Semi-Automatically Generated High-Level Fusion for Multimodal User Interfaces

Dominik Ertl, Sevan Kavaldjian, Hermann Kaindl, Jürgen Falb
Vienna University of Technology, Institute of Computer Technology
A–1040 Vienna, Austria
{ertl, kavaldjian, kaindl, falb}@ict.tuwien.ac.at

Abstract

Reliable high-level fusion of several input modalities is hard to achieve, and (semi-)automatically generating it is even more difficult. However, it is important to address in order to broaden the scope of providing user interfaces semi-automatically.

Our approach starts from a high-level discourse model created by a human interaction designer. It is modality-independent, so an annotated discourse is semi-automatically generated, which influences the fusion mechanism. Our high-level fusion checks hypotheses from the various input modalities by use of finite state machines. These are modality-independent, and they are automatically generated from the given discourse model. Taking all this together, our approach provides semi-automatic generation of high-level fusion. It currently supports input modalities graphical user interface, (simple) speech, a few hand gestures, and a bar code reader.

1 Introduction

While semi-automatic generation of user interfaces is still primarily a matter of research, the approaches are becoming increasingly mature. Our own approach is based on high-level discourse modeling, and its focus has been on graphical user interfaces (GUIs), more precisely WIMP (window, icon, menu, pointer) interfaces [4, 9, 15]. More recently, we extended this approach to multimodal interfaces.

Managing the input from several modalities requires high-level fusion. This is a challenging issue on its own, but semi-automatic generation makes it even harder. Still, we can generate finite state machines (FSMs) — for checking input hypotheses from the various modalities — from our discourse models. The results of these checks are communicative acts that provide an abstract representation of what is believed to be the input.

Our overall generation process from high-level discourses to multimodal user interfaces is a further development of the one presented in [3], including the new generation of FSMs (see Figure 1). First, the modality-independent discourse is modeled by an interaction designer. Then, the FSMs are generated, and a task-level transformation leads to an annotated discourse. At last, the annotated discourse is rendered into the final modalities.

As a running example, we use a small part of a multimodal user interface of a research robot shopping cart, that is a non-trivial application of our approach. We use this example to show the semi-automatic generation of the fusion mechanism and its runtime behavior.

The remainder of this paper is organized in the following manner. First, we provide some background on high-level fusion mechanisms for multimodal user interfaces and our approach to interaction design, in order to make this paper self-contained. Then we explain the concept of a modality provider and sketch how the modalities that we currently use for input are fed into the fusion mechanism. After that, we present our annotated discourse model and its semi-automatic generation. Based on that, the core of the paper is dedicated to explain our new approach to semi-automatic generation of fusion based on finite state machines. Finally, we discuss the fulfillment of the CARE properties [2] by our approach.

2 State of the Art and Background

In this section, we sketch some background information about high-level fusion mechanisms for multimodal user interfaces, and our interaction design approach.

2.1 High-level Fusion for Multimodal User Interfaces

A multimodal user interface offers more than one modality (e.g., speech and GUI) to the user. In order to process
input, a multimodal user interface requires a fusion mechanism for modality integration. Previous work distinguishes low-level fusion from high-level fusion [6].

Low-level fusion is part of the modalities recognition mechanism. E.g., the speech recognition as well as the gesture recognition have their own fusion modules, often with Bayesian-Networks or Hidden-Markov-Models as their underlying model.

High-level fusion, also known as semantic fusion, is an approach for integrating different multimodal inputs into one higher-level input. The fundamental issue in high-level fusion is to choose appropriate symbols and their meaning based on mainly numerical input. For this purpose, the temporal relation and timing of multiple input signal is important as well. So, the question arises when a certain input occurred and whether it is related to other inputs within a defined time window. Related inputs can be fused in four different ways. Therefore, it is considered desirable that multimodal user interfaces support fulfilling most of the CARE properties [2]:

- **Complementarity** combines input of different modalities into one symbol, e.g., saying “guide me to” and clicking on “apples” on a touch screen can be combined into a symbol or action “guide me to apples”.

- **Assignment** associates symbols or actions to a particular modality. Thus, related input on other modalities will be ignored, e.g., if the symbol or action “guide me to apples” is only assigned to clicking an icon on a GUI, any spoken utterance or gesture regarding guiding will be ignored.

- **Redundancy** takes care that in case identical input is received via different modalities, a symbol or action is executed only once. Redundancy increases the robustness of a multimodal user interface.

- **Equivalence** provides the freedom for using a modality of choice for creating an input symbol or action, e.g., a user has the freedom to either say “guide me to apples” or just clicking on an apple icon on a GUI to achieve the desired action.

Johnston et al. [7] provide a practical solution for high-level fusion that uses a finite-state based multimodal grammar approach. This work takes speech and gesture streams as inputs and delivers their joint representation as output. This approach provides a general framework for multimodal ambiguity resolution. The advantage is the lightweight in computational needs. A finite state machine (FSM) parses input from multiple modes to combine their content into a single semantic representation. Such speech and gesture inputs are lattices and their combined meaning represents a symbol.

An improved version of this approach by Portillo et al. [16] employs a hybrid strategy, taking temporal constraints and additional information at the dialogue level into account. More recent work provides a conceptual framework for design and usability of multimodal interaction with finite state machines [1]. This framework helps the designer of multimodal interfaces to make better informed decisions about the allocation of interaction modalities to multimodal commands.

Fusion for a broad spectrum of modality combinations and research results for multimodal interaction with mobile devices are presented in [19] in a comprehensive way.

Other work addresses multimodal architectures with FSMs, e.g., providing a multimodal application architecture that combines finite-state multimodal language processing and a speech-act based multimodal dialogue manager to enable rapid prototyping of multimodal user interfaces [8]. However, this work does not take (semi-)automatic generation of multimodal user interfaces and of parts of the fusion mechanism into account.

Previous work on model-based transformations for multimodal user interfaces [18] considers the generation of user interfaces. Their multimodal user interface takes GUI and speech into account, lacking other modalities like our bar code reader. Additionally, their approach does not need a fusion mechanism, since an independent interpretation pro-
cess is performed for each modality. So, they only support the CARE properties assignment and equivalence, not complementarity and redundancy. The authoring of user interfaces for combined use of modalities is presented for the multimodal TERESA environment in [14]. It provides support when designing and developing interfaces accessible through various device types from task models based on the ConcurTaskTree notation.

2.2 Interaction Design

Our approach starts with a modality-independent interaction design on a high level, based on discourses in the sense of dialogues. This enables the designer to concentrate on the communication between the user and the system primarily. Our discourse models specify the intention of the communication and the information that will be uttered. We use different natural language theories — speech act theory [17], conversation analysis [11] and rhetorical structure theory [12] — to provide the necessary modeling means.

Let us explain our approach in more detail using the example discourse shown in Figure 2. The major modeling constructs shown are communicative acts represented by rounded boxes. Communicative acts capture the intention of the communication, like asking a question, informing about new facts or requesting an action upon receiving the communicative act. A communicative act also conveys the content of the utterance, that refers to objects in the domain of the discourse and to actions of the given application. Related models have to be provided by the designer, too. Finally, a communicative act is associated with one of the two parties taking part in the discourse. This is indicated by the fill color of a box in Figure 2, which corresponds to the color of one of the communication party icons.

The communicative acts in our running example can be interpreted in the following way: the user, who is guided by the robot through a supermarket, has several options to choose from:

- requesting the robot to follow her,
- or requesting the robot to guide her to a particular product,
- or requesting the robot to meet her at a certain product,
- or informing the robot that she has put a certain product into the shopping cart.

Then, the requests can be accepted by the robot.

Communicative acts are combined into adjacency pairs for modeling typical turn-taking sequences like question – answer or request – accept. Adjacency pairs are represented by diamonds that connect the communicative acts that belong together. Adjacency pairs are the basic interaction sequences in our discourse models.

Rhetorical relations associate these adjacency pairs according to subject-matter and temporal relationships. In our running example, an Alternative relation relates the three Requests and an Informing with one another as alternatives. This example contains a temporal relationship between the adjacency pairs. Each Request and the Informing cannot be uttered in parallel, e.g., on a graphical user interface, only one button can be pressed at the same time. Rhetorical relations can also be used to build larger tree structures by combining rhetorical relations recursively.
A dialogue that conforms to a discourse model ends either when all communicative acts have been uttered or, implicitly, when communicative acts are continued in a super-ordinate discourse, or by external events.

3 Modalities

For semi-automatically generating a modality fusion component that supports a dialog according to a discourse model, a set of modalities has to be integrated into our communication platform. This communication platform is responsible for executing dialogs and modality-specific rendering of communicative acts. The integration effort varies from modality to modality. Currently we support four modalities: GUI, speech, hand gestures and bar code reader. For example, a GUI requires more integration effort than a bar code reader that only outputs a stream of numbers.

For administration and integration of several modalities, we introduce the role of a modality provider first. Then we sketch the four currently supported input modalities (GUI, speech, hand gestures and bar code reader) and how they are related to the modality provider and the fusion mechanism. Finally, we describe the configuration files of the modalities.

3.1 Modality Provider

A modality provider provides the necessary software component to register the modality in the communication platform and become part of the multimodal user interface. In this way, the modality provider administrates the various modalities for input (and output, while we restrict ourselves to input modalities in the context of this paper). As a benefit the modality provider just connects the modalities via a defined programming interface with the fusion mechanism.

So, what has to be done to make a modality available to the communication platform and thus to the user interface? In addition to the software component, a modality provider has to provide a binding of input modalities with communicative acts and associated actions in their content. This binding describes what kind of communication is, in principle, supported by a particular modality and is stored in form of modality component configuration files. Thus, this information is needed for the fusion at runtime to filter out input from the modalities that are not allowed in a particular stage of a discourse.

A given modality also requires a working implementation, of course, to support the communicative acts associated to it. The integration and implementation into the user interface differs from modality to modality. Modalities may have specific physical properties which have to be taken into account when using different devices. Such physical properties are, for example, the serialization behavior of speech opposed to the potential parallelism of a GUI, where several screen elements may be shown simultaneously.

3.2 GUI

In our running example, the GUI modality shall support Request communicative acts with actions followMe, guideMeTo and meetMeAt.

The implementation of the GUI modality is, more precisely, a WIMP (Window, Icons, Menu, Pointer) interface, where these Request communicative acts shall be rendered as buttons on a screen. It does not really matter in principle, whether this implementation is hand-crafted or semi-automatically generated. In fact, we have already developed a (model-driven) approach for semi-automatic generation of WIMP interfaces [5]. It currently leads to a final implementation of the user interface in Java Swing. On our running example robot cart, a touch screen is available for single finger touch input which customizes the GUI generation process to support a finger-based GUI modality instead of a pointer device-based GUI modality.

3.3 Speech

In our running example, speech input shall be bound to the communicative act type Request with the actions followMe, guideMeTo and meetMeAt in the content.

For the implementation of speech input, we currently use Julius\(^1\), a tool for continuous speech recognition. It is coupled with a TCP-server/client architecture for transferring messages from and to our communication platform. In case of speech, it is necessary to customize the language model to the given tasks and to provide a grammar, so that the tool can recognize a specific set of sentences or words corresponding to the given commands.

An initial version of the grammar can be generated from analyzing the propositional content of communicative acts specified in the discourse model. The grammar can then be adapted by the speech modality provider. A voca and a dict file has to be provided manually, which specify the pronunciation. All these files are read in at runtime and allow the Julius toolkit to recognize commands according to their binding to speech input. Related hypotheses are created as input to the modality fusion.

3.4 Hand Gestures

In our running example, the hand gestures modality supports the Request communicative act with the action followMe. So, one potential gestures is chosen for interpretation as the command followMe.

\(^1\)http://julius.sourceforge.jp/en\_index.php
To implement recognition of hand gestures on the robot cart, we use two-handed gesture recognition and head location in 3D. For alternative use on a PC only, we have also integrated the HandVU framework\(^2\) with our communication platform. It allows for one-handed gesture recognition. Either implementation provides a stream of potentially recognized gestures (via the TCP/IP channel) to the communication platform. They are interpreted there as distinct symbols, and related hypotheses are created as input to the fusion mechanism.

In contrast to GUI and speech, the hand gesture modality provides a symbolic mapping of gestures to action or content of communicative acts. We do not support interpreting a sign language (that would be comparable to speech input). In our current approach, one gesture has one unique meaning (one symbol). So, the number of recognized gestures (12 for the robot gesture recognition system, 6 for HandVU) limits the number of mappings to communicative acts. In theory, it is possible to combine several gestures to allow more symbols. This would lead to a simplified version of a sign language, but is outside the scope of this work.

### 3.5 Bar Code Reader

Our multimodal user interface allows the input modality bar code reader as well. In our running example, this input modality supports the Informing communicative act with the action putIntoCart.

The implementation is achieved by an external hardware device, so the needed drivers for the operating system had to be installed. In addition, a software module for the bar code reader was needed that implements the putIntoCart action to read in bar code strings.

When the bar code reader is used by a human to scan a product, a method call is triggered that sends the bar code string as an input hypothesis to the fusion mechanism.

### 3.6 Configuration Files of Modalities

Each modality requires a configuration file, that stores which types of communicative acts together with associated actions can be rendered in principle by the various modalities (e.g., Request, Informing, etc.). This mapping has to be specified by a human operator at compile time. Table 1 shows an example of such mappings for our running example. The headline contains the types of communicative acts, the left column the actions in the content. The column on the very right shows which CARE properties are fulfilled by such a mapping. Having several modalities listed in a single cell of the table means that they may be used jointly. A communicative act of type Accept — missing in this table — leads to modality-specific output elements, like labels on a touch screen or canned text for speech output.

<table>
<thead>
<tr>
<th></th>
<th>Request</th>
<th>Informing</th>
<th>CARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>followMe</td>
<td>GUI, speech, hand gesture</td>
<td></td>
<td>A, R, E</td>
</tr>
<tr>
<td>guideMeTo</td>
<td>GUI, speech</td>
<td></td>
<td>C, R, E</td>
</tr>
<tr>
<td>meetMeAt</td>
<td>GUI, speech</td>
<td></td>
<td>C, R, E</td>
</tr>
<tr>
<td>putIntoCart</td>
<td>Bar code reader</td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>

Table 1. Binding of Input Modalities to Communicative Acts and Related Actions.

---

\(^2\)http://www.movesinstitute.org/~kolisch/HandVu/HandVu.html

### 4 Annotated Discourse Model

We introduce an annotated discourse model for coupling the modeled communication with modalities, currently GUI, speech, hand gesture and bar code reader. An annotated discourse model is an extended modality-dependent discourse model that includes modality-specific information and properties of hardware devices that are relevant for the rendering of the used modalities. It specifies which communicative acts (together with their propositional content) are rendered by which modality. The annotated discourse model can be viewed as a compilation of several modality-dependent discourse models for the different modalities.

The annotated discourse model is semi-automatically generated from the modality-independent discourse and the specified modality bindings. The generation process of the annotated discourse model reads out at compile time the discourse model and the configuration files for all registered modalities. Then it maps all the communicative acts (including their content) and the modalities into one declarative model that is an abstract representation of the multimodal user interface.

Additionally, the generation process reads out device properties of the configuration files of the modalities. These device properties influence the final representation of a modality. For example, the GUI has different minimum sizes for buttons and scroll bars for GUIs on touch screens as compared to GUIs that are used with a mouse. In previous work [10], we take such application-specific specifications of GUI pointing granularity into account for semi-automatic GUI generation. Such modality-specific information is stored in the annotated discourse, too, and taken into account by the final rendering processes of the modalities.

In our semi-automatic generation process a human operator can modify the annotated discourse to orchestrate input recognition (and also output generation). Orchestration is the intentional marking or unmarking of modalities for
communicative acts in the modality-specific views. This potentially effects the interplay of the modalities.

The final rendering of the modalities requires that the render mechanism of a distinct modality takes into account only the allowed communicative acts of the modality. A specific view of the annotated discourse model for a modality is a subset of the annotated discourse model that consists only of those communicative acts with their according actions that fit the modality. Each modality may render only its own view. For example, the bar code reader modality is only able to render the Informing with putIntoCart. The final representation of the modalities is the concrete representation of the multimodal user interface.

5 FSM-Based Fusion

Our high-level fusion is based on finite state machines, that receive input from several modalities and output a communicative act to the dialogue manager of the communication platform. We briefly sketch one such cycle, as illustrated in Figure 3 from modality input to fusion output.

We assume a human user says “follow me” to the robot. Then, the speech input component sends a list of input hypotheses to the fusion component. An input hypotheses consists of an input expression and a confidence score. In our example, the speech input modality creates two input hypotheses for “follow me”. The first hypothesis is [expression:follow me, score:–100], the second hypothesis [expression:meet me at, score:–500]. The confidence score is an integer value that is calculated by the logic of the modality component and signifies the likelihood that this expression is the one that has actually been communicated. To compare the scores of different modalities, we normalize and linearize the score values.

Next, the fusion mechanism creates an instance of an FSM and inputs the expression of the first hypothesis into this FSM. An FSM corresponds to a regular grammar and analyzes a given input expression with respect to the regular language defined by this grammar. If an input expression leads to a final state of the FSM, it is a regular expression of this language. In our example, “follow me” is such a regular expression of the given language.

The fusion mechanism then takes this regular expression plus additional context information (like the scores assigned to the regular expression) to select a communicative act. However, the fusion mechanism has to consider timing and comparison issues as well. For example, two expressions from speech input may not be related to each other because of a long time interval between their utterances. And within a defined time span, other regular expressions of the same list of hypotheses could also lead to a communicative act.

The benefit for the fusion developer is twofold. First, the fusion mechanism is independent of the attached modalities. So, other modalities than the one described in this work are feasible as well. Second, when an interaction designer creates a new discourse model, the FSM is generated semi-automatically.

The remainder of this section is structured as follows. First, we describe the generation of the FSM out of the high-level discourse model. After that, we describe the fusion mechanism in more detail. Then we discuss timing issues, the comparison of the fusion hypotheses, and the final decision mechanism of the fusion. At last, we sketch the prototypical implementation that already uses this kind of fusion.

5.1 Finite State Machine Generation

We generate the FSM at compile time out of our discourse model. So, whenever an interaction designer changes the discourse model, a new FSM is created. A discourse model is modality-independent and so is the FSM. It is the annotated discourse model and not the FSM that stores the information which input modalities are allowed. In the following we describe how an FSM and its states and transitions are generated. Figure 4 illustrates this generation step-by-step using our running example.

- Consider only communicative acts from the customer (user). They are shown in yellow in Figure 2. Create a list of all the customer’s communicative acts of the discourse.
- Create a list of content specifications of the communicative acts. We assume that the content specification in a communicative act defines the corresponding regular expression. For example, the content followMe of the Informing communicative act would lead to a sequence of follow and me. Based on that, the corresponding transitions of the FSM to be generated are
labeled. The definition for one entry is \((source\ state)\rightarrow\ (target\ state)\[(label)\]. Our example results in \(s1\rightarrow s2\[\text{follow}\]\) and \(s2\rightarrow s3\[\text{meet}\]\), see Figure 4(a).

- A communicative act may contain a parameter in the content specification, like \text{putIntoCart} with the parameter of type \text{Product}. Parameters are stored as symbolic PARAM labels. The corresponding parameter values are stored in the context of the FSM instance at runtime. For example, the location of a deictic gesture with a pair of \(x\) and \(y\) coordinates is a parameter. See Figure 4(b), that shows an automaton according to our discourse model in Figure 2 up to this generation step.

- After all communicative acts have been processed, the generation algorithm searches for duplicate transitions. These are transitions with the same label. It starts from the last state — the state that only has incoming transitions — and searches recursively. In case of finding duplicate transitions, it merges all these transitions and their related paths. In our running example, the algorithm starts from state \(s3\) and creates the FSM shown in Figure 4(c).

- Then the FSM is searched for states with only one incoming and one outgoing transition. These states are removed and the labels of these transition concatenated. This leads to just one transition each. The result is an FSM to be used in the fusion mechanism for handling input hypotheses of the modalities at runtime. See Figure 4(d) for the final FSM that is generated from our discourse model.

5.2 Processing the FSM Input

At runtime, the fusion instantiates the generated FSM to process the input expressions that are part of the input hypotheses. The fusion creates, deletes and administers the FSM instances. We use the Unimod State Machine Framework\(^3\), that allows creating and administrating several instances of the FSM in memory.

We assume that at point \(t\) in time the speech modality sends a list of input hypotheses with the expressions "\text{follow me}" and "\text{meet me at}". The expressions are then input into FSM instances. We distinguish three cases, that can arise after inputting an expression and explain them using our running example.

- The FSM instance does not accept the expression. For example, we input the expression \text{follow me} into an FSM instance that has already accepted the regular expression \text{put into cart}. This means that the FSM instance is in state \(s2\) in Figure 4(d). Here, \text{follow me} does not lead to a state change.

- The FSM instance accepts the expression but does not reach the final state. For example, we input the expression \text{meet me at} into a new FSM instance. This leads to a state change of the FSM instance into state \(s2\) in Figure 4(d).

- The FSM instance accepts the expression and reaches the final state. For example, we input the expression \text{meet me at} into a new FSM instance. This leads to a state change of the FSM instance into state \(s2\) in Figure 4(d).

If an input hypothesis does not lead at least once to a state change of an already existing FSM instance, a new FSM instance is created. The hypothesis’ expression is then

\(^3\)http://unimod.sourceforge.net/
the input for this new FSM instance. At last, the FSM instances are compared with other finished FSM instances, before they can lead to the selection of a communicative act for the fusion mechanism.

5.3 Timing

A major task of the fusion mechanism is timing. The time constants used in the following are determined through heuristics. We have a time triggered approach for our active and finished FSM list management and check with short intervals for changes. Every 100 milliseconds the fusion mechanism checks if active FSM instances exceed their allowed lifetime. In such a case they are removed from the list of active FSM instances. The timing component also checks every 100 milliseconds if any active FSM instances have reached the final state. The active FSMs are then moved to the list of finished FSMs and removed from the list of active FSMs. A finished FSM instance may remain 500 milliseconds in state ‘finished’. If during this time one or more FSMs are finished that are from the same set of hypotheses, then a heuristic score comparison is used to choose a fusion hypothesis. A fusion hypothesis is the regular expression of the FSM instance plus context information. If no other fusion hypotheses arrive during this time span, a communicative act is created with the fusion hypothesis as a basis.

The input of a modality at a given point in time has a unique identifier. The identifier of such a hypotheses list is stored in the context of an FSM instance if one expression of this list leads to a state change. There exists a history of sent communicative acts and their corresponding sets of unique identifiers. In case a finished FSM instance has the same set of unique identifiers like an entry in the history, it is then dropped and no fusion hypothesis sent.

5.4 Score Comparison and Communicative Act Creation

We need to apply a score comparison if two FSM instances are in the finished list at the same time. Figure 5 illustrates an example of the fusion score comparison. The figure shows three hypotheses, with their expressions (RegEx) and their scores. At point in time $t_1$, the hypotheses $A_1$ and $B$ appear. At point $t_2$ in time, hypothesis $A_2$ appears, that has a higher score than hypothesis $A_1$. The score comparison mechanism compares the scores of the fusion hypotheses.

The higher the score of a fusion hypothesis, the more likely it is the user input. So, the FSM with the higher score leads to a communicative act, whereas the one with the lower score is removed. In our figure only hypotheses $A_2$ and $B$ are successful, leading to communicative acts. Hypothesis $A_1$ is related to hypothesis $A_2$, but has a lower score. In contrast, hypothesis $B$ is not related to any other hypothesis as no other hypothesis has its unique identifier in the list of finished FSM instances.

5.5 Prototypical Implementation

The fusion mechanism is implemented in a software service of our communication platform based on Open Services Gateway Initiative (OSGI). Generally, the fusion service is active throughout the whole lifetime of the platform, waiting and processing new input events from the modalities.

The multimodal user interface and the fusion mechanism described in this work are implemented in a semi-autonomous shopping cart research robot. This implementation includes GUI, speech, hand gestures and a bar code reader as input modalities. The fusion mechanism for input uses the algorithms presented above.

6 Fulfillment of CARE Properties

Today’s multimodal user interfaces typically try to fulfill the CARE properties. This is valid for user interfaces generated with a model transformation process as well, as the work in [13] shows. Our semi-automatic approach to the generation of multimodal user interfaces takes these CARE properties into account, too. They are dealt with by the discourse, the modality provider and the fusion mechanism.

The discourse model constrains the used communicative acts, thus it indirectly influences possible combinations of modalities. The modality provider can directly (deliberate support of functionality in several modalities) or indirectly

![Figure 5. Score Comparison of Hypotheses.](http://www.eclipse.org/osgi/)
(not aware of functionality in other modalities) influence the CARE properties.

The fulfillment of the CARE properties through our fusion mechanism is illustrated in Figure 6. The fusion mechanism with its timing behavior is responsible for recognizing if two or more hypotheses from one or more modalities are related. Each of the properties is depicted with potential input of (different) modalities at two sequenced points in time. The complementarity property is depicted with GUI input "meet_me_at" at point in time $t_1$ and a complementary speech input "apples" at point in time $t_2$. The expressions from two different modalities jointly lead to the regular expression "meet me at apples". The delta of $t_1$ and $t_2$ has to be smaller than three seconds. Otherwise the fusion mechanism is not able to recognize the input of the modalities as being related. This timing constraint is valid for the other properties as well.

As mentioned in [2], the assignment property can either be controlled by the user or by the robot trolley. The user ‘just’ uses one distinct modality for one type of communicative act. So, in contrast to complementarity, we show the assignment property with the modality speech only. So, the two, potentially independently said, expressions "meet me at" and "apples" lead to the regular expression "meet me at apples". This is the assignment property from the users’ perspective. Viewed from the other side, the system may force the user to use one modality for input. E.g., if the environment is too loud, the speech input may not be used any more.

If two modalities have the same regular expression "meet me at", the fusion allows redundancy and equivalence. Figure 3 illustrates redundancy with input from speech and GUI at $t_1$ and $t_2$, allowing for a more robust multimodal user interface. Finally, equivalence is possible if the user chooses speech input for "follow me" at one point in time, and then uses the GUI button, that triggers the hypothesis "follow me" at another point in time (e.g., 1 minute later).

7 Conclusion

In this paper, we present an approach to semi-automatic generation of high-level fusion for multimodal user interfaces. A first prototypical implementation of this approach already exists in a semi-autonomous shopping cart research robot, and it should be general enough for other applications as well.

According to our best knowledge, the following is new in our approach. We are able to generate an FSM for use in fusion from a high-level discourse model, where both the discourse model and the FSM are modality-independent. A modality-dependent annotated discourse for influencing the fusion mechanism can be generated semi-automatically, using information about binding of input modalities provided by humans.

The pragmatic implications for the designer are that key parts are generated automatically, so that effort may be saved. There is more focus on the interaction design than on low-level integration of new modalities, which can be linked in easily in our approach.

8 Acknowledgment

This research has been carried out in the CommRob project (http://www.commrob.eu), partially funded by the EU (contract number IST-045441 under the 6th framework programme).

References


