development processes. Figure 14 showcases a few of these applications and demonstrations. More details can be found on our project page [51]. *SiXton's Curse* [9] is a semi immersive, multimodal adventure game in which a user has to defend a town from AI controlled virtual ghosts. For this purpose, the user has several spells at his disposal that can be cast by the combined use of speech and gestures. The demonstration by [8] is a multimodal, mixed reality, real-time strategy game on a digital surface. Players can command their forces by speaking, touching, and gesturing. More recently, a cherry picking approach [68] has been applied to allow the exploitation of the simulation and rendering capacities of game engines. *Space Tentacle* [7] and *Robot Museum* [69] have been developed with the *Unity 3D* game engine. *Space Tentacle* is a VR adventure game in which the user has to multimodally communicate with an artificial intelligence on a space ship to solve puzzles. *Robot Museum* is a student project which explores multimodal communication with a virtual companion in a museums context. *Big Bang* [70] is a VR universe builder implemented with the *Unreal 4* game engine.

The general feasibility of the Concurrent Cursor concept and the CATN has further been validated through a comparative performance benchmark. The presented benchmark results support the fundamental practicability of our contribution. They show that the CATN is comparable in performance to state of the art ATN-based approaches. Apart from the raw performance comparison, the benchmark showcases that the comparison partner can not handle chronologically unsorted input dynamically. Further, the processing of probabilistic input is merely possible due to the artificial construction of an exponentially growing set of single ATN configurations (as a function of the number of n-best guesses). While this high number of networks makes the generic ATN approach impractical with respect to the definition of valid multimodal utterance, the concise description language of the

CATN has been perceived to be usable during the development of the presented proof of concept demonstrations. For high numbers of n-best guesses the CATN outperforms its comparison partner. Thus, it may also prove effective in earlier stages of the input processing pipeline as well as in (future) use cases that consider a considerably higher number of modalities for fusion than just speech and gesture.

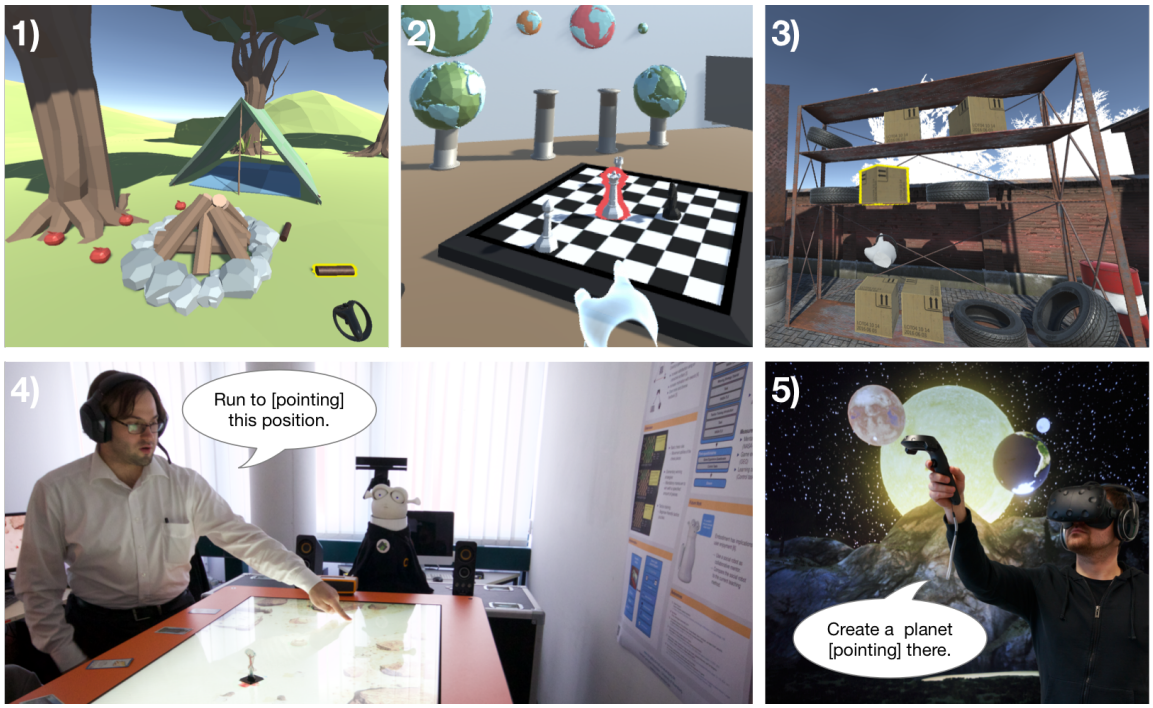

Figure 14. Illustration of student projects using the CATN. (1)–(3) are results of a master-level *Multimodal Interface* course. Students implemented virtual environments with *Unity 3D* and designed instruction-based multimodal interfaces. Example interactions comprise commands like: "*Put [pointing] that firewood [pointing] there.*" (1), "*Select [pointing] that white chess piece.*" (2), "*Delete [pointing] that box.*" (3). (4) showcases a comparison of a multimodal- with a menu-based interface for interactive surfaces. (5) illustrates the *Big Bang* demonstration that is implemented with the *Unreal 4* engine.

The main limitations of our contribution are twofold and indicate canonical future research. Firstly, an elaborated formal performance analysis that especially considers typical real-time constraints of RIS has not been conducted. Yet, the fundamental benchmark and the development of the presented proof of concept demonstrations indicate a principal suitability and revealed further insight. Conditions should check inputs in terms of their temporal, syntactical, and semantical validity early on, to avoid the creation of cursor strands for false guesses. In addition, it has proven to be advantages to model high frequency data sources as context, i.e., in the application state. They can thus be polled on demand instead of being pushed to the fusion method in large numbers, where they would substantially increase the number of active cursors. For instance, in case of the mixed reality board-game [8], the detected positions of the users' hands touching the interactive surface are represented as properties of a semantic entity *TouchInput*. In response to an uttered demonstrative key word, i.e., "*that*" or "*there*", a dedicated arc condition checks if touches occurred during the speech input by means of semantic queries and the application state history. Both of these best practices, representing high frequency data sources as context and defining restrictive arc conditions, led to proof of concept demonstrations with no noticeable performance impact regarding the flow of interaction. Secondly, the CATN's description language and its network abstraction constructs are designed to facilitate development, however its effectiveness in this regard has solely been controlled by the application of an API peer review method [20]. The usability of a system's programming interface for developers, i.e.,

its API usability [71] includes its learnability, the efficiency and correctness with which a developer can use it, its quality to prevent errors, its consistency, and its matching to the developers' mental models. API usability is paramount for facilitating a rapid prototyping process (of MMIs). However, properly assessing or even comparing API usability is a delicate endeavor with a limited number of methods and no obvious choices [72,73]. Subjective methods are the primary choice for API usability assessment and comprise expert reviews based on guidelines and user studies, such as think-aloud usability evaluation, API peer reviews [71,74], or questionnaires based on programming tasks [72,73].

5. Conclusions

This article targets semantic fusion methods for natural, synergistic, and intentional multimodal interactions that are compliant with rapid development cycles. On the basis of a comprehensive review of associated contributions and with the help of an interaction use case, we identify seven fundamental requirements: Action derivation, continuous feedback, context-sensitivity, temporal relation support, access to the interaction context, as well as the support of chronologically unsorted and probabilistic input. The Concurrent Cursors concept provides a solution for fulfilling the latter two requirements and is proposed as the main contribution of this paper. The feasibility of the Concurrent Cursors concept is validated by our reference implementation the CATN in a series of proof of concept demonstrations as well as through a comparative performance benchmark. The CATN bases on four design decisions: An ATN, the Scala programming language, network abstraction constructs, and semantics-based state- and behavior-management techniques. Altogether, the CATN thus fulfills all identified requirements and—to our best knowledge—fills a lack amongst previous solutions. The CATN supports rapid prototyping of MMIs by means of five concrete traits: its declarative nature, the recursiveness of the underlying transition network, the network abstraction constructs of its description language, the utilized semantic queries, and the utilized lexicon. Our reference implementation is used in various student projects as well as master-level courses and shows that non-experts can effectively implement MMIs for non-trivial application areas like virtual and mixed reality. The CATN is available for researchers [51].

In our ongoing research we plan to further formally analyze the performance of the proposed concept under realistic conditions with respect to typical real-time constraints of interactive systems. Moreover, we intend to evaluate the API usability of the CATN's description language and its network abstraction constructs. Finally, we aim to research the practical limitations of a universal multimodal interface that is theoretically supported by the abstraction layers utilized in the CATN.

Author Contributions: All authors contributed equally to this work.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MMI	Multimodal Interaction
MMS	Multimodal System
RIS	Real-time Interactive System
ATN	Augmented Transition Network
tATN	temporal Augmented Transition Network
cATN	concurrent Augmented Transition Network
FSA	Finite-State Automaton
FST	Finite-State Transducer
SDK	Software Development Kit
SGIM	Speech and Gesture Interfaces for Multimedia
API	Application Programming Interface

References

1. Oviatt, S.; Cohen, P. Perceptual User Interfaces: Multimodal Interfaces That Process What Comes Naturally. *Commun. ACM* **2000**, *43*, 45–53. [CrossRef]
2. Nigay, L.; Coutaz, J. A design space for multimodal systems: Concurrent processing and data fusion. In Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems, Amsterdam, The Netherlands, 24–29 April 1993; pp. 172–178.
3. Kaiser, E.; Olwal, A.; McGee, D.; Benko, H.; Corradini, A.; Li, X.; Cohen, P.; Feiner, S. Mutual Disambiguation of 3D Multimodal Interaction in Augmented and Virtual Reality. In Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI '03), Vancouver, BC, Canada, 5–7 November 2003; ACM: New York, NY, USA, 2003; pp. 12–19. [CrossRef]
4. Sharma, R.; Pavlovic, V.I.; Huang, T.S. Toward multimodal human-computer interface. *Proc. IEEE* **1998**, *86*, 853–869. [CrossRef]
5. Oviatt, S.; Coulston, R.; Lunsford, R. When do we interact multimodally?: Cognitive load and multimodal communication patterns. In Proceedings of the 6th International Conference on Multimodal Interfaces, State College, PA, USA, 13–15 October 2004; pp. 129–136.
6. Oviatt, S. Multimodal interfaces. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, 3rd ed.; Lawrence Erlbaum Associates Inc.: Mahwah, NJ, USA, 2012; pp. 405–430.
7. Zimmerer, C.; Fischbach, M.; Latoschik, M.E. Space Tentacles—Integrating Multimodal Input into a VR Adventure Game. In Proceedings of the 25th IEEE Virtual Reality (VR) Conference, Tuebingen/Reutlingen, Germany, 18–22 March 2018; pp. 745–746.
8. Link, S.; Barkschat, B.; Zimmerer, C.; Fischbach, M.; Wiebusch, D.; Lugrin, J.L.; Latoschik, M.E. An Intelligent Multimodal Mixed Reality Real-Time Strategy Game. In Proceedings of the 23rd IEEE Virtual Reality (IEEE VR) Conference, Greenville, SC, USA, 19–23 March 2016.
9. Fischbach, M.; Wiebusch, D.; Giebler-Schubert, A.; Latoschik, M.E.; Rehfeld, S.; Tramberend, H. SiXton's curse—Simulator X demonstration. In Proceedings of the Virtual Reality Conference (VR), Singapore, 19–23 March 2011; pp. 255–256.
10. Fischbach, M.W. Enhancing Software Quality of Multimodal Interactive Systems. Ph.D. Thesis, Universität Würzburg, Würzburg, Germany, 2017.
11. Peters, S.; Johanssen, J.O.; Bruegge, B. An IDE for Multimodal Controls in Smart Buildings. In Proceedings of the 18th ACM International Conference on Multimodal Interaction (ICMI), Tokyo, Japan, 12–16 November 2016; ACM: New York, NY, USA, 2016; pp. 61–65. [CrossRef]
12. Cacace, J.; Finzi, A.; Lippiello, V. A robust multimodal fusion framework for command interpretation in human-robot cooperation. In Proceedings of the 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Lisbon, Portugal, 28 August–1 September 2017; pp. 372–377. [CrossRef]
13. Pfleger, N. Context Based Multimodal Fusion. In Proceedings of the 6th International Conference on Multimodal Interfaces (ICMI '04), State College, PA, USA, 13–15 October 2004; ACM: New York, NY, USA, 2004; pp. 265–272. [CrossRef]
14. Lalanne, D.; Nigay, L.; Palanque, P.; Robinson, P.; Vanderdonckt, J.; Ladry, J.F. Fusion Engines for Multimodal Input: A Survey. In Proceedings of the 2009 International Conference on Multimodal Interfaces, Cambridge, MA, USA, 2–4 November 2009; pp. 153–160.
15. Neal, J.G.; Thielman, C.Y.; Dobes, Z.; Haller, S.M.; Shapiro, S.C. Natural Language with Integrated Deictic and Graphic Gestures. In Proceedings of the Workshop on Speech and Natural Language (HLT '89), Cape Cod, MA, USA, 15–18 October 1989; Association for Computational Linguistics: Stroudsburg, PA, USA, 1989; pp. 410–423. [CrossRef]
16. Latoschik, M.E. Designing Transition Networks for Multimodal VR-Interactions Using a Markup Language. In Proceedings of the 4th IEEE International Conference on Multimodal Interfaces (ICMI '02), Pittsburgh, PA, USA, 14–16 October 2002; IEEE Computer Society: Washington, DC, USA, 2002; p. 411. [CrossRef]

17. Nigay, L.; Coutaz, J. A generic platform for addressing the multimodal challenge. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Denver, CO, USA, 7–11 May 1995; ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 1995; pp. 98–105.
18. Duarte, C.; Carriço, L. A Conceptual Framework for Developing Adaptive Multimodal Applications. In Proceedings of the 11th International Conference on Intelligent User Interfaces (IUI '06), Sydney, Australia, 29 January–1 February 2006; ACM: New York, NY, USA, 2006; pp. 132–139. [CrossRef]
19. Holzapfel, H.; Nickel, K.; Stiefelhagen, R. Implementation and Evaluation of a Constraint-based Multimodal Fusion System for Speech and 3D Pointing Gestures. In Proceedings of the 6th International Conference on Multimodal Interfaces (ICMI '04), State College, PA, USA, 13–15 October 2004; ACM: New York, NY, USA, 2004; pp. 175–182. [CrossRef]
20. Fischbach, M.; Wiebusch, D.; Latoschik, M.E. Semantic Entity-Component State Management Techniques to Enhance Software Quality for Multimodal VR-Systems. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 1342–1351. [CrossRef] [PubMed]
21. Bolt, R.A. 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), Seattle, WA, USA, 14–18 July 1980; ACM: New York, NY, USA, 1980; pp. 262–270. [CrossRef]
22. Latoschik, M.E. A User Interface Framework for Multimodal VR Interactions. In Proceedings of the 7th International Conference on Multimodal Interfaces, Torento, Italy, 4–6 October 2005; pp. 76–83.
23. Zhang, B.; Essl, G.; Mower Provost, E. Automatic Recognition of Self-reported and Perceived Emotion: Does Joint Modeling Help? In Proceedings of the 18th ACM International Conference on Multimodal Interaction (ICMI 2016), Tokyo, Japan, 12–16 November 2016; ACM: New York, NY, USA, 2016; pp. 217–224. [CrossRef]
24. Kalimeri, K.; Saitis, C. Exploring Multimodal Biosignal Features for Stress Detection During Indoor Mobility. In Proceedings of the 18th ACM International Conference on Multimodal Interaction (ICMI 2016), Tokyo, Japan, 12–16 November 2016; ACM: New York, NY, USA, 2016; pp. 53–60. [CrossRef]
25. Dibeklioğlu, H.; Hammal, Z.; Yang, Y.; Cohn, J.F. Multimodal Detection of Depression in Clinical Interviews. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15), Seattle, WA, USA, 9–13 November 2015; ACM: New York, NY, USA, 2015; pp. 307–310. [CrossRef]
26. Pérez-Rosas, V.; Abouelenien, M.; Mihalcea, R.; Burzo, M. Deception Detection Using Real-life Trial Data. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15), Seattle, WA, USA, 9–13 November 2015; ACM: New York, NY, USA, 2015; pp. 59–66. [CrossRef]
27. Koons, D.B.; Sparrell, C.J. Iconic: Speech and Depictive Gestures at the Human-machine Interface. In Proceedings of the Conference Companion on Human Factors in Computing Systems, Boston, MA, USA, 24–28 April 1994; pp. 453–454.
28. Dumas, B.; Lalanne, D.; Ingold, R. HephaisTK: A Toolkit for Rapid Prototyping of Multimodal Interfaces. In Proceedings of the 2009 International Conference on Multimodal Interfaces (ICMI-MLMI '09), Cambridge, Massachusetts, USA, 2–4 November 2009; ACM: New York, NY, USA, 2009; pp. 231–232. [CrossRef]
29. Cohen, P.R.; Johnston, M.; McGee, D.; Oviatt, S.; Pittman, J.; Smith, I.; Chen, L.; Clow, J. QuickSet: Multimodal Interaction for Distributed Applications. In Proceedings of the Fifth International Conference on Multimedia, Seattle, WA, USA, 9–13 November 1997; pp. 31–40.
30. Johnston, M.; Cohen, P.R.; McGee, D.; Oviatt, S.L.; Pittman, J.A.; Smith, I. Unification-based Multimodal Integration. In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics (ACL '98), Madrid, Spain, 7–12 July 1997; Association for Computational Linguistics: Stroudsburg, PA, USA, 1997; pp. 281–288. [CrossRef]
31. Wu, L.; Oviatt, S.L.; Cohen, P.R. From members to teams to committee-a robust approach to gestural and multimodal recognition. *IEEE Trans. Neural Netw.* **2002**, *13*, 972–982. [CrossRef] [PubMed]
32. Chai, J.Y.; Hong, P.; Zhou, M.X. A Probabilistic Approach to Reference Resolution in Multimodal User Interfaces. In Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04), Funchal, Madeira, Portugal, 13–16 January 2004; ACM: New York, NY, USA, 2004; pp. 70–77. [CrossRef]

33. Dumas, B.; Signer, B.; Lalanne, D. Fusion in Multimodal Interactive Systems: An HMM-based Algorithm for User-induced Adaptation. In Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '12), Copenhagen, Denmark, 25–26 June 2012; ACM: New York, NY, USA, 2012; pp. 15–24. [CrossRef]
34. Johnston, M.; Bangalore, S. Finite-state multimodal parsing and understanding. In Proceedings of the 18th Conference on Computational Linguistics, Saarbrücken, Germany, 31 July–4 August 2000; Association for Computational Linguistics: Stroudsburg, PA, USA, 2000; Volume 1, pp. 369–375.
35. Johnston, M.; Bangalore, S. Finite-state Multimodal Integration and Understanding. *Nat. Lang. Eng.* **2005**, *11*, 159–187. [CrossRef]
36. Adams, W.H.; Iyengar, G.; Lin, C.Y.; Naphade, M.R.; Neti, C.; Nock, H.J.; Smith, J.R. Semantic Indexing of Multimedia Content Using Visual, Audio, and Text Cues. *EURASIP J. Adv. Signal Process.* **2003**, *2003*, 987184. [CrossRef]
37. Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; Ng, A.Y. Multimodal deep learning. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, WA, USA, 28 June–2 July 2011; pp. 689–696.
38. Martínez, H.P.; Yannakakis, G.N. Deep multimodal fusion: Combining discrete events and continuous signals. In Proceedings of the 16th International Conference on Multimodal Interaction, Istanbul, Turkey, 12–16 November 2014; pp. 34–41.
39. Atrey, P.K.; Hossain, M.A.; El Saddik, A.; Kankanhalli, M.S. Multimodal fusion for multimedia analysis: A survey. *Multimed. Syst.* **2010**, *16*, 345–379. [CrossRef]
40. Hoste, L.; Dumas, B.; Signer, B. Mudra: A Unified Multimodal Interaction Framework. In Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI '11), Alicante, Spain, 14–18 November 2011; ACM: New York, NY, USA, 2011; pp. 97–104. [CrossRef]
41. Dumas, B.; Lalanne, D.; Oviatt, S. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In *Human Machine Interaction*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 3–26. [CrossRef]
42. Potamianos, G.; Marcheret, E.; Mroueh, Y.; Goel, V.; Koumbaroulis, A.; Vartholomaios, A.; Thermos, S. Audio and Visual Modality Combination in Speech Processing Applications. In *The Handbook of Multimodal-Multisensor Interfaces*; Association for Computing Machinery and Morgan & Claypool: New York, NY, USA, 2017; pp. 489–543. [CrossRef]
43. Erhan, D.; Bengio, Y.; Courville, A.; Manzagol, P.A.; Vincent, P.; Bengio, S. Why Does Unsupervised Pre-training Help Deep Learning? *J. Mach. Learn. Res.* **2010**, *11*, 625–660.
44. Oviatt, S.; Cohen, P.R. *The Paradigm Shift to Multimodality in Contemporary Computer Interfaces*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2015.
45. Putze, F.; Popp, J.; Hild, J.; Beyerer, J.; Schultz, T. Intervention-free Selection Using EEG and Eye Tracking. In Proceedings of the 18th ACM International Conference on Multimodal Interaction (ICMI 2016), Tokyo, Japan, 12–16 November 2016; ACM: New York, NY, USA, 2016; pp. 153–160. [CrossRef]
46. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. In Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12), Lake Tahoe, Nevada, 3–6 December 2012; pp. 2951–2959.
47. Blum, A.L.; Langley, P. Selection of relevant features and examples in machine learning. *Artif. Intell.* **1997**, *97*, 245–271. [CrossRef]
48. Oviatt, S.; Schuller, B.; Cohen, P.R.; Sonntag, D.; Potamianos, G.; Krüger, A. (Eds.) *The Handbook of Multimodal-Multisensor Interfaces: Foundations, User Modeling, and Common Modality Combinations—Volume 1*; Association for Computing Machinery and Morgan & Claypool: New York, NY, USA, 2017.
49. Mayhew, D.J. The Usability Engineering Lifecycle. In Proceedings of the CHI '99 Extended Abstracts on Human Factors in Computing Systems (CHI EA '99), Pittsburgh, PA, USA, 15–20 May 1999; ACM: New York, NY, USA, 1999; pp. 147–148. [CrossRef]
50. Woods, W.A. Transition Network Grammars for Natural Language Analysis. *Commun. ACM* **1970**, *13*, 591–606. [CrossRef]
51. Zimmerer, C.; Fischbach, M.; Latoschik, M.E. Concurrent Augmented Transition Network—Project Page. 2018. Available online: https://www.hci.uni-wuerzburg.de/projects/mmi/ (accessed on 22 August 2018).

52. Poddar, I.; Sethi, Y.; Ozyildiz, E.; Sharma, R. Toward natural gesture/speech HCI: A case study of weather narration. In Proceedings of the Workshop on Perceptual User Interfaces (PUI98), San Francisco, CA, USA, 5–6 November 1998; pp. 1–6.
53. Krahnstoever, N.; Kettebekov, S.; Yeasin, M.; Sharma, R. A Real-Time Framework for Natural Multimodal Interaction with Large Screen Displays. In Proceedings of the 4th IEEE International Conference on Multimodal Interfaces (ICMI '02), Pittsburgh, PA, USA, 14–16 October 2002; IEEE Computer Society: Washington, DC, USA, 2002; p. 349. [CrossRef]
54. Serrano, M.; Nigay, L.; Lawson, J.Y.L.; Ramsay, A.; Murray-Smith, R.; Denef, S. The Openinterface Framework: A Tool for Multimodal Interaction. In Proceedings of the Extended Abstracts on Human Factors in Computing Systems, Florence, Italy, 5–10 April 2008; pp. 3501–3506.
55. Wagner, J.; Lingenfelser, F.; Baur, T.; Damian, I.; Kistler, F.; André, E. The Social Signal Interpretation (SSI) Framework: Multimodal Signal Processing and Recognition in Real-time. In Proceedings of the 21st ACM International Conference on Multimedia (MM '13), Barcelona, Spain, 21–25 October 2013; ACM: New York, NY, USA, 2013; pp. 831–834. [CrossRef]
56. Latoschik, M.E. A general framework for multimodal interaction in virtual reality systems: PrOSA. In Proceedings of the Future of VR and AR Interfaces-Multimodal, Humanoid, Adaptive and Intelligent—Workshop at IEEE Virtual Reality, Yokohama, Japan, 14 March, 2001; No. 138, pp. 21–25.
57. Bouchet, J.; Nigay, L.; Ganille, T. ICARE Software Components for Rapidly Developing Multimodal Interfaces. In Proceedings of the 6th International Conference on Multimodal Interfaces, State College, PA, USA, 13–15 October 2004; pp. 251–258.
58. Latoschik, M.E.; Tramberend, H. Short Paper: Engineering Realtime Interactive Systems: Coupling & Cohesion of Architecture Mechanisms. In Proceedings of the 16th Eurographics Conference on Virtual Environments & Second Joint Virtual Reality (EGVE—JVRC'10), Stuttgart, Germany, 27 September–1 October 2010; Eurographics Association: Aire-la-Ville, Switzerland, 2010; pp. 25–28. [CrossRef]
59. Latoschik, M.E.; Fischbach, M. Engineering variance: Software techniques for scalable, customizable, and reusable multimodal processing. In Proceedings of the International Conference on Human-Computer Interaction, Heraklion, Crete, Greece, 22–27 June 2014; Springer: Cham, Switzerland, 2014; pp. 308–319.
60. Fischbach, M. Software Techniques for Multimodal Input Processing in Realtime Interactive Systems. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15), Seattle, WA, USA, 9–13 November 2015; ACM: New York, NY, USA, 2015; pp. 623–627. [CrossRef]
61. Molich, R.; Nielsen, J. Improving a Human-computer Dialogue. *Commun. ACM* **1990**, *33*, 338–348. [CrossRef]
62. Roche, E. Transducer Parsing of Free and Frozen Sentences. *Nat. Lang. Eng.* **1996**, *2*, 345–350. [CrossRef]
63. Bourguet, M.L. A toolkit for creating and testing multimodal interface designs. *Companion Proc. UIST* **2002**, *2*, 29–30.
64. Hopcroft, J.E.; Ullman, J.D. *Introduction To Automata Theory, Languages, And Computation*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1990.
65. Allen, J.F. Maintaining Knowledge About Temporal Intervals. *Commun. ACM* **1983**, *26*, 832–843. [CrossRef]
66. Zimmerer, C.; Fischbach, M.; Latoschik, M.E. Maintainable Management and Access of Lexical Knowledge for Multimodal Virtual Reality Interfaces. In Proceeding of the 22nd ACM Symposium on Virtual Reality Software and Technology (VRST), Munich, Germany, 2–4 November 2016; pp. 347–348.
67. Marcus, M.P.; Marcinkiewicz, M.A.; Santorini, B. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguist.* **1993**, *19*, 313–330.
68. Wiebusch, D.; Zimmerer, C.; Latoschik, M.E. Cherry-Picking RIS Functionality—Integration of Game and VR Engine Sub-Systems based on Entities and Events. In Proceeding of the 10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), Los Angeles, CA, USA, 19 March 2017; IEEE Computer Society: Los Alamitos, CA, USA, 2017.
69. Heidrich, D.; Zimmerer, C.; Fischbach, M.; Latoschik, M.E. Robot Museum. 2018. Available online: https://www.hci.uni-wuerzburg.de/2018/06/12/robot-museum-demo/ (accessed on 22 August 2018).
70. Zimmerer, C.; Fischbach, M.; Latoschik, M.E. Big Bang. 2016. Available online: https://www.hci.uni-wuerzburg.de/2016/10/11/planetarium/ (accessed on 22 August 2018).
71. McLellan, S.G.; Roesler, A.W.; Tempest, J.T.; Spinuzzi, C.I. Building more usable APIs. *IEEE Softw.* **1998**, *15*, 78–86. [CrossRef]

72. Piccioni, M.; Furia, C.A.; Meyer, B. An Empirical Study of API Usability. In Proceeding of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, 10–11 October 2013; pp. 5–14. [CrossRef]
73. Myers, B.A.; Stylos, J. Improving API Usability. *Commun. ACM* **2016**, *59*, 62–69. [CrossRef]
74. Ruiz, N.; Chen, F.; Oviatt, S. Chapter 12—Multimodal Input. In *Multimodal Signal Processing*; Thiran, J.P., Marqués, F., Bourlard, H., Eds.; Academic Press: Oxford, UK, 2010; pp. 231–255. [CrossRef]