

RESEARCH OUTPUTS / RÉSULTATS DE RECHERCHE

Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems

Dumas, Bruno

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for pulished version (HARVARD):

Dumas, B 2010, 'Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems', Ph.D., Université de Fribourg, Fribourg, Suisse.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Department of Informatics
University of Fribourg (Switzerland)

**FRAMEWORKS, DESCRIPTION LANGUAGES AND FUSION
ENGINES FOR MULTIMODAL INTERACTIVE SYSTEMS**

THESIS

Presented to the Faculty of Science, University of Fribourg (Switzerland)
in consideration for the award of the academic grade of
Doctor Scientiarum Informaticarum

by

Bruno Dumas

from

Vuisternens-devant-Romont FR (Switzerland)

Thesis N° 1695

UniPrint, Fribourg

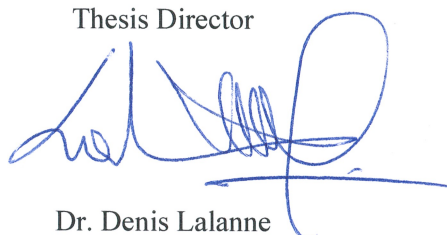
2010

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) upon the recommendation of

- Prof. Ulrich Ultes-Nitsche, University of Fribourg, Switzerland (Jury President)
- Dr. Denis Lalanne, University of Fribourg, Switzerland (Thesis Director)
- Prof. Rolf Ingold, University of Fribourg, Switzerland (Expert)
- Prof. Laurence Nigay, Université Joseph Fourier, Grenoble, France (External Expert)
- Prof. Philippe Palanque, Université Paul Sabatier, Toulouse, France (External Expert)

Fribourg, November 16th, 2010

Thesis Director



Dr. Denis Lalanne

Faculty Dean



Prof. Rolf Ingold

Table of Contents

Table of Contents	5
Remerciements	9
Abstract.....	11
Résumé	13
1 Introduction	15
1.1 New Trends in Human-Machine Interaction	15
1.2 Multimodal Interaction: Promises and Challenges.....	16
1.3 Specific Challenges Addressed in this Thesis	18
1.4 Thesis Context	19
1.5 Thesis Structure	19
2 Conceptual Background.....	23
2.1 Foundations, Aims and Features of Multimodal Interaction.....	24
2.1.1 Definition of Multimodality	24
2.1.2 Key Features of Multimodal Interaction.....	27
2.1.3 Cognitive Foundations of Multimodal Interaction	28
2.1.4 Seminal Works, Findings and Guidelines	31
2.2 Multimodal Human Machine Interaction Loop.....	33
2.3 Design Spaces and Theoretical Frameworks for Reasoning on Multimodal Interaction	35
2.3.1 The TYCOON Theoretical Framework.....	36
2.3.2 The CASE Design Space	36
2.3.3 The CARE Properties	38
2.4 Conclusion	39

3	Multimodal Interfaces Prototyping Frameworks & Architectures	41
3.1	Terminology.....	42
3.2	Related Work	43
3.3	Multimodal Interaction Creation Tools: Prominent Features Related to Fusion	47
3.3.1	Multimodal Interaction Creation Tools: Input Management	47
3.3.2	Multimodal Interaction Creation Tools: Architecture features.....	48
3.3.3	Multimodal Interaction Creation Tools: Dialog Description & Programming.....	49
3.3.4	Multimodal Interaction Creation Tools: Fusion Engine Characteristics	51
3.3.5	Multimodal Interaction Creation Tools: Miscellaneous Features.....	51
3.3.6	Final Analysis	52
3.4	Multimodal Interfaces: a Study of Architectures.....	53
3.4.1	Computational Architecture and Key Components	53
3.4.2	Java Swing MM extension: Usability	55
3.4.3	SCS, the Service Counter System: Expressiveness	57
3.5	The HephaisTK framework, and its Architecture.....	60
3.5.1	Initial Requirements for HephaisTK.....	61
3.5.2	General Architecture of HephaisTK	62
3.5.3	Usability and Expressiveness: Looking for Balance.....	68
3.6	Usability, Expressiveness and Prototyping Multimodal Systems: an Outlook.....	68
3.6.1	Summarization and Observations	68
3.6.2	Positioning HephaisTK	69
3.7	Conclusion	69
4	Multimodal Interaction Modeling	73
4.1	Related Work	74
4.2	Spectrum of Multimodal Dialog Description Languages	78
4.3	Multimodal Dialog Description: the Synchronisation Problem.....	80
4.4	Guidelines For Languages for Describing Multimodal Dialog	81
4.5	The SMUIML Description Language.....	83
4.5.1	Structure of the Language	84

4.5.2	Recognizers.....	86
4.5.3	Triggers.....	87
4.5.4	Actions.....	89
4.5.5	Dialog	90
4.5.6	SMUIML language interpretation	91
4.6	Positioning of SMUIML.....	92
4.7	Conclusion	93
5	Multimodal Fusion: Dialog Management & Algorithms	95
5.1	Multimodal Fusion of Input Data: A State of the Art.....	96
5.1.1	Levels of Fusion of Input Modalities	96
5.1.2	An Historical View on Fusion Engines	98
5.1.3	Dialogue vs Fusion Management: a Discussion.....	98
5.1.4	Dialogue Management.....	100
5.1.5	Algorithms for Fusion of Input Modalities.....	101
5.2	Time Synchronization Issues.....	104
5.3	Fusion Handling in HephaïsTK: Affected Components and General Organization	106
5.4	Fusion Algorithms in HephaïsTK.....	108
5.4.1	Meaning Frame-based Fusion of Input Data	109
5.4.2	HMM-Based Fusion of Multimodal Input.....	113
5.5	A Benchmarking Tool for Assessment of Fusion Algorithms	116
5.5.1	A Benchmarking Tool for Fusion Engines.....	117
5.5.2	Metrics and Software Requirements.....	120
5.6	Conclusion	121
6	Evaluations	123
6.1	SMUIML Modeling Evaluation	124
6.2	Fusion Evaluation	126
6.3	HephaïsTK Evaluation Through Use Cases	130
6.3.1	HephaïsTK Programming Tutorial Examples	131
6.3.2	Proof of Concepts	133

6.4	Conclusion	135
7	Conclusions & Perspectives.....	137
7.1	The Three-Headed Hydra of Multimodal Input Fusion	137
7.1.1	The First Head: Software Architecture for Fusion of Input Data	138
7.1.2	The Second Head: Multimodal Human-Machine Dialog Modeling.....	139
7.1.3	The Third Head: Fusion Engine Algorithms.....	139
7.2	Perspectives & Future Work	140
7.2.1	The HephaistTK framework: Perspectives	140
7.2.2	Multimodal Dialog Modeling: Perspectives	141
7.2.3	Fusion Algorithms: Perspectives	141
7.3	Wrap Up.....	142
	Bibliography.....	143
	Table of Figures	151
	Table of Tables.....	155
	Appendix 1: XML Schema Definition of SMUIML	157
	Appendix 2: Benchmark SMUIML and EMMA Files.....	163
	Appendix 3: Multimodal Music Player Example: SMUIML and Java Files	169
	Appendix 4: XPaint Drawing Table SMUIML Configuration File.....	175

Remerciements

Cette thèse aura rythmé cinq années de ma vie, durant lesquelles j'ai eu la chance de pouvoir m'appuyer sur de nombreuses personnes, collègues comme proches. A mon tour de leur témoigner ma reconnaissance.

Avant toute chose, toute ma gratitude va au professeur Rolf Ingold, qui a accepté de m'accueillir dans son groupe de recherche en tant qu'assistant-doctorant et qui m'a accordé sa pleine confiance durant ces années. Immense merci également au Dr Denis Lalanne, mon directeur de thèse, pour son soutien sans faille au cours de ces cinq années, et ce malgré mes questions existentielles du vendredi soir. Merci aux professeurs Laurence Nigay et Philippe Palanque d'avoir accepté de faire partie de mon jury de thèse : leur longue connaissance du domaine a beaucoup apporté à ce texte. Merci aussi au professeur Ulrich Ultes-Nitsche d'avoir assuré avec brio la présidence lors de mon examen de thèse.

Je remercie évidemment les membres du département d'informatique de l'université de Fribourg (DIUF), et tout particulièrement les membres du groupe DIVA pour leur enthousiasme communicatif ; un clin d'œil particulier à Catherine et Florian qui, non contents d'avoir dû me supporter pendant nos études communes, ont rempli pendant cinq ans de plus ! Merci également à la Dr Agnès Lisowska d'avoir gentiment accepté de relire les parties principales de cette thèse.

Un travail de doctorat se vivant vingt-quatre heures par jour, mes amis ont enduré plus d'une fois mes plaintes larmoyantes avec un stoïcisme et une empathie dont je leur serai éternellement reconnaissant. Merci à tous, vous savez à quel point vous m'êtes chers.

Finalement, je remercie ma famille pour son support absolu au cours de ces années, et tout particulièrement Marie-Luce, ma mère, Jean-Paul, mon père, et Clélia, ma sœur. Et pour terminer, je remercie Estelle, qui m'a beaucoup supporté (dans tous les sens du terme) au cours des années les plus pénibles de ce doctorat.

Abstract

The field of multimodal interaction grew during the last decade, as a consequence of the advent of innovative input interfaces, as well as the development of research fields such as speech recognition. However, multimodal fusion and combination did not evolve at the same rate, which lead to a chasm between the use of input modalities and the different possibilities of combining them. This PhD thesis seeks to reduce the chasm between interaction means and fusion of their data.

Fusion of multimodal input is approached in a global way in this research: first, from the point of view of the architecture of a multimodal system as a whole, then, from the point of view of multimodal dialog modeling, and finally from an algorithmic point of view. The architectural angle focuses on necessary features of an architecture to allow beneficial integration of a fusion engine, using usability and expressivity as attributes to characterize qualities and drawbacks of different architectures. Three different architectures targeting the creation of multimodal interfaces are subsequently studied. The HephaisTK framework, which served as the experimental foundation for this thesis work, is described in detail. The second angle, dialog modeling oriented, presents eight guidelines for creation of multimodal dialog modeling languages, then proposes the SMUIML language as an example of a language following those guidelines. Finally, the algorithmic angle studies multimodal fusion itself through the implementation of two fusion algorithms: an algorithm based on meaning frames and an algorithm based on hidden Markov models (HMMs). An evaluation of the performances of those two algorithms with the help of a proposed integrated benchmarking tool is also presented.

Through theoretical as well as practical study of these three angles – architecture, modeling, algorithms – the different issues of multimodal fusion were defined and clarified. This newly elucidated knowledge combined with the software framework implemented in the scope of this thesis work, should allow further research on two subjects we consider important to study in the future: error management, and adaptation to user.

Résumé

Le domaine de l'interaction multimodale a pris beaucoup d'importance lors de cette dernière décennie, du fait de la création et de la mise sur le marché de nombreuses interfaces novatrices, ainsi que de l'évolution de domaines de recherche tels que la reconnaissance vocale. Cependant, la fusion et la combinaison de modalités n'ont pas évolué avec la même vigueur, aboutissant à un fossé entre l'utilisation de modalités d'entrée et les différentes possibilités de les combiner. Cette thèse de doctorat s'attache à réduire ce fossé entre les dispositifs d'interactions à disposition et la fusion de leurs données.

Dans cette recherche, la fusion d'inputs multimodaux est abordée dans sa globalité : en premier lieu, sous l'angle de l'architecture d'un système multimodal dans son ensemble, puis sous l'angle de la modélisation de la fusion multimodale et, enfin, sous un angle plus algorithmique. Le premier angle, architectural, s'intéresse aux exigences nécessaires pour une intégration profitable de moteurs de fusion, en utilisant les critères d'utilisabilité et d'expressivité pour caractériser les qualités et les inconvénients de différentes architectures. Trois différentes architectures pour la création d'interfaces multimodales sont ensuite étudiées. Le framework HephaïstTK, qui a servi de base d'étude expérimentale pour ce travail de thèse, est finalement décrit. Le deuxième angle, la modélisation du dialogue, présente huit directives pour la création de langages de modélisation du dialogue multimodal et propose le langage SMUIML comme exemple d'un langage suivant ces directives. Le troisième angle, algorithmique, étudie la fusion multimodale proprement dite par le biais de l'implémentation de deux algorithmes de fusion : d'abord, un algorithme « meaning frames », ensuite un algorithme basé sur les modèles de Markov cachés (HMM). Une évaluation des performances de ces deux algorithmes à l'aide d'un outil de mesure intégré est présentée en dernière analyse.

Par l'étude à la fois théorique et expérimentale de ces trois axes – architectural, modèle, algorithmique –, les différentes problématiques de la fusion multimodale ont été délimitées et clarifiées. Ces nouvelles connaissances établies, ainsi que la plate-forme logicielle implémentée dans le cadre de ce travail de thèse, devraient permettre d'étudier à l'avenir deux thèmes importants à nos yeux : la gestion des erreurs ainsi que l'adaptation à l'utilisateur.

1 Introduction

“Archimède fut le premier à démontrer que, lorsqu'on plonge un corps dans une baignoire, le téléphone sonne.”

Pierre Desproges, “Vivons heureux en attendant la mort.”

1.1	New Trends in Human-Machine Interaction.....	15
1.2	Multimodal Interaction: Promises and Challenges	16
1.3	Specific Challenges Addressed in this Thesis.....	18
1.4	Thesis Context.....	19
1.5	Thesis Structure.....	19

Twenty-six years ago, the well-known WIMP paradigm (*Window, Icon, Menu, Pointer*), developed at Xerox PARC, was popularized by the Apple Macintosh computer; this led to the complete adoption of the WIMP paradigm as the main human-machine style of interaction in the 80's, and which is still used nowadays. At about the same time (1981-1983), Richard A. Bolt was exploring a novel style of interaction, which would become known as “multimodal interaction”. This style, taking advantage of the parallel processing capabilities of human beings, was only sparsely studied until the late 90's [61]. It then resurfaced, as recognition of specific modalities (such as speech or gesture) made considerable progress and computing processing capabilities grew.

1.1 New Trends in Human-Machine Interaction

The end of the 90's saw the appearance of a number of completely new interaction styles, such as tangible interaction [46], augmented or mixed reality [119], and multimodal interaction. These interaction styles, first growing on their own, have recently been gathered under the umbrella of “Reality-based interaction”.

Jacob et al. [47] consider the following four themes as characteristic features of reality-based interaction:

- Naïve physics
- Body awareness & skills
- Environment awareness & skills
- Social awareness & skills

In short, these interaction styles take full advantage of their users' knowledge and skills to improve human-machine interaction. For example, tangible interaction is based on "humans' most basic knowledge about the behavior of the physical world" [114]. As another example, smartphones like the iPhone or Android-based phones make use of naïve physics such as inertia or springiness in their user interfaces. Scaled-up versions of multitouch interactions, like the Microsoft Surface or CircleTwelve (ex-MERL) DiamondTouch tables, promote multi-user interaction, thus encouraging their users to not only simply use the device, but also to collaborate, making active use of social skills. All these new styles of interaction target a smoother communication between humans and machines, by taking full advantage of the innate capabilities of human beings.

1.2 Multimodal Interaction: Promises and Challenges

As a prominent member of reality-based interaction styles, multimodal interfaces seek to promote a more "human" way to interact with computers. Amazingly enough, multimodal interfaces were first seen as potentially more efficient and quicker than standard WIMP interfaces. Studies [87] proved this belief wrong, but also showed that multimodal interfaces were more robust and more stable [88] than their WIMP counterparts; furthermore, multimodal interfaces were preferred by a wide majority of users. This should not come as a surprise, when considering multimodal interaction as a member of such reality-based interaction styles presented before. Indeed, multimodal interfaces promise to take full advantage of naïve physics (multitouch interfaces), body awareness & skill (gesture & speech interfaces), environment awareness & skills (plasticity), as well as social awareness & skills (collaboration, as well as emotion-based interfaces).

However, these promises do not come for free. By their very nature, multimodal interfaces make use of any number of modalities, and frequently state of the art modalities, requiring developers to have a wide range of skills in such unrelated domains as software engineering, human-machine interaction, artificial intelligence, machine learning or document engineering. In particular,

a number of specific challenges have arisen in the last ten years, identified by prominent voices in the domain. Oviatt et al. [90] consider the following four research directions: *new multimodal interface concepts*, such as blended interface styles combining both passive and active modes; *error handling techniques*, such as higher levels of mutual disambiguation, or new language and dialogue processing techniques; *adaptive multimodal architectures*, that is, systems able to continually adjust themselves to the users and their surroundings and, finally, *multimodal research infrastructures*, such as semi-automatic simulation methods for empirical data collection and prototyping of new systems, and software tools that support the rapid creation of next-generation multimodal interfaces. On his side, Garofolo identifies in [40] the following technological challenges: *data resources and evaluation*, as few multimodal corpora currently exist, making thorough evaluations even more difficult to achieve and compare; *core fusion research*, such as novel statistical methods, knowledge and data representation methods, heuristics and algorithms and, finally, *driver applications*, needed to motivate and guide research directions.

Summarizing these points of view, and blending them with the features of Reality-Based Interaction presented in section 1.1, spawns a complete list of challenges specific to the design of multimodal interaction, of which we extract the following subset, which we believe to be a representation of the most important challenges in the field:

- *Architectures* for multimodal interaction are needed because of the concurrent nature of these types of interactions. In particular, tools to help quickly design and prototype multimodal interaction are required if multimodal interaction is to become more mainstream.
- *Modeling* the human-machine dialog is a delicate task, because of the complex nature induced by the presence of multiple input and output modalities.
- *Fusion of input modalities* is a research domain, tightly linked to human-machine dialog modeling, in which little research has been done, in particular concerning effective fusion algorithms able to take into account the three following aspects.
- *Time synchronicity* demands from the fusion and dialog manager in a multimodal system the ability to take into account, and adapt to multiple input modal commands which combination can trigger different meanings, following their order, and delay between them.
- *Plasticity/adaptivity to user & context* denotes the capability of a human-machine interface to adapt to both the system's physical characteristics and to the environment while preserving usability [111].
- *Error management* has been the poor relation of multimodal interaction research since the beginning, with developers assuming that users will behave in perfect accordance with the way

the system expects them to behave, and that no unwanted circumstance will appear. Evidently, this is not the “real life” case, and error management will have to be carefully handled if multimodal interfaces are to be used broadly.

- *User feedback* is somewhat related to error management, in that the user is allowed to correct or adjust the behavior of the multimodal system in real time.

1.3 Specific Challenges Addressed in this Thesis

It would not be possible to address every single one of the challenges presented in section 1.2. We therefore focus on the third challenge, i.e. *fusion of input modalities*. Choice was made to focus on fusion of input modalities because it also entails time synchronicity issues, and is the necessary foundation for research on error management, as well as plasticity. Furthermore, we chose to consider the problem from three different points of view, two of which partly discuss the challenges of *architectures* and *modeling*, and all of which also address the problem of *Time synchronicity*.

First, fusion of input modalities can be achieved at a number of different levels of abstraction, as well as considering increasing levels of complexity. However, not all software architectures are able to support those different abstraction and complexity levels with the same ease. Thus, the first part of this thesis was a study of different software architectures for tools dedicated to prototyping of multimodal interface and the capabilities of said architectures, in particular in view of fusion of input modalities. This first axis is discussed in Chapter 3.

Second, a fusion engine is frequently managed by a set of rules. Yet, existing ways to script multimodal human-machine interaction are for the most part strongly web-oriented and lack important features directly related to fusion of input modalities, such as time synchronicity. Thus, the second part of this thesis work was dedicated to multimodal interaction modeling. This second axis is discussed in Chapter 4.

Finally, in the core of a fusion engine resides logic and algorithms used to integrate data coming from different input recognizers into an application-usable result. A study of different algorithms for fusion of input modalities, as well as the design and implementation of an hidden Markov models (HMM)-based algorithm, comprise the third part of this thesis. This third axis is discussed in Chapter 5.

1.4 Thesis Context

This thesis took place jointly in the context of the IM2.HMI (Human-Machine Interaction) work package of the IM2¹ (Interactive Multimodal Information Management) NCCR project, and of the MeModules² project.

The work package on “Human Machine Interaction” is part of the The Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2). While other activities in IM2 develop multimodal analysis and recognition technologies, the main goal of the IM2.HMI work package was to design, develop and evaluate, with human subjects, novel interactive multimodal meeting browsers/assistants. The IM2 NCCR is one of the 20 Swiss National Centers of Competence in Research (NCCR), and reached its 3rd phase in the end of 2009. Shifting to the 3rd phase of the project (2010-2013) saw the IM2.HMI work package transfer to the IM2.IP2 project, dedicated to human-centered design and evaluation.

The MeModules project [76] had the objective of developing, experimenting and evaluating the concept of tangible shortcuts to multimedia digital information. Moreover, it investigated the opportunity of a more complex, multi-sensorial combination of physical objects with multimedia information by associating tangible interaction with multiple other interaction modalities such as voice, gesture, etc. One of the research outcomes of the project was to assess which modalities are best combined with tangible interaction depending on the context and application.

1.5 Thesis Structure

Following the problems raised in the preceding paragraphs, we decided to address them by creating a tool that would help developers when designing multimodal interactions. Thus, the theoretical challenges addressed by this thesis, presented in section 1.3, will be illustrated with the framework implemented during this PhD thesis. This framework for creating multimodal interfaces, named HephaïsTK, will in consequence be quoted in most sections of this thesis, as it accompanied the different research topics presented in this document.

¹ See <http://www.im2.ch/> (accessed 06.08.2010).

² See <http://www.memodules.ch/> (accessed 06.08.2010).

As presented in section 1.3, three closely related topics will in particular be investigated in depth in the following chapters: *multimodal interface tool architectures*, *multimodal dialog description languages* and *multimodal input modality fusion*. The whole thesis document is structured around these three axes, as illustrated in Figure 1.

Following this introduction, **Chapter 2** will focus on presenting the conceptual background behind multimodal interaction. Foundations, aims and features of multimodal interaction will be discussed, in particular with a definition of multimodality, presentation of key features, the cognitive foundations underlying multimodal interaction, as well as seminal works, findings and guidelines. Based on Donald Norman's action cycle [83], a multimodal human machine interaction loop will be introduced; then, two influential design spaces for multimodal interaction, the CASE design space and the CARE properties, will be detailed.

	Axis 1: Architectures for Prototyping Frameworks	Axis 2: Multimodal Interaction Modeling	Axis 3: Multimodal Fusion Dialog & Algorithms
Ch. 1: Introduction	Introduction	Introduction	Introduction
Ch. 2: Conceptual Background	Conceptual background	Conceptual background	Conceptual background
Ch. 3: Prototyping Frameworks & Architectures	State of the Art Proposed solution		
Ch. 4: Multimodal Interaction Modeling		State of the Art Proposed solution	
Ch. 5: Multimodal Fusion: Dialog Management & Algorithms			State of the Art Proposed solution
Ch. 6: Evaluations	Evaluation	Evaluation	Evaluation
Ch. 7: Conclusions & Perspective	Conclusion	Conclusion	Conclusion

Figure 1. Structure of this thesis.

Creation of multimodal interface prototyping tools, and architectures for such tools, encompasses a number of issues that go beyond mere engineering difficulties. Indeed, careful examination of the state of the art reveals that a number of different approaches and architectures have been used for the creation of such tools. These different approaches all target the same broad goal of helping multimodal interface prototyping, but their dissimilarities are tied to architectural, as well as theoretical concepts that, when chosen, will define the type of multimodal application that may be created with the given tool. **Chapter 3**, dedicated to *Multimodal Interface Prototyping Frameworks and Architectures* will focus on a number of those theoretical and architectural concepts

relevant to such tools, and explore the influences each of those concepts has on the creation space available in a prototyping tool. In particular, after a review of the related work and a study of specific features of said related work, this thesis will look at different software architectures on which such tools can be built. Finally, the HephaisTK framework will be presented, its software architecture will be thoroughly studied, and HephaisTK will be positioned in relation to the concepts viewed in this chapter.

Chapter 4 will focus on *multimodal interaction modeling*. Although it appears to be a very different topic to the former one, multimodal dialog description is in fact closely tied to multimodal interface prototyping. Indeed, any tool allowing the creation of multimodal interfaces will need to provide its user with some way to model multimodal interaction between the human and the machine. Thus, most current instantiations of multimodal dialog descriptions revolve either around a specific tool for creation of multimodal interfaces, or around web-based multimodal applications. Such web-based multimodal applications make for an interesting case of multimodal application creation, as XML-based languages play the roles of modeling, programming and interpreting, while each being distinct of the other languages. Modeling languages will then have the task to tie the different languages to one another, while providing the developer with a means to model the human-machine multimodal interaction. After review of the related work, this chapter will focus on what could be called the multimodal input synchronization problem, which any multimodal dialog description should offer the means to describe and handle. This chapter will present the CARE properties introduced in subsection 2.3.3 as a means to describe in a formal way the different cases of multimodal input synchronization. The chapter will then provide a use case of a multimodal dialog description language with SMUIML (*Synchronized Multimodal User Interaction Modeling Language*), an XML-based language used to script the behavior of the HephaisTK framework. Thus, SMUIML and its main characteristics will be reviewed, as well as the way in which SMUIML handles the synchronization problem. Based on the state of the art and experiments done with SMUIML, nine guidelines for languages for multimodal interface creation tools have been derived, and will be detailed in this chapter.

Ultimately, numerous features of multimodal dialog description languages take on their full significance when linked with fusion algorithms able to take into account advanced concepts such as synchronization of multimodal input data. However, such fusion algorithms for multimodal interaction have been sparsely studied in the last decades. **Chapter 5** will thus focus on fusion principles, as well as fusion algorithms implemented and tested in HephaisTK. After a review of the state of the art on fusion principles and fusion algorithms, as well as a discussion on the roles taken

by fusion and dialog management in a multimodal architecture, SMUIML interpretation in HephaïsTK will be detailed, as it entails potential fusion schemes and their implementation. Then, the specific fusion schemes implemented in HephaïsTK will be described, as well as their features. In particular, two fusion algorithms implemented in HephaïsTK will be surveyed in detail: a meaning frame-based fusion algorithm, as well as an HMM-based fusion algorithm. Presentation of a benchmarking tool integrated in HephaïsTK concludes this chapter.

After presentation and study of each of the three axes around which this thesis is built, **Chapter 6** is dedicated to the evaluation of those three axes. Evaluation of our solution for multimodal interaction modeling, the SMUIML language, based on a qualitative user study, is first presented. Evaluation of the fusion algorithms, based on the benchmarking tool introduced in Chapter 5, follows closely. Evaluation of the HephaïsTK framework, based on a set of applications created with help of the HephaïsTK framework, is detailed afterward.

This thesis ends with Chapter 7, which is composed of a general conclusion, as well as perspectives and future work related to the different axes presented in this thesis.

2 Conceptual Background

“知彼知己，勝乃不殆；知天知地，勝乃可全。”

“So it is said that when you know yourself and others, victory is not in danger; when you know sky and earth, victory is inexhaustible.”

Sun Tzu, “The Art of War”.

2.1	Foundations, Aims and Features of Multimodal Interaction.....	24
2.1.1	Definition of Multimodality	24
2.1.2	Key Features of Multimodal Interaction	27
2.1.3	Cognitive Foundations of Multimodal Interaction	28
2.1.4	Seminal Works, Findings and Guidelines	31
2.2	Multimodal Human Machine Interaction Loop	33
2.3	Two Design Spaces for Reasoning on Multimodal Interaction	35
2.3.1	The CASE Design Space.....	36
2.3.2	The CARE Properties	38
2.4	Conclusion.....	39

This thesis focuses on multimodal interaction, and particularly on fusion of multimodal input data. This chapter will thus give a view of foundations of multimodal interaction, as well as introduce conceptual background relevant to the topics of this thesis³.

³ Part of this chapter, as well as “related works” sections of some of the next chapters, are derived from the following reference: Dumas, B., Lalanne, D., Oviatt, S. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In Denis Lalanne, Jürg Kohlas eds. Human Machine Interaction, LNCS 5440, Springer- Verlag, pp. 3-26 (2009) [33].

The chapter will begin by giving a view of foundations, aims and features of multimodal interaction; cognitive foundations, as well as seminal works, findings and guidelines will help underline the different challenges presented by multimodal interaction as a research topic. Then, a standard model of the multimodal interaction man-machine loop will be detailed; next, multimodal interaction will be formalized with help of the CARE and CASE models.

2.1 Foundations, Aims and Features of Multimodal Interaction

2.1.1 Definition of Multimodality

Broadly speaking, multimodal interaction can be thought as a sub branch of human-computer interaction using a set of modalities to achieve communication between user(s) and machine(s). This first, naïve definition immediately raises the issue of defining what a “modality” is. We will first consider the following characterization of a modality, as being

A channel or path of communication between the human and the computer.

The definition is purposely fuzzy, as the notion of modality discusses a number of different aspects of a given communication channel between a human and a computer. At a high level, one can refer to a modality as one of the different senses through which the human can perceive the output of the computer (audition, vision, touch, smell, taste and, from those, proprioception, thermoception, nociception & equilibrioception); conversely, modalities will cover the range of input devices and sensors allowing the computer to receive information from the human.

Some authors prefer to pinpoint the medium-like, transferring of data between non-digital and digital world characteristic of the concept of modalities. For example, Nigay gives this formal definition of a modality [82]:

“A modality m is defined as the couple $m = \langle r, d \rangle$ where r denotes a representational system, and d the physical I/O device used to convey expressions of the representational system.”

On definitions of multimodal interaction, one has been formulated in the following way by Oviatt [85]:

“Multimodal interfaces process two or more combined user input modes (such as speech, pen, touch, manual gesture, gaze, and head and body movements) in a coordinated manner with multimedia system output. They are a new class of interfaces that aim to recognize naturally occurring forms of human language and behavior, and which incorporate one or more recognition-based technologies (e.g. speech, pen, vision).”

To sum up, a multimodal interface needs at least two communication modes (or streams) between a user and a machine, that will be used to provide (or improve) interaction between both.

Although being one of the most thorough definitions of multimodal interaction, a few remarks can be raised about the quote above. First, Oviatt assumes only combination of *input* modes as valid multimodal interaction, speaking only of “multimedia system output”; one could object though that multimodality on the output side has already been achieved, and is a sprawling research domain⁴. Furthermore, Oviatt assumes at least one of the modalities to be a “recognition-based technolog[y]”; once again, one could see this definition as somewhat restrictive⁵. Nevertheless, these two remarks should not hide the strong points of Oviatt’s definition:

- “Two or more [...] input modes”: this is (evidently) the core of multimodal interfaces, denoted by their very designation; however, only taking into account numerous input streams is not enough to identify an interface as being multimodal; hence the following precision:
- “Combined [...] input modes [...] in a coordinated manner”: one of the main advantage of multimodal interfaces over other types of human-machine interaction is their ability to combine multiple input modes, and extract from this very combination a richer and/or more robust interpretation, or consider these very same input modes in a non-combined, exclusive or equivalent manner. “Coordinated manner” also underlines the importance of time synchronicity in multimodal interaction, which will be examined in the following chapters.
- “Naturally occurring forms of human language and behavior”: as will be seen later in this chapter, multimodal interactions are based on numerous findings in cognitive psychology; this potentially strong connection to natural human language and behavior is another powerful quality of multimodal interaction.

⁴ The interested reader will find information on multimodal fission in reference [38], a good example of multimodality on the output side in reference [1], as well as in Frédéric Vernier’s PhD thesis [115].

⁵ One could assume that Oviatt wanted to avoid including in her definition “classic” couples of input modalities such as keyboard/mouse, hence enforcing the novelty of multimodal interfaces.

Hence, the objective of multimodal interfaces is twofold: (1) to support and accommodate users' perceptual and communicative capabilities; and (2) to integrate computational skills of computers in the real world, by offering more natural ways of interaction to humans [80][90].

At this point, it is worth mentioning that multimodality is not a specific feature of human-machine interaction. Melichar [71] considers four different ways of understanding the term "multimodal"⁶:

- Different communication interfaces that allow the users to select the one (or the combination) they prefer.
- Multiple coordinated streams generated by the user, and processed by the machine to enhance robustness of recognition; for example, lips movement and speech modalities processed together in order to improve speech recognition.
- Virtual streams of information coming from a same physical stream; for example, speech signal can be processed at a linguistic, prosodic or emotional level.
- Set of independent streams of information, processed by the machine in order to get context information; an example would be the location of a user in a room.

As can be seen, multimodal streams of information can come either from voluntary action from the user (-> active modality), or be extracted from the environment, be it context and/or user (-> passive modality) [12]; furthermore, those multimodal streams can be processed either in an online or an offline fashion. Biometrics, in particular, have been known using multimodal data to enhance recognition rates and protection against forgeries. Examples of multimodality in biometrics include, for example, speech and signature for user verification [39], or speech and handwriting for user authentication [45]. More detail on multimodal biometric data fusion can be found in [99].

More broadly, multimodal data (recorded, for example, in a meeting) can be analyzed concurrently, as well as in an offline or online manner, in order to get richer and more robust information over context and participants. In Switzerland, the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2)⁷ aims at developing natural multimodal interfaces for human-computer interaction and to foster collaboration, focusing on new multimodal technologies to support human interaction, in the context of smart meeting rooms and remote meeting assistants.

⁶ Cf. Melichar [71], p. 31.

⁷ The Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2) is one of the 20 Swiss National Centers of Competence in Research (NCCR). See Section 1.4 for more information.

It is necessary to stress that only *multimodal interaction*, that is, the online human-machine facet of multimodality, is of relevance in the context of this work. This fact enforces a certain number of specific points of interest, compared to other aspects of multimodality. In particular, features such as composition between modalities, real-time adaptation to user & context, or minimal response time of the multimodal interface are of prime interest in human-machine interaction, whereas they are not as important in other fields taking advantage of multimodality. Typically, multimodality in biometrics or multimodal data analysis will focus on taking advantage of the multiple modalities to enhance recognition results and reduce false positives and false negatives, whereas multimodal interaction will primarily take into account the capability of multimodal interfaces to offer better usability to users through choice of modalities, plasticity or fusion of modalities depending on context.

2.1.2 Key Features of Multimodal Interaction

The exploitation of characteristics such as composition between modalities, real-time adaptation to user & context or inter-modalities fusion allows for a number of key features compared to standard GUI-based interfaces. Specifically, using a number of different modalities for human-machine interaction allows for:

- Support and accommodation of users' perceptual and communicative capabilities;
- Integration of computational skills of computers in the real world, by offering more natural ways of interaction to humans;
- Enhanced robustness due to combining different partial information sources;
- Flexible personalization based on user and context;
- New functionality involving multi-user and mobile interaction.

Evaluations [87] have indeed shown that users made 36% fewer errors while using a multimodal interface in place of a unimodal interface, in the case of spatial tasks: multimodal commands allowed fewer complex spatial descriptions, thus reducing errors. Furthermore, between 95% and 100% of all users of these evaluations confirmed their preference for the multimodal interface over other types of human-machine interfaces. Moreover, multimodal interfaces, and in particular pen/speech interfaces, have shown greater expressive power and greater potential precision in visual-spatial tasks [87]. Interestingly, although one could expect multimodal interfaces to allow quicker interaction than unimodal interfaces, the same evaluations have shown an increase in efficiency only of 10%, thus can be considered as not statistically significant.

Following on the differences between multimodal interfaces and more classical interfaces, Table 1 lists essential differences between Graphical User Interfaces (GUI) and Multimodal User Interfaces (MUI) [90]. First and foremost, GUIs are accustomed to manage input events in a sequential way, considering all input events as a single input stream. The very nature of multimodal interfaces does not allow to consider speech, gestural, emotional, etc. input events as a single stream of events, but as multiple, differentiated input streams. This seemingly purely technological difference is in fact the basis on which most of the distinctive features of multimodal interaction are built. For example, the fact that multimodal interfaces require parallel processing capabilities, in contrast to standard graphical user interfaces, is a direct consequence of this multiple input stream-based essence.

Table 1. Differences between GUIs and MUIs.

GUI	MUI
Single input stream	Multiple input streams
Deterministic	Continuous, probabilistic
Sequential processing	Parallel processing
Centralized architectures	Distributed & time-sensitive architectures

Another distinctive feature of multimodal interfaces is also a consequence of the input streams on which they are (in most cases) based. Input modalities such as speech or gestures are recognized by means of machine learning algorithms; the results of these recognizers, in their turn, can be interpreted by machine learning-based fusion modules. Thus, multimodal interfaces are able to make extensive use of probabilistic algorithms, once again in contrast to graphical user interfaces, which use only atomic, deterministic user interaction devices, such as the keyboard and the mouse. Finally, managing multiple input streams, their parallel processing and fusion by probabilistic algorithms demands more complex software architectures than the centralized architectures commonly used in WIMP-based interfaces; thus, multimodal interfaces frequently make use of distributed and time-sensitive architectures.

2.1.3 Cognitive Foundations of Multimodal Interaction

The advantages of multimodal interface design are elucidated in the theory of cognitive psychology, as well as human-computer interaction studies, most specifically in cognitive load theory, gestalt

theory, and Baddeley's model of working memory [5][86][91]. Findings in cognitive psychology reveal:

- humans are able to process modalities partially independently and, thus, presenting information with multiple modalities increases human working memory;
- humans tend to reproduce interpersonal interaction patterns during multimodal interaction with a system;
- human performance is improved when interacting multimodally due to the way human perception, communication, and memory function.

For example, when processing both auditory and visual information during speech, a listener is able to extract a higher rate of lexical intelligibility (Grant & Greenberg [42]).

This section subsequently presents works from cognitive science related to multimodal interaction, following cognitive load theory, gestalt theory and Baddeley's model of working memory; the section ends with the description of a framework aimed at human performance prediction.

On the side of multimodal presentation, Mousavi et al. [75] experimented with presenting students content using partly auditory and partly visual modes. The split-attention effect (Sweller et al. [99]) that resulted “suggested that working memory has partially independent processors for handling visual and auditory material.” The authors argued that, if working memory is a primary limitation in learning, then increasing effective working memory by presenting information in a dual-mode form rather than a purely visual one, could expand processing capabilities. The results of Mousavi et al. were confirmed by Tindall-Ford et al. [112], who used more general types of tasks than pure mathematical ones, and by Mayer & Moreno [70] who studied the same effect with multimedia learning material. All this work is in line with the cognitive load theory, which assumes a limited working memory in which all conscious learning and thinking occurs, and an effectively unlimited long-term memory that holds a large number of automated schemas that can be brought into working memory for processing. Oviatt [86] applied these findings to educational interface design in testing a number of different user-centered design principles and strategies, showing that user-interface design that minimizes cognitive load can free up mental resources and improve student performance. One strategy for accomplishing this is designing a multimodal interface for students.

On the side of multimodal input, fewer work has been achieved. In the case of design of map-based pen/voice interfaces, Oviatt et al. [91] demonstrated that Gestalt theoretic principles successfully predicted a number of human behaviors, such as: users consistently followed a specific

multimodal integration pattern (i.e. sequential versus simultaneous), and entrenched further in their pattern during error handling when you might expect them to switch their behavior. Gestalt theory also correctly predicted in this study a dominant number of subjects applying simultaneous integration over sequential integration.

The original short-term memory model of Baddeley & Hitch [6], refined later by Baddeley [5], described short-term or working memory as being composed of three main components: the central executive (which acts as supervisory system and controls the flow of information), the phonological loop, and the visuo-spatial sketchpad, with the latter two dedicated to auditory-verbal and visuo-spatial information processing, respectively. Although these two slave processors are coordinated by a central executive, they function largely independently in terms of lower-level modality processing. This model was derived from experimental findings with dual-task paradigms. Performance of two simultaneous tasks requiring the use of two perceptual domains (i.e. a visual and a verbal task) were observed to be nearly as efficient as performance of individual tasks. In contrast, when a person tries to carry out two tasks simultaneously that use the same perceptual domain, performance is less efficient than when performing the tasks individually. As such, human performance is improved when interacting with two modalities that can be co-processed in separate stores.

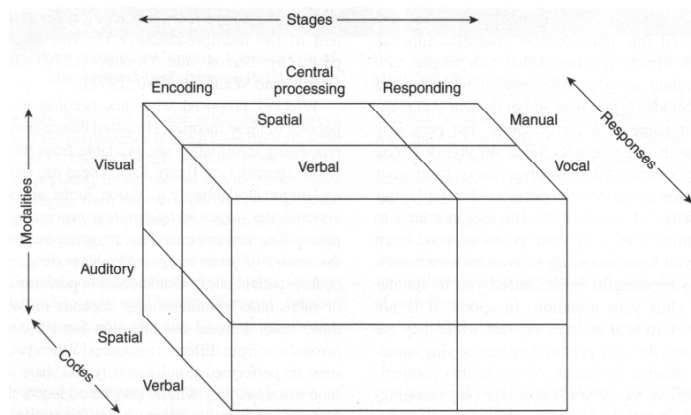


Figure 2. The component structure of Wickens' multiple-resource model of attention (taken from [121]).

Wickens [120][121] also developed a framework, the “multiple resource model”, aimed at performance prediction involving coordination between user input and system output modes for different types of tasks. This model suggests that four different dimensions are to be taken into account when predicting coordination versus interference during human task processing involving different modes. The four dimensions considered are stages (perceptual/cognitive vs. response), sensory modalities (auditory vs. visual), codes (verbal vs. spatial) and channels of visual information

(focal vs. ambient). The multiple resource model allows system designers to predict when tasks can be performed concurrently, interfere with each other, or when increases in the difficulty of one task will result in a loss of performance of another task.

2.1.4 Seminal Works, Findings and Guidelines

Multimodal interfaces emerged approximately 30 years ago within the field of human/computer interaction with Richard Bolt's "Put-That-There" application [10], which was created in 1981 (Figure 3). First multimodal systems sought ways to go beyond the standard interaction mode at this time. Bolt's "Put-that-there" processed spoken commands linked to a pointing gesture using an armrest-mounted touchpad to move and change shapes displayed on a screen in front of the user. Since this seminal work, multimodal interaction practitioners have strived to integrate more modalities, to refine hardware and software components, and to explore limits and capabilities of multimodal interfaces. Historically, the main trend has focused on pointing and speech combined using speech/mouse, speech/pen [23], speech/gesture [76], or speech/gaze tracking [56]. Later multimodal interfaces evolved beyond pointing into richer interaction, allowing users to produce symbolic gestures such as arrows and encircling.

Another direction in multimodal research has been speech/lip movement integration [16][94], driven by cognitive science research in intersensory audio-visual perception. This kind of work has included classification of human lip movement (visemes) and the viseme-phoneme mappings that occur during articulated speech. Such work has contributed improving robustness of speech recognition in noisy environments. For more details about these systems, see [9].

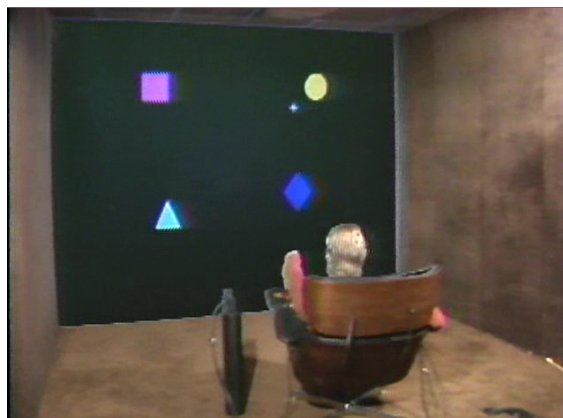


Figure 3. Bolt's "Put-that-there" seminal multimodal interface.

In the course of the last decade, researchers have highlighted particular empirical findings that have guided the design of multimodal interfaces compared to other sorts of human-computer interfaces. Key findings are illustrated in the following “10 myths” shown in Table 2, which exposed common engineering myths regarding how people interact multimodally [88]. Based on empirical findings, Oviatt distilled implications for how more effective multimodal interfaces could be designed.

Table 2. 10 myths of multimodal interaction [88]

Myth #1: If you build a multimodal system, users will interact multimodally.

Myth #2: Speech and pointing is the dominant multimodal integration pattern.

Myth #3: Multimodal input involves simultaneous signals.

Myth #4: Speech is the primary input mode in any multimodal system that includes it.

Myth #5: Multimodal language does not differ linguistically from unimodal language.

Myth #6: Multimodal integration involves redundancy of content between modes.

Myth #7: Individual error-prone recognition technologies combine multimodally to produce even greater unreliability.

Myth #8: All users' multimodal commands are integrated in a uniform way.

Myth #9: Different input modes are capable of transmitting comparable content.

Myth #10: Enhanced efficiency is the main advantage of multimodal systems.

In more recent years, research has also focused on mainstreaming multimodal interfaces. In this trend, Reeves et al. defined the following “guidelines for multimodal user interface design” [98]:

- Multimodal systems should be designed for the broadest range of users and contexts of use, since the availability of multiple modalities supports flexibility. For example, the same user may benefit from speech input in a car, but pen input in a noisy environment.
- Designers should take care to address privacy and security issues when creating multimodal systems: speech, for example, should not be used as a modality to convey private or personal information in public contexts.
- Modalities should be integrated in a manner compatible with user preferences and capabilities, for example, combining complementary audio and visual modes that users can co-process more easily.

- Multimodal systems should be designed to adapt easily to different contexts, user profiles and application needs.
- Error prevention and handling is a major advantage of multimodal interface design, for both user- and system-centered reasons. Specific guidelines include integrating complementary modalities to improve system robustness, and giving users better control over modality selection so they can avoid errors.

Readers will recognize in this list of guidelines a number of challenges mentioned in Section 1.2, in particular adaptation to users & contexts, as well as error prevention and handling. This is no wonder, as those particular challenges are key to the success of multimodal interfaces.

2.2 Multimodal Human Machine Interaction Loop

Human-computer interaction can be represented in the form of a perception-action⁸ cycle, e.g. Norman's human action cycle [83]: humans *perceive* the world with their senses, *interpret* these perceptions, *evaluate* those interpretations, and based on their evaluation of those interpretations, decide on some kind of *goal*; this goal is transformed into an *intention* to act, this intention itself is then translated into a *sequence of actions*, which in turn is *executed*, or applied to the world.

From Norman's cycle, Nigay's Pipe-Lines model (being itself a system-side extension of Norman's cycle) [79], and based on well accepted findings and taxonomies, the following model of the multimodal human-machine interaction loop has been drawn (Figure 4). Basically, it is composed of a variation of Norman's cycle models, human-side and of a variation of Nigay's Pipe-Line model, machine-side. It should also be noted that the computer's cycle is more of a "perception-reaction" than a "perception-action" cycle, as most interactive applications are bound to react to some kind of user input. Thus, in this model, the human and the computer will follow a perception-interpretation-decision-action cycle, each in turn. To get something done, the user will first have to come up with some intention – the *goal*, in Norman's cycle –, that will then be transformed into specific intentions. These intentions will then themselves be transformed into a sequence of actions, with a few particular peculiarities: this sequence of actions will have to be mapped onto a set of modalities recognizers, depending of the multimodal interface running on the side of the computer; incidentally, one of the goals of multimodal interfaces is to make this mapping of human actions into a sequence of computer-intelligible actions as straightforward as possible. Also, it is to be noted that, in parallel

⁸ Or "evaluation-execution" cycle, to quote D.A. Norman.

to the starting intention of the user, the sequence of actions transmitted to the computer (or any kind of receiver, for that matter) will also carry partly- or non-intentional information. One example would be the eye gaze direction. When the first gaze trackers appeared, people thought that gaze direction would be the perfect replacement for the mouse as a pointing device. In fact, researchers quickly noticed that the eye gaze never stops moving, as it is not *under full conscious control*. Thus, eye gaze as a modality should be used as a pointing device (i.e. at an intentional level) only in specific cases⁹. In fact, gaze tracking becomes an interesting modality when you consider it as an indicator of the user's interest, i.e. at an *attentional* level. Another example of non-intentional information carried by modalities can be found in the voice. While the semantic content of a speech act definitely comes from an intention expressed by the user, voice can also be analyzed at a lower, signal level, in order to detect the mood of the user, i.e. considering the *emotional* level of the modality. Another of the potential strengths of multimodal interaction is the ability to take into account those three levels of human communications: intention, attention and emotion.

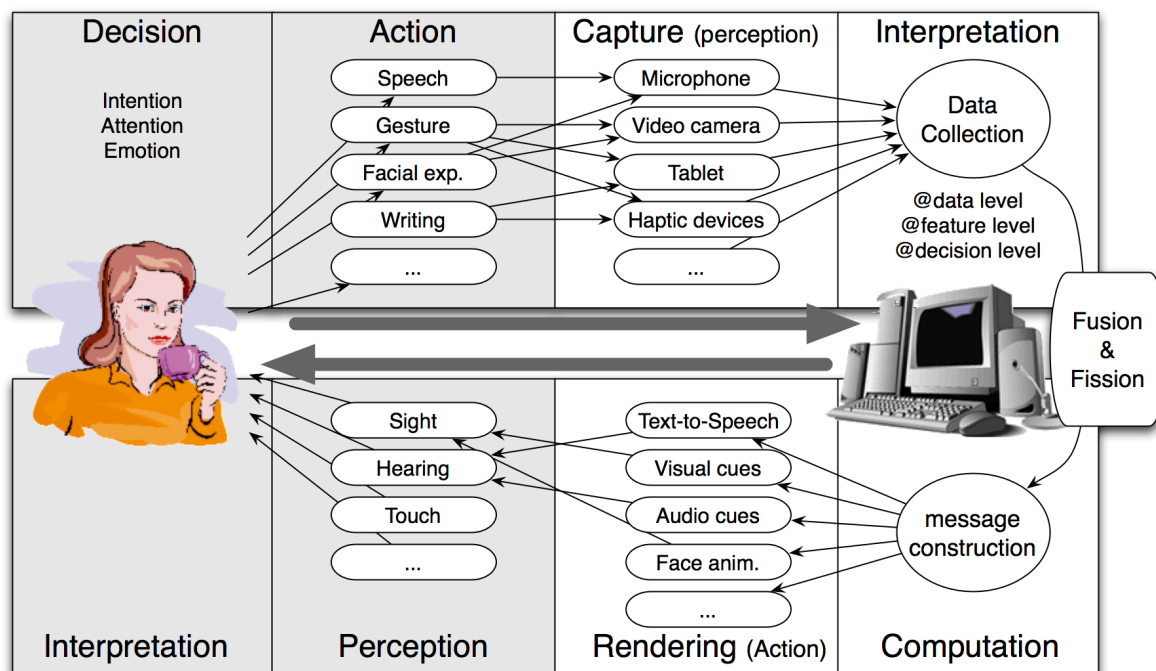


Figure 4. Description of the multimodal human machine loop.

On the side of the computer, the perception-interpretation-decision-action cycle can be thought of as a capture-interpretation-computation-rendering cycle. Perception (or capture) is achieved

⁹ "Specific cases" would here refer for example to interfaces for users with disabilities, or computer games based on gaze tracking, in which the difficulty of keeping or switching focus on a particular subject becomes the core of the game mechanics.

through various sensors and devices, such as microphones, webcams or touch surfaces. Interpretation of captured data takes place at various levels: first, per modality, at a raw, signal level, then, for some modalities, at a higher, semantic level, and finally, all modalities are fused together for the multimodal system to derive a unified interpretation. Fusion of input data will be discussed deeper in chapter 5. We will however note that fusion of input data can be achieved also at different levels: data level, feature level, or decision level [104]. After computation has been accomplished based on the interpretation extracted from the various input data, a sequence of action is decided upon by a fusion engine. This sequence of actions will use different output modalities such as text-to-speech, animated faces, or audio or visual cues. The user will then perceive the multimodal system's output with her five senses.

2.3 Design Spaces and Theoretical Frameworks for Reasoning on Multimodal Interaction

Multimodal interaction, with its particular features, could not re-use most of the theoretical background created for analysis of WIMP-based human-computer interaction. Instead, specific design spaces had to be designed for this goal. This section will thus present one theoretical framework, as well as two design spaces specifically fashioned for analysis of multimodal interaction. The theoretical framework which will be presented is the TYCOON framework, created by Martin [69]. As for the design spaces, they conceptualize the different possible relationships between input and output modalities. First, the CASE model [80] classifies interactive multimodal systems according to two of their main features: concurrent processing and data fusion. On their side, the CARE properties (Coutaz et al. [24]) seek to assess the usability of multimodal interaction, as well as characterize system-centered features.

As will be seen in the next chapters, those design spaces and theoretical frameworks (and in particular, the CARE properties) have been used in this thesis in multiple places. One could however object these theoretical frameworks not being up-to-date with the most recent findings. Nevertheless, a revision of these frameworks fell out of the scope of this thesis, and thus these theoretical tools, which have already shown their efficiency, were applied as is.

2.3.1 The TYCOON Theoretical Framework

Martin proposed in [69] TYCOON, a theoretical framework for studying multimodality. TYCOON (*TYpes and goals of COOperationN between modalities*) is a two-dimension framework (see Figure 5), with five basic types of cooperation between modalities forming the x-axis, and a number of goals defining the second dimension of the framework. The five types of cooperation are of particular interest in the context of this thesis, and are as following:

- *Transfer*, when a chunk of information produced by a modality is used by another modality. Typically, when a mouse click triggers the display of an image.
- *Equivalence*, when a chunk of information may be processed as an alternative, by either of them; e.g. when two or more modalities are available for a given command.
- *Specialisation*, when specific kind of information is always processed by the same modality.
- *Redundancy*, when the same information is processed by a number of modalities, e.g. for improving recognition.
- *Complementarity*, when different chunks of information are processed by each modality, but need to be merged. Typical example here is Bolt’s “Put-That-There”.

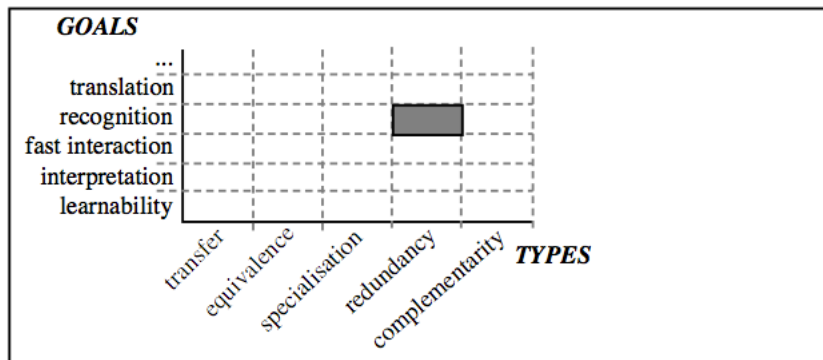


Figure 5. The TYCOON Theoretical Framework for studying multimodality (taken from [69]).

2.3.2 The CASE Design Space

The CASE design space was first proposed by Nigay & Coutaz in 1993 [80] in order to provide multimodal interaction practitioners with a way to characterize multimodal interfaces according to three definite dimensions: type of fusion, use of modalities and levels of abstraction. “Level of abstraction” dimension is used to differentiate between the multiple levels at which a given input device data can be processed. As a classical example, consider speech, which can be processed at a signal (data) level, at a phoneme (feature) level or at a semantic (decision) level, as illustrated in

Figure 6. “Use of modalities” expresses the temporal use of different modalities. Two cases are distinguished: either modalities are used in a sequential manner, one at a time, or simultaneously, i.e. in parallel. Finally, “fusion of modalities” concerns possible combination of different types of data: “independent” means absence of fusion, no coreference between modalities, whereas “combined” implies the need to fuse modalities to get the full meaning of a given multimodal command.

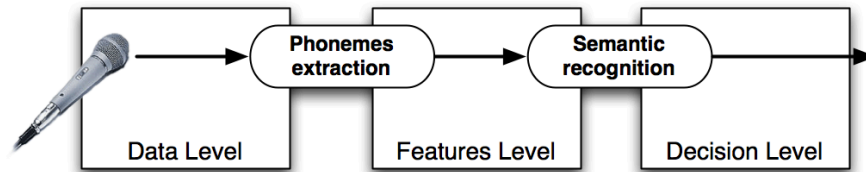


Figure 6. Different levels of abstraction for a same input source.

Figure 7 maps the “use of modalities” and “fusion of modalities” dimensions in order to obtain four different categories characterizing multimodal commands: Concurrent, Alternate, Synergistic and Exclusive. A “concurrent” command thus makes a parallel use of modalities, but does not combine them. An “alternate” command combines modalities, but in a sequential manner. A “synergistic” command combines modalities in a parallel manner (“put-that-there” being a classic case of synergistic command). Finally, an “exclusive” command is composed of sequential and independent modalities.

		USE OF MODALITIES	
		Sequential	Parallel
FUSION OF MODALITIES	Combined	ALTERNATE	SYNERGISTIC
	Independent	EXCLUSIVE	CONCURRENT

Figure 7. The CASE design space (figure based on reference [80]).

Through characterization of every multimodal command available in a given system, and weighing of those same multimodal commands in accordance with their relative use in the system, one can then make use of the CASE design space to classify and compare different multimodal systems based on their commands types. The authors also state that “the CASE design space can be

used in conjunction with the [CASE] classification scheme to study the effect of shifting commands within the design space with regard to the user's expertise or to the task to be performed“.

2.3.3 The CARE Properties

The CASE design space is a useful tool to analyze the features of a multimodal system, but it has the drawback of being too system-centered to be fully used for user-centered design of multimodal system. Thus, Coutaz et al. [24] proposed in 1994 the CARE properties, as a more generic set of properties to be used in usability testing of multimodal systems, as well as characterization of system-centered features, such as devices, languages and tasks (see Figure 8) [82]. The CARE properties can be applied to the design of input as well as output multimodal interfaces. The four CARE properties are Complementarity, Assignment, Redundancy, Equivalence. A formal definition of each of the four properties can be found in [24]. We will give a condensed definition of each property below. Let M be a set of modalities, and s and s' two states of a given multimodal system, with s' following s , then:

- Equivalence: modalities of set M are equivalent, if it is necessary and sufficient to use *any one of the modalities* to reach state s' from s . There is thus an availability of choice between multiple modalities.
- Assignment: modality m is assigned in state s to reach s' , if no other modality is used to reach s' from s . In contrast to equivalence, assignment expresses *the absence of choice*.
- Redundancy: modalities of set M are used redundantly to reach state s' from state s , if they have *the same expressive power* (\sim equivalent) and if all of them are used *within the same temporal window*.
- Complementarity: modalities of set M must be used in a complementary way to reach state s' from state s within a temporal window, *if all of them must be used* to reach s' from s . Thus, no modality taken individually is sufficient to reach the target state. Complementarity can occur sequentially or in parallel.

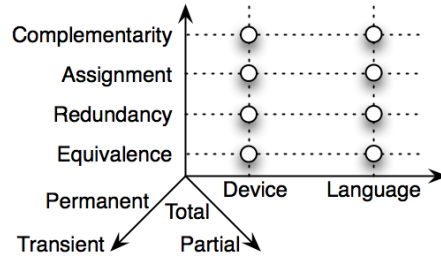


Figure 8. CARE properties as a framework to characterize multifeature user interfaces (figure taken from reference [82]).

In [82], Nigay & Coutaz extend the CARE properties to the two sides of multimodal human-computer interaction, in differentiating between *system CARE properties* and *user CARE properties*. Then among system CARE properties, a further differentiation is achieved between *Language-level* properties and *Device-level* properties.

Although the CARE properties can be applied as much for output than for input [67], they were used mainly for input categorization in the frame of this thesis, as will be seen in the following chapters.

2.4 Conclusion

This chapter presented the conceptual background underlying multimodal interfaces. Multimodality, as a concept of human-machine interaction, was defined. Key features as well as cognitive foundations of multimodal interaction helped further underline the capabilities and promises of this new type of human-machine interaction. Presentation of seminal works, findings and guidelines gave a picture of the evolution of research in multimodal research for the past 30 years, as well as sketch the current challenges of the field. The introduction of a multimodal human-machine interaction loop, derived from Norman's action cycle, contributed to give a clearer picture of the challenges presented by multimodal interaction. The CASE design space and the CARE properties, as tools for reasoning about the design of multimodal systems, help address some of these challenges, and as such, will be put to use in the three axes of this thesis. We will in particular focus on the CARE properties, as their application to system-side as well as user-side characterization of multimodal interaction has been established in [82].

3 Multimodal Interfaces Prototyping Frameworks & Architectures

*"L'architecture, c'est formuler les problèmes
avec clarté."*
Le Corbusier

3.1	Terminology	42
3.2	Related Work.....	43
3.3	Multimodal Interaction Creation Tools: Discussion of Prominent Features Related to Fusion.....	47
3.3.1	Input Management.....	47
3.3.2	Architecture features	48
3.3.3	Dialog Description & Programming	49
3.3.4	Fusion Engine Characteristics	51
3.3.5	Miscellaneous features	51
3.3.6	Final Analysis.....	52
3.4	Multimodal Interfaces: a Study of Architectures	53
3.4.1	Computational Architecture and Key Components.....	53
3.4.2	Java Swing MM extension: Usability.....	55
3.4.3	SCS, the Service Counter System: Expressiveness	57
3.5	The HephaïsTK framework, and its Architecture	60
3.5.1	Initial Requirements for HephaïsTK	61
3.5.2	General Architecture of HephaïsTK.....	62
3.5.3	Usability and Expressiveness: Looking for Balance	68
3.6	Usability, Expressiveness and Prototyping Multimodal Systems: an Outlook	68
3.6.1	Summarization and Observations.....	68
3.6.2	Positioning HephaïsTK	69
3.7	Conclusion.....	69

As pointed out by Oviatt et al. in [90], “multimodal systems still are very new and hard to build. In order for the fledgling multimodal research community to develop high-performance multimodal systems and eventually commercialize them, considerable research will be needed to develop appropriate infrastructure”¹⁰. Indeed, creation of multimodal interfaces imply deep knowledge in a number of state-of-the-art research domains, such as speech or gesture recognition, modalities fusion, or adaptive interfaces. Furthermore, in addition to the novelty of employed technologies, creating multimodal interfaces requires numerous resources due to their potential complexity.

Consequently, the need for tools allowing easy creation of multimodal interfaces has rapidly risen. In the last five years, a number of such tools have appeared, each with specific advantages and drawbacks. Of these tools, one of them is at the core of this thesis. This chapter will thus offer a view of our experiences and investigations, after having built a tool for multimodal interface prototyping with studies on fusion of multimodal input data in mind, all the while building on foundations, findings and theoretical frameworks described in Chapter 2.

After a short section on terminology used in this chapter, the second section of this chapter will give a view on the current state of the art in tools for multimodal interface prototyping. The third section will take the form of a discussion of the different solutions presented in the state of the art, following a number of different axes. The fourth section presents our experiments with various architectures for frameworks dedicated to creation of multimodal interfaces. The fifth section introduces the architecture of HephaisTK, as an illustration of the previous section. The sixth section summarizes the particularities of tools presented in this chapter.

3.1 Terminology

As stated in the introduction, this chapter is heavily focused on tools targeted at creation of multimodal interfaces. Thus, this first section will define a number of terms used in this document. Tools for creation of user interfaces (not only multimodal ones) can be broadly divided in four different categories:

¹⁰ § 19.5.4, p. 452 in the reprinted version of “Human-Computer Interaction in the New Millennium” (ed. J. Carroll), Addison-Wesley Press, Reading, MA, 2001.

- *Toolkits* are typically pieces of software constituted of a number of widgets, e.g. GUI widgets, in which each widget facilitates a specific user-machine interaction. Examples include AWT or Swing for Java, Tk in Tcl, or XForms for the X Window System.
- *Frameworks* are defined as abstractions in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality. Typically, a framework possesses a predefined flow of control, and a default behaviour; moreover, it can be extended.
- *UIDEs* (User Interface Development Environments) are software applications that provide comprehensive facilities to computer programmers for user interface development. Typical examples will have a graphical user interface in which the developer can visually edit the interface.
- *UIMS* (User interface Management Systems) are systems supporting the development and execution of user interfaces, usually on top of windowing systems. UIMS can also designate the models underlying such systems (e.g. the ARCH Model [7]).

Aside from the definition of these four categories of tools for development of user interfaces, two terms need also to be narrowed down in the scope of this chapter: prototyping and development, in the context of tools for the creation of user-machine interfaces. In this context, “prototyping” implies the possibility to do user testing on a non-final version of an interface, either through a simplified iteration, or through, e.g., integrated wizard-of-Oz testing. “Development” refers more frequently to the creation of a complete, finalised version of a given interface, even if the frontier between both can, at times, be blurry.

3.2 Related Work

As soon as research in multimodal interaction began to thrive, the need for tools for rapid prototyping of multimodal interfaces grew accordingly. Thus, a number of such tools have appeared in the past 10 years, with most of them emerging between 2005 and 2010.

As a first case of a tool allowing creation of multimodal interfaces, Krahnstoeve et al. [57] proposed in 2002 a framework using speech and gesture to create a natural interface. The output of their framework was designed to be used on large screen displays enabling multi-user interaction. Fusion was done using a unification-based method. Cohen et al. [23] worked on Quickset, a speech/pen multimodal interface, based on Open Agent Architecture [68], which served as a test bed

for unification-based and hybrid fusion methods. Although Quickset was not per se a dedicated framework, it served to build a number of different applications and evaluation platforms. Quickset served also as a source of inspiration for Sinha et al. [106], who created Crossweaver, with a similar architecture than Quickset. Crossweaver is indeed based on Open Agent Architecture, and is focused on events-based input coming from speech and pen input; Sinha et al. built on this already validated architecture a complete prototyping tool allowing testing and analyzing of multimodal interfaces. Bourguet [14] endeavored in the creation of a multimodal toolkit in which multimodal scenarios could be modeled using finite state machines. This multimodal toolkit consisted in two parts, the multimodal scenario interpreter, called MEngine, and a GUI to build the scenarios, called IMBuilder. In their multimodal system, Flippo et al. [37] also worked on the design of a multimodal framework, geared toward direct integration into a multimodal application. One of the most interesting aspects of their work is the use of a parallelizable application-independent fusion technique. The general framework architecture is based on agents, while the fusion technique itself uses frames. Dragicevic et al. [26] worked on ICON (Input Configurator), an input toolkit that allows interactive applications to achieve a high level of input adaptability. A graphical tool allows developers to connect input among different so-called “devices”. Bouchet et al. [13] proposed a component-based approach to fusion called ICARE thoroughly based on the CARE properties¹¹. These components cover elementary tasks, modality-dependent tasks or generic tasks like fusion. Finally, communication between components is based on events.

It is worth noting that all these systems, although each being notable steps toward complete frameworks for the creation of multimodal interfaces, were more ad-hoc tools for the creation of specific application types or evaluations. None of these projects was ever released to multimodal interaction practitioners at large. Researchers had to wait the year 2007 to see the first broadly available frameworks for creation of multimodal interfaces released. OpenInterface [102][104] is a full-fledged framework for the rapid prototyping of multimodal interfaces, created in the context of a European STREP project¹². OpenInterface is based on a components architecture, which allows the framework to be configured according to the needs of a particular application. Typical components manage input modalities, filter or transform input of said modalities; such components can also be replaced with “dummy” components, allowing for example a wizard-of-oz to inject data into the system [103]. Fusion of modalities are taken care of with help of specific components, which are based on the CARE properties model. Two GUIs are available: the first one, called OIDE (OpenInterface Interaction Development Environment) enables to create pipelines of components in

¹¹ See 2.3.3 for more details on the CARE properties.

¹² Reference: EU IST FP6-35182. See <http://www.oi-project.org/> (accessed 06.08.2010).

order to build applications with the OpenInterface framework; a more recent GUI, called SKEMMI [63], provides a tool that lets developers easily generate and package components to be used by the OpenInterface kernel. A second example of a framework for multimodal interaction is SSI (Smart Sensor Integration) [117]. Like OpenInterface, SSI was created in the context of a European project: the CALLAS project¹³. As the CALLAS project is targeted at emotionally-aware systems, SSI has been created primarily for the development of multimodal online emotion recognition systems. Nevertheless, its structure allows it to be used for the development of multimodal interfaces. Also like OpenInterface, SSI is built on the concept of pipelining a number of components (named “services”), each one dedicated to a particular task. However, SSI adds two interesting notions to this: first, a three-layered architecture, allowing each component to directly tap into the data collected by every provider service of the system; second, a distinction between “pure” stream-fed services and event-triggered services. One can finally note that SSI and OpenInterface both share a similar conceptual architecture, but their different primary goals lead to substantial differences in the fusion schemes they offer: OpenInterface targets pure or combined modalities, and thus bases its fusion modules on the CARE properties, whereas SSI has more interest in situation awareness and video processing components, with emotion-oriented fusion schemes, such as the PAD model [41]. ICO formalism (Interactive Cooperative Objects), based on Petri Nets, have also been used to create multimodal interfaces. In particular, a tool named PetShop [58][100] allows users to define the behavior of multimodal systems through Petri Nets¹⁴. This formal specification technique has already been applied in the field of Air Traffic Control interactive applications, space command and control ground systems, or interactive military or civil cockpits (as a collaboration with ICARE project, cf. [13]). Finally, the most recent manifestation of a framework targeted at creation of multimodal interfaces is Squidy [55]. Squidy¹⁵ has been created as a framework for unifying device drivers, other frameworks and tracking tools into a common library. Emphasis has been put on an efficient and powerful visual interaction design GUI, making use of semantic zoom to adjust the complexity of the user interface. Once again, this framework makes use of components, which are pipelined in the design tool to create interfaces.

On the topic of tools allowing creation of multimodal interfaces, it is also worth mentioning the PhD thesis of Serrano [101], who did a thorough analysis on comparing and assessing tools dedicated to prototyping multimodal interaction following a complete set of criteria, including usability- and expressiveness-related features.

¹³ See <http://www.callas-newmedia.eu/> for more information (accessed 06.08.2010).

¹⁴ See <http://ihcs.irit.fr/petshop/> for more information (accessed 06.08.2010).

¹⁵ See <http://www.squidy-lib.de/> for more information (accessed 06.08.2010).

Table 3. Architecture traits of current multimodal systems.

	Krahnstoeve et al. [57]	Quickset [23]	CrossWeaver [106]	MEngine/IMBuilder [14]	Flippo et al. [37]	ICON [26]	ICARE [13]	OpenInterface [104]	Smart Sensor Integration [117]	Squidy [55]	ICO [58]
Input Management:											
– Stream-based						x	x	x	x	x	x
– Event-driven	x	x	x	x	x						
Architecture Features:											
– Components-based							x	x		x	
– Software agents-based		x	x		x						
– Description with state machines	x			x							x
– Other/specifically tailored architecture						x			x		x
Dialog Description and Programming:											
– Graphical user interface			x	x		x		x		x	x
– Script (XML-based, e.g.)					x						
– API available									x		
– “Hard-coding” necessary	x	x					x				
– Description with state machines	x			x							x
Fusion Engine Characteristics:											
– Frame-based fusion	x	x	x		x		x	x			
– Symbolic-statistical fusion		x									
– Based on CARE properties							x	x			x
Miscellaneous:											
– Easy extensibility to other modalities		x		x	x	x		x	x	x	x
– Pluggability into a given application			x			x		x		x	x
– Reusable modules		x				x	x	x	x	x	x
– Open source/broadly available			x			x		x	x	x	

3.3 Multimodal Interaction Creation Tools: Prominent Features Related to Fusion

This section will now provide a comprehensive comparison between the different interaction creation tools described in section 3.2. In particular, the tools will be compared along five different axes:

- Input management
- Architecture features
- Dialog description & programming
- Fusion engine characteristics
- Other miscellaneous features

These axes were selected because of their tight link to the central topic of this thesis research work, i.e. fusion of multimodal input data. *Input Management* revolves representation means of input data in view of their fusion, *Architecture Features* list the type of architecture chosen to create each tool, *Dialog Description & Programming* describe the way the fusion engine is programmed/scripted, *Fusion Engine Characteristics* are directly attached to the type of fusion algorithms present in the tool. Finally, *Other Miscellaneous Features* are not directly related to fusion management, but help contextualize each tool in its capabilities and goals.

Each of the five following subsections details one of those five different axes. Table 3 gives a summary of the aforementioned discussion.

3.3.1 Multimodal Interaction Creation Tools: Input Management

The careful reader certainly noticed similarities between the tools presented in section 3.2. In particular, some of the recent tools for creation of multimodal interfaces, like for example Squidy or ICOs (Interactive Cooperatives Objects) [58], consider input data coming from modalities as raw input streams, then apply transformations on those streams with help of a pipeline of transformer components. In contrast to those, a number of other tools, such as IMBuilder/MEngine, consider data as individual atoms, each one capable to be the source of some kind of event. The distinction could seem artificial, as every stream of input data is composed of individual events, and, conversely, atomic events coming from a given source form a stream of data. However, this distinction between stream- and event-based management of input data implies a very different representation of said data at the software level, which in turn has deep repercussions on the software architecture of tools

for the creation of multimodal interfaces, and in particular on the fusion process capabilities of the tool. Indeed, fusing streams of data, such as pointer position streams coming from two mice, will definitely use different types of algorithms than a task like the “put-that-there” from Richard Bolt presented in chapter 2.

Obviously, this issue takes its root to the continuous/discrete duality of input data, observed already decades ago by (among others) Buxton [18] or Card/Mackinlay/Robertson [19]. The implications of this duality, as will be seen in the following sections, are nonetheless numerous when designing a framework targeted at prototyping multimodal interfaces, in particular with regard to fusion of input data.

Frameworks targeting creation of multimodal interfaces face another, yet related challenge: as multimodal interfaces have to cope with state-of-the-art input recognition technologies, they should take into account a huge number of different input formats; even worse, still-not-developed input technologies could bring to the table input formats unknown to the tool... Frameworks face then the following choice: either restrict their users to specific classes of “architecture-friendly” input devices, therefore banning all other kinds of devices; or consider input devices and sources as abstract providers of information, and work toward the acceptance of any kind of input data, coming from any previous, current or still-not-invented device or software tool. Needless to say, this last solution should be privileged, but opens a host of software engineering and design challenges.

3.3.2 Multimodal Interaction Creation Tools: Architecture features

Tools targeted at creation of multimodal interfaces have rapidly asked for more complex architecture features than standard WIMP interfaces. In particular, as pointed out in subsection 2.1.2, multimodal interfaces ask for software architectures able to manage multiple concurrent input sources, as well as the multiple tools needed to handle and interpret these concurrent input sources.

Typical software architectures for multimodal interfaces creation tools are based on components, software agents, or centered around state machines, with other tools sporting specifically tailored architectures, with no common features with the rest of the state of the art. The framework created by Krahnstoeve et al., as one of the first occurrences of such tools, was built around state machines, which were used to manage the different stages of the application. Bourguet extended the concept, with the addition of a graphical tool allowing explicit creation of multimodal interaction. Software agents followed soon after, with Quickset as one of the main examples of a complex multimodal application making use of a software agents-based architecture. Crossweaver as

well as the work of Flippo et al. also made extensive use of software agents when developing their tool for creation of multimodal interfaces. On the side of components-based architectures, one of the main examples is ICARE. Once again, the “proof of concept” application was followed by a tool, in this case, even two: OpenInterface and Squidy. For its part, the ICO notation allows an extended and thorough modeling of human-machine interaction, with a tiered architecture. Smart Sensor Integration is in the same case.

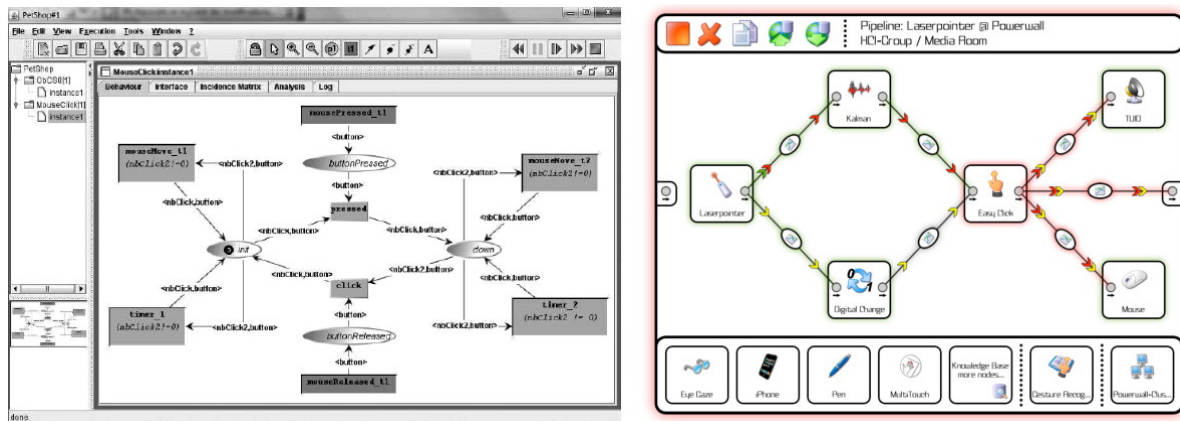


Figure 9. Examples of purely stream-oriented graphical tools: ICO vs. Squidy.

3.3.3 Multimodal Interaction Creation Tools: Dialog Description & Programming

Tools targeted at creation of interfaces (multimodal or not) have to provide some means to access the systems and functions offered. In the case of tools for creation of multimodal interfaces, dialog description and programming means took the following forms: function libraries (APIs), scripting languages, or graphical interfaces. Three of the different systems presented did not explicitly offer some means to access their functions, as they were cases of multimodal systems not completely dedicated to creation of multimodal interfaces, even if they had to some extent the capability to help prototyping of multimodal interfaces. Those are Krahnstoeve et al., Quickset and ICARE. All the others are dedicated tools. frameworks offering specific APIs contain only SSI, the framework developed in the context of the CALLAS project, although a more developed means for accessing the available tools of SSI is certainly in the works. As for scripting languages, a good example is the framework created by Flippo et al. Scripting languages for multimodal interaction will be detailed in section 4.1. Finally, as one could expect, most recent advanced tools offer a graphical user interface to help their user create multimodal interfaces. It is indeed the case of MEngine/IMBuilder, OpenInterface, Squidy and ICO. Differences between these tools mainly come down to the stream-

based or event-driven orientation of their input management. For example, tools heavy on the stream-based side, like ICO, ICON or Squidy, will offer tools in which a number of processing boxes are linked (see Figure 9).

OpenInterface is a particular case, because it offers not only one, but two different graphical editors. SKEMMI offers a collection of eclipse plugins interfacing all the tools and features of the underlying platform. On the other hand, OIDE is a java graphical user interface that enables the user to graphically manipulate and assemble components for a given task. SKEMMI and OIDE are presented on Figure 10.

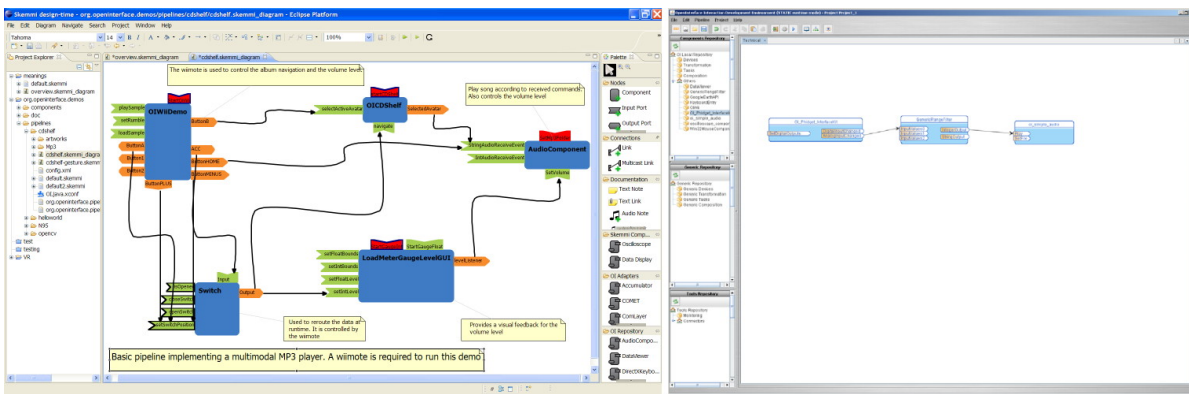


Figure 10. OpenInterface tools: Skemmi and OIDE.

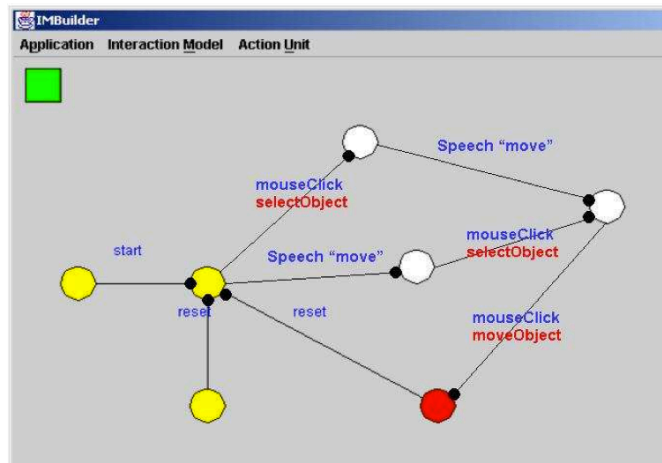


Figure 11. Typical example of an event-driven graphical tool: the IMBuilder tool.

Finally, IMBuilder is a good example of an event-driven graphical editor. Crossweaver is conceptually close to IMBuilder, with more accent on quick prototyping. Although it could look like such a tool is not really different, compared to the tools above, conceptually speaking, the difference is important: indeed, in a tool like IMBuilder, nodes represent states of the multimodal interface, contrary to a tool like Squidy, in which nodes are processing units. In other words, in an event-driven

graphical tool, each node will result in some kind of interaction, whereas in a stream-based graphical tool, the stream has to go through multiple, if not all nodes, before actual interaction will take place. In an events-based description, the resulting graph corresponds to a state machine describing the flow of conversation between the user and machine, whereas, in stream-based applications, the resulting graph is a processing chain.

3.3.4 Multimodal Interaction Creation Tools: Fusion Engine Characteristics

When dealing with multiple input sources, fusion of these input sources is a necessary feature of multimodal interaction creation tools. In fact, fusion of input data can be considered as one of the distinguishing features of multimodal interaction. More details on multimodal fusion will be given in Chapter 5. We will mainly focus on specific features of multimodal fusion, namely the availability of advanced fusion algorithms, such as frame-based or unification-based fusion algorithms, and symbolic-statistical fusion algorithms; furthermore, tools considering CARE properties (see section 2.3.3) as ways to express modalities combinations have been taken into account. In particular, it is the case of ICO, ICARE and OpenInterface, three tools which allow expression of combination of input modalities through the use of the CARE properties.

Tools offering advanced fusion algorithms are less frequent. In fact, on Table 3, only five tools offer frame-based or unification-based fusion of input data: ICARE, OpenInterface, the tool of Krahnstoever et al., of Flippo et al., and Quickset; of those, only Flippo et al. and OpenInterface offered a “real” tool for creation of multimodal interfaces. Finally, it is worth noting only Quickset offers symbolic-statistical-based fusion algorithms; to put this in context, Quickset has been created as a tool targeted at studying multimodal fusion. It is nonetheless worth noting that, to our understanding, nobody built upon the knowledge discovered in the course of the Quickset project since then.

3.3.5 Multimodal Interaction Creation Tools: Miscellaneous Features

This subsection presents a number of particular features one could wish for in a multimodal interaction creation tool. The four selected features are extensibility, pluggability, reusable components, and open-source.

Extensibility focuses on the possibility to easily create newer tools or interfaces from the base one. Logically, most dedicated tools and frameworks offer this possibility. Only preliminary works such as Krahnstoeber et al. don't offer directly extensibility.

Pluggability revolves around the ability to plug into the tool or framework other types of input sources. Newer tools like OpenInterface, ICO or Squidy, have made of this characteristic one of their key features, with dedicated libraries of pluggable components and input recognizers managers.

Reusable components, somewhat linked to pluggability, focuses around offering internal components, like fusion algorithm, stream processing or similar types of components. Evidently, tools built around software components or agents fall into this category, because of their architecture, which allows them to be divided into easily reusable components. Thus, Quickset, ICARE, OpenInterface, SSI and Squidy are such systems. To some extent, ICO also offers reusable components.

Open-source quite simply defines the possibility for other developers to easily access, use and modify a tool for creation of multimodal interfaces. This feature has been added because, until 2005, most (if not all) of the tools dedicated to creation of multimodal interfaces were not available to other developers than their creators. Since then, the situation has hopefully evolved, and the more recent tools, like OpenInterface, SSI and Squidy, are available to a broad audience.

3.3.6 Final Analysis

When observing Table 3, a striking observation one could make is the very deep influence input management has over a lot of other more complex features. It is particularly notable, because stream-based vs event-driven input management is a low-level technical choice, mostly done at the very beginning of the implementation of such tools as the ones discussed, and which could be deeply underestimated: after all, a stream is a series of events, and a series of events form a stream, or so they should.

Careful examination of Table 3 tells us a different story. Independently from the architecture chosen, no "pure" stream-based tool offers advanced fusion algorithms such as frame-based fusion or symbolic-statistical fusion (ICARE and OpenInterface being composite cases, between stream-based and event-driven tools). Conversely, every stream-based tool offers advanced graphical edition tools, whereas event-driven tools struggle to offer a visual representation of their capabilities. We firmly believe those observations go beyond a mere coincidence, and thus, that a seemingly obscure

technical choice has deep influence on the final form a tool for creation of multimodal interfaces will take.

3.4 Multimodal Interfaces: a Study of Architectures

In this section we present work done in the context of this thesis on architectures for multimodal interaction. First, a model of a typical computational architecture for multimodal human-computer interaction is presented. Then, the section presents two of three toolkits and frameworks we implemented and experimented with, in order to further explore particularities of given architectures for multimodal interaction¹⁶. The Java Swing MM extension proposes to extend the Java Swing GUI toolkit, present in the Java language Foundation classes, with other modalities than just the standard WIMP approach, so that programmers do not have to learn a new environment. As will be demonstrated, the drawback of this approach lies in the multimodal expressiveness. The *Service Counter System* uses a finite state machine to enable multimodal fusion and as a way to program interaction scenarios, but at the expense of a lot of “hard-coding”. As for *HephaistTK* framework, it will be presented more deeply in the next section.

The two toolkits and frameworks introduced in this section, as well as *HephaistTK* framework, try to mix multimodal interaction (coming from a number of different recognizers, such as from Phidgets physical widgets [43], Papier-Maché toolkit [54], Sphinx speech recognition [118] or Reactivision [51]) with other forms of input. We present in this section the first two systems, emphasizing on their major characteristics.

3.4.1 Computational Architecture and Key Components

Section 2.2 detailed a theoretical model of the multimodal human-machine interaction loop. It is to be noted that this model borrows much to usability and, as such, is to be considered as a helpful tool for user-centered multimodal interaction design. The current section focuses more on the “machine” side of multimodal interaction and details an architectural model of a canonical multimodal application, depicted in Figure 12.

¹⁶ This section is an extended version of the following publication: “Dumas, B., Lalanne, D., Guinard, D., Koenig, R., and Ingold, R. 2008. Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces. In TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction (Bonn, Germany, February 2008). ACM, pp. 47-54” [28].

Broadly speaking, a typical multimodal application can be separated in four different components: input modalities and their recognizers, output modalities and their respective synthesizers, the so-called *integration committee* and, last but not least, the application logic. Indeed, using multimodality efficiently implies a clear abstraction between the results of the user's input analysis, the processing of said input, answer generation and output modalities selection. As shown in Figure 12, this clear separation is achieved with help of the integration committee, responsible for management of all input and output modalities.

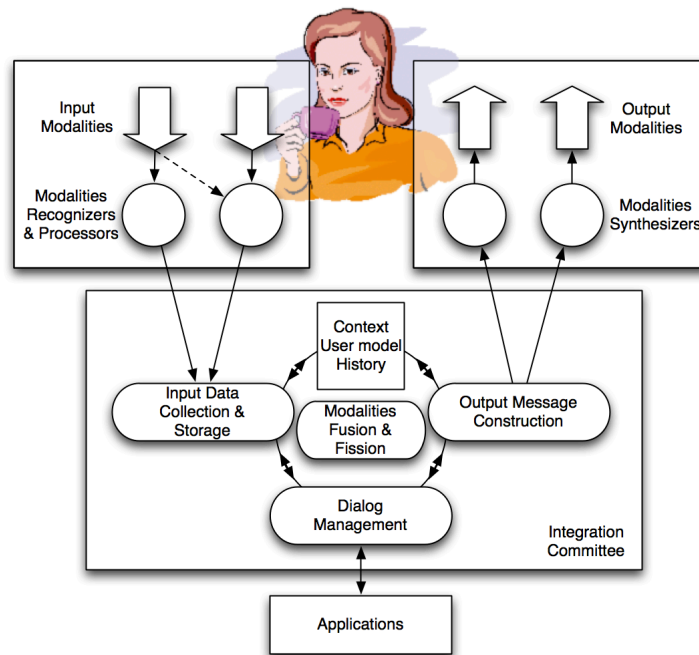


Figure 12. Canonical architecture of a multimodal interactive system.

The integration committee can itself be separated in five different subcomponents. First, modalities input is collected into the *input data collection & storage* module, which is in charge of identifying and storing input data. Central to the whole multimodal integration process, the *Modalities fusion & fission* module manages input data, prepares it for processing by the application logic, for example by applying fission on imbricated data, or fusion on incomplete, yet co-referencing data. Once again, fusion of input modalities will be studied in greater detail in chapter 5. When the fusion & fission engines reach an interpretation, it is passed to the *dialog management* module. The dialog management module is in charge of identifying the dialog state, the transitions to perform, the action to communicate to linked applications, and, based on the feedback of said applications, the message to return to the user through the output modalities fission module; the dialog manager frequently also is tasked with instantiation of the other modules in the integration committee. The *output message construction* role, for its part, is to generate a suitable output corresponding to the

message, through the most adequate modality or combination of modalities, based on the user profile and context of use, once again with help of the *modalities fusion & fission* module. Finally, the integration committee frequently comprises a *context manager* module, in charge of keeping a track of the interaction history and user profiles, as well as detecting the context of use. These information are then fed to the four other modules, so that they can adapt accordingly their interpretations. Actually, a multimodal application is thus able to adapt to a context of use (e.g. car, home, work), type of task (e.g., information search, entertainment) or type of user (e.g. visually impaired, elderly). Fission techniques for output message construction [38] allow a multimodal application to generate a given message in an adequate form according to the context and user profiles. Technically speaking, fission on the output side consists of three tasks:

- Message construction, where the information to be transmitted to the user is created; approaches for content selection and structuring revolve mainly around either schema-based approaches or plan-based approaches [65][74].
- Output channel selection, where interfaces are selected according to context and user profile in order to convey all data effectively in a given situation. Characteristics such as available output modalities, information to be presented, communicative goals of the presenter, user characteristics and task to be performed are forms of knowledge that can be used for output channel selection [2][3].
- Construction of a coherent and synchronized result: when multiple output channels are used, layout and temporal coordination are to be taken into account. Moreover, some systems will produce multimodal and cross-modal referring expressions, which will also have to be coordinated.

Fusion will not be detailed in this part of this thesis, as Chapter 5 is entirely dedicated to that subject.

3.4.2 Java Swing MM extension: Usability

A first attempt at providing an architecture to help creation of tangible and multimodal interfaces was achieved by extending the Java Swing GUI framework. The goal of the Java Swing MM extension was to come up with a simple and rugged solution toward rapid prototyping of multimodal and tangible interfaces, based on a standardized GUI framework.

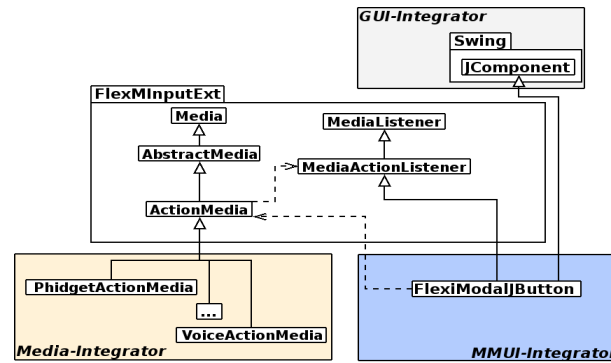


Figure 13. Java Swing MM Extension: MMIButton

The choice of the Java Swing GUI framework as a basis was a default choice at the time we experimented with this approach: in fact, any sufficiently modular GUI framework should be able to be applied the same approach.

The Java Swing MM Extension provides a specific developers' design-pattern which improves any graphical component into a multimodal and/or tangible input component. The strength of this pattern lies in the separation of the different development tasks:

- The GUI-Integrator, which builds new graphical components, does not have to know anything about the actual input recognizers used.
- The Media-Integrator only has to implement the desired media-interface and does not have to know about fusion techniques.
- The MMUI-Integrator only needs to extend the graphical component in order to integrate the desired media listener. The chosen fusion behavior is implemented there.

Any developer can then use the multimodal components as the new components can be handled just as if they were standard GUI components.

The proof of concept for the Input Extension was performed through the improvement of the Swing JButton towards a MMIButton (The JButton is a simple graphical component provided by the Swing-Framework used in Java). The MMIButton component has been equipped with the possibility of being activated by keyboard, mouse, external Phidgets [43] hardware interface components (joystick, physical button, etc.) and voice.

The Output Extension provides another specific design-pattern aiming for the same goal: fission at runtime, separation between the different integration development processes, straightforward usage of the designed output synthesizers.

An architecture such as the one proposed by the Java Swing MM extension allows developers to quickly add multimodal and tangible input to an existing application, or develop such an application from the ground up with a minimum of new knowledge to grasp. Conversely, applications built using the Java Swing MM extension are limited to multimodal or tangible input mapped to specific commands. Regarding synchronicity of modalities, equivalence of modalities is offered right away (see CARE properties in 2.3.3), as long as all modal or tangible commands lead to a specific action; however complementarity can be achieved only in specific cases and with much added work, due to the clear separation between the multiple multimodally enhanced components.

Compared to the “model” architecture presented at the beginning of this section, Java Swing MM Extension exchanges a full-fledged integration committee with a multitude of low-level, low footprint integration committees at the GUI component level. Indeed, in this model, every GUI component becomes a little multimodal architecture, with the obvious drawback of being able to make components exchange information only with extreme difficulties.

3.4.3 SCS, the Service Counter System: Expressiveness

The SCS (*Service Counter System*) is a framework addressing the design and implementation of multimodal user interfaces. It aims to solve the issue in a programmatic way, i.e. it helps the programmer creating a multimodal user interface. Towards this goal it provides:

- An extensible object-oriented abstraction layer (or a framework) of existing input and output libraries.
- A central state machine that can be used to model the interaction flows.

Although designed to serve the design of a broad variety of multimodal user interfaces, the SCS was primarily created to be used for use-cases in which a clear and deterministic interaction flow can be extracted. Thus, it is particularly adapted, but not limited, to semi or fully embedded use-cases such as modeling a rental service, an information desk, an interactive guide, etc.

The SCS is articulated around a central state-machine. Using this automata-generator designed for multimodal interfaces the programmer can describe the human-computer and computer-human interaction flows. In other words, she can design the system’s states and transitions. For instance, using the state-machine, one could enforce the fact that after authentication (i.e. after changing the state), a customer can either rent a DVD (first possible transition) or return it (second possible

transition). As an example, Figure 14 represents the automata designed for the smart librarian use case that we will describe later.

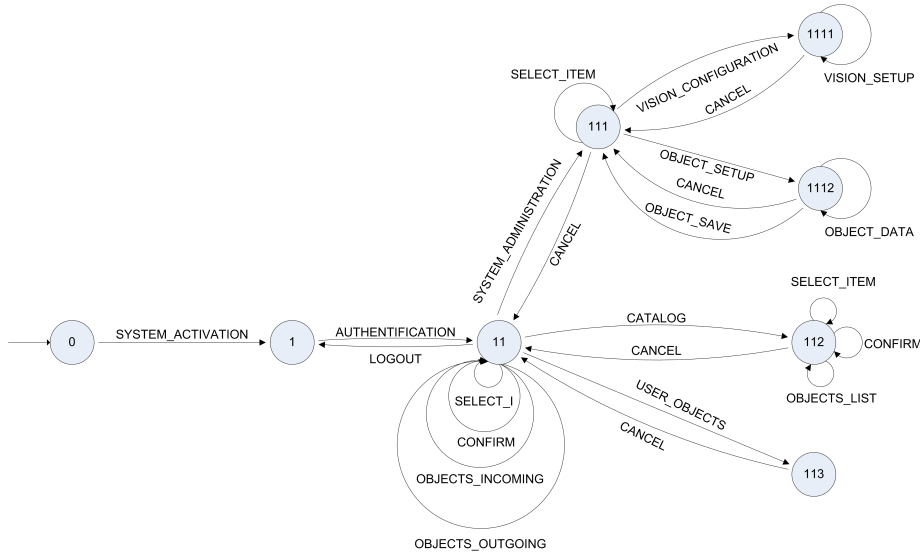


Figure 14. A sample automata modeling the interaction flow.

As mentioned before, the SCS not only provides a way to design the interaction flows of the user interface but also offers to abstract the concerns related to the input and output channels. The abstraction layer is offered by the so called Input and Output drivers. They represent a uniform way for accessing the underlying components and libraries. Each concrete library used as input or output communicates with a SCS driver extending the InputDriver, respectively the Outputdriver class.

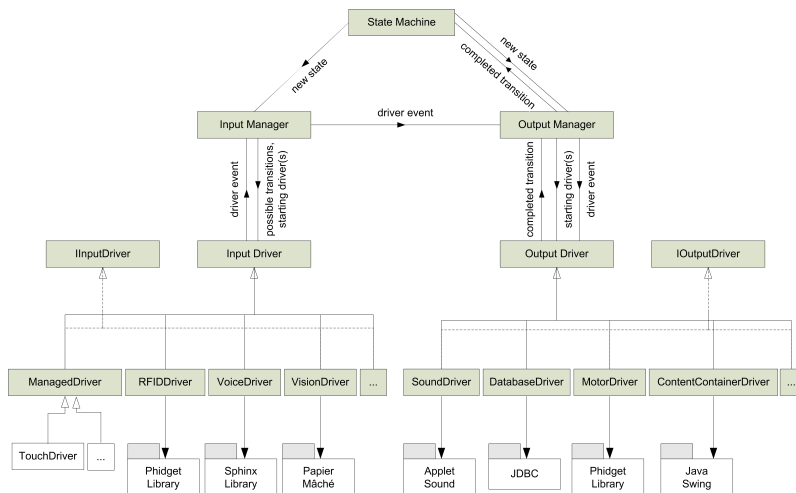


Figure 15. The SCS framework basic architecture.

This way the “phobs” of the Papier-Mâché library [54], the “event handlers” of the Phidgets [43] or the “tags” of the Sphinx speech recognition engine [118] can be accessed in a uniform manner by the programmer.

In terms of synchronization of modalities, the SCS enables equivalence of the input channels (see CARE properties in subsection 2.3.3). Thanks to the state-machine coupled to both the InputDrivers and the observer pattern, the end user can accomplish an action by sequentially choosing one of the 1..n different modalities activated in each state. This way, browsing the options of a menu bar can be operated first using the user’s voice (e.g. with the Sphinx SCS driver), then using a physical joystick (e.g. with the Phidget Joystick SCS driver), and eventually by tracing the user’s arms movements (e.g. with the Papier-Mâché SCS driver).

Furthermore, the SCS offers multimodal fission functionalities. The system constructs a feedback to the user’s actions using various different output channels. As before, this is operated using the OuputDrivers architecture coupled with notification methods provided by the InputDrivers.

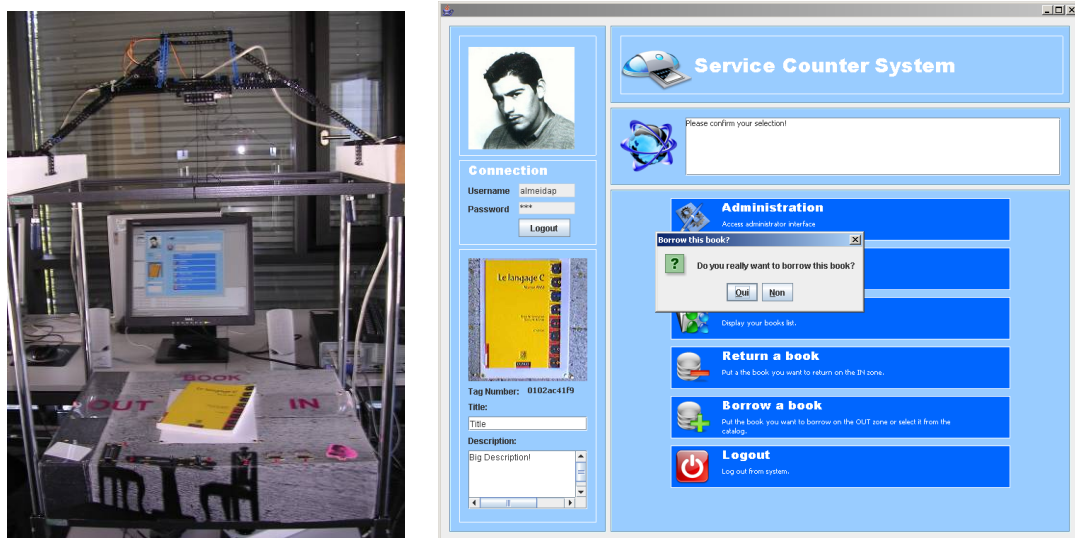


Figure 16. The Smart Librarian use case.

As use case, the Smart Librarian (see Figure 16) project is a concrete multimodal application using the SCS framework. It models a self-service library. The idea behind it is to emphasize multimodality as a help towards more accessible application software and more natural human-computer interaction schemes. This is particularly important since the targeted user group of the Smart Librarian comprises people without exhaustive computer-skills or disabled people.

Thanks to the SCS architecture, the user can use the communication channel he feels to be the most adapted at each step. As an example consider the user wanting to borrow a book. In order to

start this action he can either: talk to the system (saying something similar to “I want to borrow this book” or just “borrow”), use the physical widgets (joystick, button), dispose a book on the “out” zone (monitored by an RFID reader), or make a gesture towards the “out” zone (monitored by the video camera), etc. In order not to confuse the reader with all the activated channels for each state the Smart Librarian uses the fission mechanisms provided by the SCS to turn on signalization LEDs as well as icons on the display and vocal information.

As opposed to the Java Swing MM Extension, the SCS framework favors as much expressiveness as possible: the built-in state machine allows modeling complex human-computer dialog schemes, and the input and output drivers offer extended versatility to the developer using the SCS framework. The price of this expressiveness is a tool asking extended expertise from the developer using it, and the necessity to fully integrate the SCS framework into one’s multimodal or tangible application. In this view, the SCS framework follows closely the architecture presented in subsection 3.4.1, to the point of asking developers to directly integrate their application into its model. In this view, the SCS framework and the Java Swing MM Extension can be seen as two extremes of the usability vs. expressiveness balance.

3.5 The HephaïsTK framework, and its Architecture

This section is dedicated to our third, and most advanced, attempt at devising a framework for the creation of multimodal interfaces ¹⁷. As will be seen along the next chapters, it served as a practical laboratory not only for the study of architectures presented in this chapter, but also for the studies on languages for description of multimodal interfaces in chapter 4, as well as the work on fusion algorithms in chapter 5.

In this section, we present in detail the software architecture of the HephaïsTK framework. First, initial requirements which led to the development of HephaïsTK is enunciated. The general architecture of HephaïsTK is then sketched, with particular focus on software engineering topics related to the overall organization of the framework. Finally, positioning of the framework in terms

¹⁷ This section is an extended and updated version of the following publications: “Dumas, B., Lalanne, D., and Ingold, R. 2009. HephaïsTK: a toolkit for rapid prototyping of multimodal interfaces. In ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 2009). ACM, pp. 231-232” [31] as well as “Dumas, B., Lalanne, D., and Ingold, R. 2008. Démonstration: HephaïsTK, une boîte à outils pour le prototypage d'interfaces multimodales. In IHM '08: Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine (Metz, France, September 2008). ACM, pp. 215-216” [30].

of usability and expressiveness is discussed. It is to be noted though that modeling of human-machine dialog will be detailed later, in particular in section 4.5. In the same vein, fusion & dialog management, as well as discussion on the fusion algorithms in HephaïsTK, are the topics of Chapter 5.

3.5.1 Initial Requirements for HephaïsTK

As HephaïsTK was to serve as a research laboratory for the creation of multimodal interfaces, it had to follow a number of requirements:

- “Allowing rapid creation of multimodal interfaces”: this is the most obvious of the requirements, and the source of the project. As such, the tool had to be expressive enough to describe a wide range of potential multimodal applications, yet usable enough as to not require tremendous knowledge in multimodal interaction.
- “Fusion of different input sources”: one of the goals of this PhD work was the study of different algorithms for fusion of interactive multimodal input. Thus, HephaïsTK had to provide facilities to integrate, switch between, and test/evaluate different fusion algorithms.
- “Fission of output based on context and user profiles”: at the beginning of the project, it made sense to integrate management of fission of output modalities based on context and user profiles. However, we quickly realized that this part was not essential in regard to the other topics on which we decided to focus. As will be explained later, fission management in HephaïsTK is skeletal at best.
- “Open-source”: when first seeking after tools for creating multimodal interfaces in view of the state of the art, one could not help but notice the lack of openly available software¹⁸. Thus, the decision was taken to release the HephaïsTK framework under a GPL-type license when it would be complete.
- “Extensible”: HephaïsTK had to be extensible in a number of specific areas. First of all, as the framework had to be able to use different fusion algorithms, and to swap them, to a certain extent. Second, as a framework allowing the creation of multimodal interfaces, HephaïsTK had to be able to accept input from a number of different input modality recognizers – and even offer the possibility to accept data from completely new types of input modalities. Third,

¹⁸ The situation hopefully improved since then: indeed, recent frameworks such as OpenInterface [104] or Squidy [55] are available under GPL-like licenses.

In HephaisTK, the software agents framework responsible for management of the different agents is JADE¹⁹. JADE was selected over other solutions such as Open Agent Architecture (OAA) because of its architecture not depending on a central facilitator, its neutral definition of agents, and its choice of Java as single programming language, allowing direct multi-platform compatibility, provided that a JVM (Java Virtual Machine) is present; moreover, JADE complies with FIPA specifications²⁰, in particular regarding inter-agents messaging. As for the choice of basing the framework on the Java programming language, it was also considered as a simplification: management of input libraries written in a set of different languages was not a specification of HephaisTK framework²¹.

Figure 17 depicts the global architecture of HephaisTK. Circles represent agents (or, in certain cases, groups of agents), while triangles designate elements external to the framework. Most notably, apart from the client applications (of which we will speak later in this section), the main external components are **input recognizers**. Every one of these input recognizers, in general, focuses on one given modality source, such as speech, gesture, emotions... or one particular device, such as the mouse, or a multi-touch table²². Input recognizers can be any type of software, as long as they are able to transmit their results to code written in the Java language²³. On the side of HephaisTK, **one agent is dispatched for each one of these recognizers**. This software agent monitors the input recognizer, launches it if needs be, manages dialog between the framework and the recognizer (in particular, configuration parameters specified by the user), and has the main task of encapsulating and forwarding all input data generated by the recognizer, as well as metadata and annotations regarding this data. For instance, the agent responsible of a speech recognizer would propagate not only the speech meaning extracted, but also metadata such as a confidence score. Finally, as each of the recognizer agents announces the current change of state of the recognizer it monitors, as well as the managed modality, the framework is always aware of the different recognizers and modalities it has at hand. This feature is also used when parsing a SMUIML scripting file (see Chapter 4 for more details): the framework is able to indicate at parse time whether it will be able to execute the script in

¹⁹ JADE stands for “Java Agent DEvelopment framework”; interested readers can refer to [8], or to the following website: <http://jade.tilab.com/> (accessed 06.08.2010).

²⁰ Cf. <http://www.fipa.org/> (accessed 06.08.2010). FIPA stands for “Foundation for Intelligent Physical Agents”, and is an IEEE Computer Society standards organization.

²¹ As an argument in favor of the choice of Java as sole programming environment, one could advance tools like Jini, which allow the use of libraries written in C/C++ with Java. In fact, a number of drivers, like for example Phidgets [43], make good use of Jini technology.

²² One could object that device drivers are not *per se* “recognizers” in its most literal sense; we will nevertheless use this term to designate every external component producing input data for the toolkit, based on intentional, attentional or emotional cues of the end user.

²³ Typical examples are: code already written in Java (for example the Sphinx speech recognition), native system drivers with a Jini layer (such as Phidgets drivers) or application transmitting their data via an external protocol, such as Reactivision transmitting every data by means of the TUIO protocol [52].

its current state. Finally, every message from the different software agents managing recognizers is sent to the postman agent.

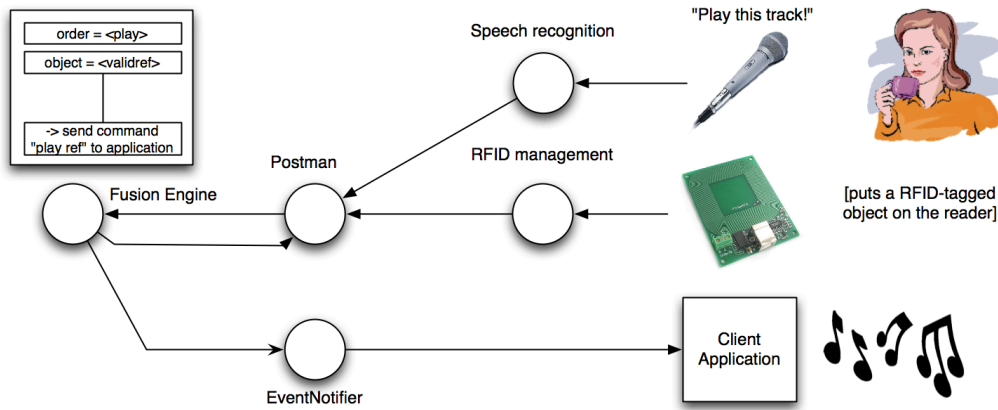


Figure 18. Example illustrating the operation of HephaisTK toolkit in presence of two inputs events.

Figure 18 illustrates this with two recognizers, a speech recognizer connected to a microphone on one hand, and a RFID tag reader on the other hand. Both recognizers are connected to a dedicated software agent. In this example, the user ushers a speech command “play this track”, while approaching the RFID reader with a RFID-tagged object (e.g. representing a music track). Both recognizers will thus generate an input event which will be transmitted to the Postman agent.

Input recognizers agents in HephaisTK have been designed with the possibility in mind to add any type of new recognizer to the framework. Each recognizer-managing agent inherits from a class named *RecognizerControllerAgent*, which offers handles to parameter management, as well as specifications to the overall shape a recognizer-managing agent is expected to provide. In particular, each of this category of software agents is to issue a “controller”, a class inheriting from the *RecognizerController* class. This class in turn asks the developer to provide the piece of information needed by the framework to manage a given recognizer, i.e. the “connector”. Connectors can be seen as, at the same time, abstract representations of a given class of input data, and providers of every facility needed to manage transmission of said input data to the rest of the framework. For example, every speech recognizer tool can make use of a predefined *SpeechRecognizerConnector*, which contains everything needed to transmit speech utterances to the framework, as well as corresponding metadata. As every recognizer agent is asked to specify its corresponding connector, the framework knows at startup which kind of input it has at hand, as well as the specific recognizers which will provide data for a given class of input. By using reflection capabilities of the Java language, connectors are also analyzed in order to know which kind of input data are provided, as well as

variables names. This is in turn used when parsing SMUIML scripts. Finally, it should be mentioned that an agent able to communicate through XML EMMA with any external recognizers is currently in the works.

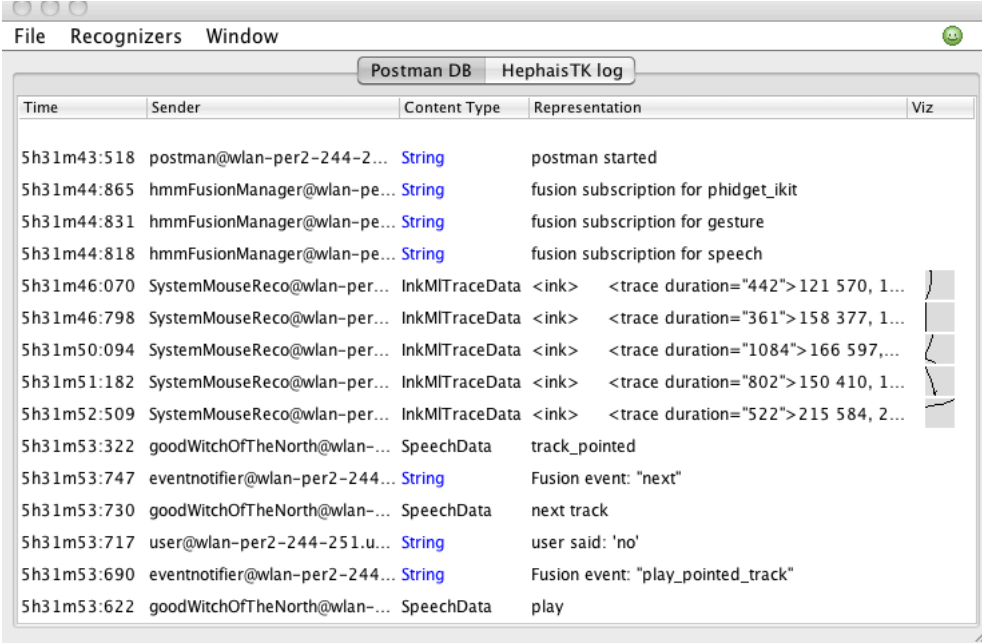
The **postman agent** is a central blackboard collecting data from the recognizers and storing them in a local database. Hence, all data coming from the different sources are standardized in a central place, where other interested agents can dig them at will. Another advantage of a central blackboard architecture is to have one central authority that manages timestamps. The problem of synchronizing different timestamp sources is hence avoided, at the cost of a potential greater shift between the timestamp of the actual event and the recorded one; it can also be seen as a normalization between data coming from sometimes extremely different input sources. It is to be noted that this central agent does not act like a facilitator, though: only agents dealing with recognizers-wise input are able to send data to it. Moreover, the postman agent also offers a mechanism of subscription to other agents: any agent can subscribe to events it is interested in. This mechanism of subscription allows for instance dynamic modifications or complete change of input modalities fusion method. The postman database is based on Apache Derby, and, due to Derby's low footprint and integration with the Java language, can be set up on the fly at the current place of execution of the postman agent, thus allowing seamless deployment of HephaistTK on a given platform.

Typical software agents likely to subscribe to the postal agent are the **FusionManager**, **FissionManager** and **DialogManager** agents. Those agents form HephaistTK's **Integration Committee**, responsible of meaning extraction from data coming from the different input recognizers. This Integration Committee is instantiated for a given client application by a SMUIML document. The SMUIML document contains information about the dialog states (**DialogManager**), the events leading from one state to another (**FusionManager**) and the information communicated to the client application, given the current dialog state and context (**FissionManager**). It is to be noted that the **FissionManager** is only a "dummy" agent, currently forwarding data without any interference; in fact, the lack of a proper management of context and user profiles hinders any deeper fission management in the current state of HephaistTK. Actual implementation of fission is nonetheless a potential future work²⁴.

Getting back to the example depicted in Figure 18, upon receiving the input events from the two recognizers, the postman agent would send them directly to the integration committee, and in

²⁴ See <http://www.phdcomics.com/comics/archive/phd050310s.gif> (last sentence) for more details (accessed 06.08.2010).

particular the FusionManager agent. This agent would then feed these new events to the currently selected fusion algorithm, which, in our case, would trigger a result, in the form of a message to be sent to the client application, indicating that the user wishes to play a given music track.



Time	Sender	Content Type	Representation	Viz
5h31m43:518	postman@wlan-per2-244-2...	String	postman started	
5h31m44:865	hmmFusionManager@wlan-pe...	String	fusion subscription for phidget_ikit	
5h31m44:831	hmmFusionManager@wlan-pe...	String	fusion subscription for gesture	
5h31m44:818	hmmFusionManager@wlan-pe...	String	fusion subscription for speech	
5h31m46:070	SystemMouseReco@wlan-per...	InkMITraceData	<ink> <trace duration="442">121 570, 1...	
5h31m46:798	SystemMouseReco@wlan-per...	InkMITraceData	<ink> <trace duration="361">158 377, 1...	
5h31m50:094	SystemMouseReco@wlan-per...	InkMITraceData	<ink> <trace duration="1084">166 597,...	
5h31m51:182	SystemMouseReco@wlan-per...	InkMITraceData	<ink> <trace duration="802">150 410, 1...	
5h31m52:509	SystemMouseReco@wlan-per...	InkMITraceData	<ink> <trace duration="522">215 584, 2...	
5h31m53:322	goodWitchOfTheNorth@wlan-...	SpeechData	track_pointed	
5h31m53:747	eventnotifier@wlan-per2-244...	String	Fusion event: "next"	
5h31m53:730	goodWitchOfTheNorth@wlan-...	SpeechData	next track	
5h31m53:717	user@wlan-per2-244-251.u...	String	user said: 'no'	
5h31m53:690	eventnotifier@wlan-per2-244...	String	Fusion event: "play_pointed_track"	
5h31m53:622	goodWitchOfTheNorth@wlan-...	SpeechData	play	

Figure 19. Typical startup log in HephaisTK.

The Integration Committee sends all fusion results to an agent named **EventNotifier**, whose task is solely to let the client application interface easily with the HephaisTK framework. The client application needs to implement a set of simple Java listeners in order to receive extracted information from the framework, which is a common implementation scheme in the Java language for GUI development. The EventNotifier is used by the client application to communicate with the framework and provide higher-level information, which could be used by the Integration Committee. HephaisTK hence sees the client application as its client, but also as another input source, and consequently the EventNotifier takes also the role of a recognition agent. Communication with the client application is achieved through a set of messages. Those messages are predefined in the SMUIML script provided by the client application developer. To each message can be attached a set of variables, allowing for example to transfer to the client application the content extracted by a given input recognizer. The messages are then encapsulated in a MultimodalInputFusedEvent, and sent up the chain (Figure 20). In our simplified example of Figure 18, the message indicating that the user wishes to listen to a given music track would be first dispatched to the EventNotifier agent, which would in turn send it to the client application. The client application, now knowing the intent of the user as well as the identity of the track to be played, would only have to comply.

dialog. This downside is still considered acceptable, as correcting it would mean a far stronger bond between HephaïsTK and client applications, and therefore less liberty for developers.

3.5.3 Usability and Expressiveness: Looking for Balance

This framework had to find a balance between usability and expressiveness for the user interfaces developer wishing to use it; in this regard, HephaïsTK has been created as able to plug itself into an existing or new application without heavy modifications; on the other side, programming is achieved by means of an XML-based file able to describe rich interactions with the possibility to design a GUI tool based on this XML language. This balance does not come for free, however: first, as the developer's application and the framework are separated, the developer has to be careful that the framework and her application are in a same "state"; also, the framework is a bulky software, maybe too much bulky for simple use cases not needing advanced multimodal fusion algorithms.

3.6 Usability, Expressiveness and Prototyping Multimodal Systems: an Outlook

The previous section detailed three experiments made to explore different types of architectures for such systems: Java Swing Multimodal Extension, Service Counter System and the HephaïsTK framework. This section will now summarize the different observations reached in the course of this chapter in a recapitulative table, and comment on those observations.

3.6.1 Summarization and Observations

Table 4 is based on Table 3, with the addition of the three frameworks presented in section 3.5. Listed on top of the table are all the state-of-the-art systems mentioned in section 3.2, as well as our three experimental systems: Java Swing Multimodal Extension, Service Counter System and the HephaïsTK framework. The criteria considered for this recapitulation are the same as the ones presented in section 3.3.

A few interesting facts can be derived from Table 4. Most of the frameworks present capabilities for easy extensibility to other modalities; in fact, some tools like OpenInterface or Squidy even offer integrated online libraries of components, of which most are dedicated to

management of input modalities. On the topic of components, one can directly see the relevance of using components-based or software agents-based architectures, only by recognizing that all systems based on such architectures allow for extended reusability of components. Also, systems using state machines for dialog description all make use of event-driven input management. The same can be said about systems using fusion algorithms like frame-based or symbolic-statistical fusion: those systems all need event-driven input management. Globally, our three proposed architectures confirm most of the observations made in Section 3.3. On a final note, it is interesting to notice that open-source, broadly available multimodal prototyping systems have appeared only recently, between 2006 and 2008.

3.6.2 Positioning HephaïsTK

In regard to the state of the art, HephaïsTK takes its place among the recent influx of tools allowing creation of multimodal interfaces. As was detailed in this chapter, HephaïsTK was also a test platform, together with Java Swing MM Extension and Service Counter System, for the development of our views on usability vs. expressiveness. As will be further explained in the two following chapters, HephaïsTK was also the test platform for our work on multimodal dialog description, as well as for our study of fusion algorithms.

In consequence, Table 4 reflects the adequacy of HephaïsTK with regard to these different themes. But what is less reflected, is the “laboratory” orientation of HephaïsTK, in contrast to more accessible tools like OpenInterface or Squidy. The simple fact that HephaïsTK does not currently sport a Graphical User Interface for the description of multimodal human-machine dialog is enough to restrict its potential domain of application to specific pieces of software needing, for example, advanced fusion algorithms such as the ones which will be presented in Chapter 5.

3.7 Conclusion

This chapter, dedicated to multimodal interface prototyping tools and frameworks, and in particular their architectures in relation to fusion of multimodal input data, presented our experimentations and observations with a set of different architectures. First, a selection of relevant tools and applications in the domain of multimodal interaction was presented, and analyzed according to five different axes: input management, architecture features, dialog description & programming, fusion engine characteristics, as well as a set of miscellaneous features. An analysis of the state of the art according

to these five different axes showed the deep influence stream-based vs. event-driven input management has on the development of multimodal interfaces creation tools, in particular on fusion of input data management.

Following this analysis, three different architectures for creation of multimodal interfaces were presented. The first one, named “Java Swing MM Extension”, targets quick prototyping, as it adds a layer to an already broadly available GUI library for the Java programming language. The second architecture, centered around a state machine to describe multimodal human-machine interaction, was seen as offering much more expressiveness to the developer, but at the cost of usability. Finally, the HephaïsTK framework was presented, and its architecture detailed, as a full-fledged solution seeking balance between usability and expressiveness. In regard to the state of the art, HephaïsTK was positioned as a “laboratory” framework dedicated to research on description of dialog and fusion of input data.

Table 4. Architecture traits of current multimodal systems, with works presented in this thesis.

	Krahnstoeve et al. [57]	Quickset [23]	CrossWeaver [106]	MEngine/IMBuilder [14]	Flippo et al. [37]	ICON [26]	ICARE [13]	OpenInterface [104]	Smart Sensor Integration [117]	Squidy [55]	ICO [58]	Java Swing MM Extension (cf. 3.4.2)	Service Counter System (cf. 3.4.3)	HephaisTK (cf. 3.5)
Input Management:														
– Stream-based						x	x	x	x	x	x	x		
– Event-driven	x	x	x	x	x								x	x
Architecture Features:														
– Components-based							x	x		x				
– Software agents-based		x	x		x									x
– Description with state machines	x			x							x		x	x
– specifically tailored architecture						x			x		x	x		
Dialog Description and Programming:														
– Graphical user interface			x	x		x		x		x	x			
– Script (XML-based, e.g.)					x									x
– API available									x			x		
– “Hard-coding” necessary	x	x					x						x	
– Description with state machines	x			x							x		x	x
Fusion Engine Characteristics:														
– Frame-based fusion	x	x	x		x		x	x						x
– Symbolic-statistical fusion		x												x
– Based on CARE properties							x	x			x			x
Miscellaneous:														
– Easy extensibility to modalities		x		x	x	x		x	x	x	x	x	x	x
– Pluggability into a given application			x			x		x		x	x	x		x
– Reusable modules		x				x	x	x	x	x	x			x
– Open source/broadly available			x			x		x	x	x			x	x

4 Multimodal Interaction Modeling

“All models are false but some models are useful.”

George E. P. Box

4.1	Related Work.....	74
4.2	Spectrum of Multimodal Dialog Description Languages	78
4.3	Multimodal Dialog Description: the Synchronisation Problem	80
4.4	Guidelines For Languages for Describing Multimodal Dialog.....	81
4.5	The SMUIML Description Language	83
4.5.1	Structure of the Language	84
4.5.2	Recognizers	86
4.5.3	Triggers.....	87
4.5.4	Actions.....	89
4.5.5	Dialog	90
4.5.6	Language Interpretation.....	91
4.6	Positioning of SMUIML	92
4.7	Conclusion.....	93

Chapter 3 presented HephaïsTK, our solution for a tool dedicated to the creation of multimodal interfaces. As any such tools, a way to describe the human-machine dialog had to be found. In the case of HephaïsTK, as the tool had to be able to describe at a high level the human-machine dialog, the choice was made to enable multimodal human-machine dialog description by means of a XML-based scripting language, specifically created for use with the framework, but generic enough to be

used in another context, e.g. to script other tools. This language, called SMUIML, as well as the studies and observations made toward its inception, are the topic of this chapter²⁶.

The chapter will begin with a review of the state of the art in languages for the creation of multimodal interfaces. These different languages will be compared following a number of characteristics. Then, the spectrum of typical users and uses of multimodal creation scripting languages will be explored. The third section of this chapter will focus on the problem of synchronization in multimodal interfaces, as well as the means to describe it at a high level. The fourth section will describe the SMUIML language itself, and will be followed by a fifth section describing a number of guidelines for the design of languages targeted at the description of multimodal human-machine interaction. A final section about the positioning of the language concludes this chapter.

4.1 Related Work

Interesting attempts at creating a full-fledged language for the description of human-machine multimodal interaction have come up in the past few years. Some of the approaches presented below revolve around the concept of a “multimodal web”, enforced by the World Wide Web Consortium (W3C) Multimodal Interaction Activity and its proposed multimodal architecture²⁷. This theoretical framework describes major components involved in multimodal interaction, as well as potential or existent markup languages used to relate those different components. Many elements described in this framework are of practical interest for multimodal HCI practitioners, such as the W3C EMMA markup language, or modality-focused languages such as VoiceXML, EmotionML or InkML. The framework of the W3C inspired Katsurada et al. [53] for their work on the XISL XML language. XISL focuses on synchronization of multimodal input and output, as well as dialog flow and transition. Another approach of the problem is the one of Araki et al. [4], who propose MIML (Multimodal Interaction Markup Language). One of the key characteristics of this language is its three-layered description of interaction, focusing on interaction, tasks and platform. Ladry et al. [59] use the ICO notation for the description of multimodal interaction. This approach is closely binded to

²⁶ This chapter is a reworked and extended version of the following publication: Dumas, B., Lalanne, D., and Ingold, R. 2010. Description languages for multimodal interaction: a set of guidelines and its illustration with SMUIML. In *Journal on Multimodal User Interfaces: "Special Issue on The Challenges of Engineering Multimodal Interaction"*, vol. 3, no. 3, 2010. pp. 237-247 [32].

²⁷ See <http://www.w3.org/TR/mmi-framework/> (accessed 06.08.2010) for more information.

a visual tool allowing edition and simulation of interactive systems²⁸, while being able to monitor at a low level a systems operation. Stanciulescu et al. [109] followed a transformational approach for developing multimodal web user interfaces based on UsiXML²⁹, also in the steps of the W3C. Four steps are achieved to go from a generic model to the final user interface. Thus, one of the main features of their work is a strong independence to the actual input and output available channels. This transformational approach is also used in Teresa XML (see Paterno et al. [93]). Finally, at a higher level of modeling, NiMMiT (see De Boeck et al. [25]) is a graphical notation associated to a language used for expressing and evaluating multimodal user interaction.

Table 5. State of the art multimodal interaction description languages and their features.

	Abstraction layers	Events mgmt.	Time sync.	Plasticity	Web-oriented	Error handling	Data modeling
EMMA							X
XISL		X	X		X		
ICO		X	X			X	X
UsiXML	X	X		X	X		
TeresaXML	X	X		X			
MIML	X				X		
NiMMiT	X	X	X				

All these approaches seem, at first glance, rather different one from the other. Some features are however common between most of them. Table 5 lists a number of features that could be extracted between the different approaches, as well as some more specific features, related to current research in multimodal interaction. EMMA is included in the table, even if this language targets primarily data transfer between entities of a given multimodal system; in this regard, EMMA perfectly addresses input and output data source representation, as can be seen in Figure 21; in fact, this is the only language to fully address this. XISL is a language targeted at web interaction, and offering a SMIL-like language for multimodal interaction, as illustrated by Figure 22; thus, it provides control over time synchronicity (e.g. with parallel or sequential playing), at least on the output side. The ICO notation targets safety-critical applications, with simulation capabilities; it has events description capabilities but lacks layers of abstraction. On the contrary, UsiXML and TeresaXML were based on specific layers of abstraction, namely AUI (abstract user interface), CUI (concrete user interface) and FUI (final user interface), with XSLT transformations managing

²⁸ PetShop, presented in Section 3.2.

²⁹ See <http://usixml.developpement.defimedia.be/en/home.html?IDC=6> (accessed 06.08.2010) for more information.

crossing from one layer to another. UsiXML as well as TeresaXML also takes into account plasticity (i.e. the capability of a human-machine interface to adapt to both the system's physical characteristics and to the environment while preserving usability). Furthermore, TeresaXML offered extensive events management capabilities. MIML, targeted at multimodal web interaction, also offers layers of abstraction: it is composed of three different languages, managing user/machine interaction, events description, and input/output representation, respectively. Finally, NiMMiT also takes into account separate layers and events management capabilities, as shown in Figure 23.

```
<emma:one-of id="r1" emma:medium="acoustic" emma:mode="voice">
  <emma:interpretation id="int1">
    <origin>Boston</origin>
    <destination>Denver</destination>
    <date>03112003</date>
  </emma:interpretation>
  <emma:interpretation id="int2">
    <origin>Austin</origin>
    <destination>Denver</destination>
    <date>03112003</date>
  </emma:interpretation>
</emma:one-of>
```

Figure 21. An extract of a XML EMMA script, illustrating how EMMA represents data.

```
<operation comb="alt">
  <input type="touch" event="click"
    match="[item:=@id]"/>
  <input type="speech" event="recognize"
    match="./grammar.txt#items" return="item"/>
</operation>
<action>
  <output type="tts" event="speech">
    <![CDATA[ <param name="text">
      You ordered <value expr="item">. </param> ]]>
  </output>
</action>
```

Figure 22. An extract of a XML XISL script, illustrating the way XISL represents human-machine dialog.

A detailed analysis of Table 5 reveals that description languages for multimodal human-machine interaction either are complete on a input/output data stream management point of view, but lack abstraction and expression power, or focus on higher levels of human-machine interaction description, thus losing control over data, or even events. For example, ICO, which addresses multimodal interaction at a low-level, is thus forced to have precise data modeling and error handling; on the contrary, a language such as UsiXML offers higher-level abstraction, with fewer control over the lower layers. It is also to be noted that particular features are incompletely addressed by state of the art languages. Control over time synchronicity has been addressed on the output side of multimodal interaction, but those languages still lack capability to manage complementary, assigned, redundant or equivalent (see CARE properties in subsection 2.3.3) input events. As for error handling, most languages lack ways to express graceful recovery from recognition errors, data fusion mistakes, or system errors. Most languages also lack context and user model modeling, and furthermore plasticity (user and context adaptability) control, with the exception of UsiXML.

```
<?xml version="1.0" encoding="utf-8" ?>
<InteractionTechnique name="TouchSelection" ...>
  <Labels>
    <Label name="highlighted" type="object"/>
    ...
  </Labels>
  <Ports>
    <OPort name="selected" type="object" label="selected"/>
  </Ports>
  <TaskChains>
    <TaskChain id="0" name="CollisionDetect">
      <Tasks>
        <Task id="0" name="UHObj" type="predefined">
          <Ports>
            <Iport name="objects" type="object" ...
          </Ports>
        </Task>
        ...
      </Tasks>
    </TaskChain>
    ...
  </TaskChains>
</InteractionTechnique>
```

Figure 23. An extract of a XML NiMMiT script, illustrating how NiMMiT handles abstraction layers.

Most of the languages described above focus on the "multimodal web", and thus assume interpretation by browser plugins, or even future versions of the browsers themselves. Description languages for multimodal interaction can however be used to be interpreted by dedicated tools for the creation of standalone multimodal applications. As can be seen in Chapter 3, apart from HephaïstosTK and the SMUIML scripting language, of which we will talk in the following sections, only Flippo et al. [37] used a scripting language to program the behavior of their dialogue controller.

4.2 Spectrum of Multimodal Dialog Description Languages

An interesting question to ponder is: who is the user of description languages? The answer is not as obvious as it could appear. People who would want to design multimodal interaction with such languages could be either engineers creating multimodal systems based on programming tools, designers using higher-level tools for creating multimodal interfaces without having to delve too much into code, or even advanced users wishing to customize interaction. These different users will not have the same approach to multimodal interaction description languages. Through various workshops, informal discussions with colleagues and students, and a study of the current state-of-the-art, we envisioned three types of approaches for a description language: a highly formal language approach, perfectly fit for configuring a tool, a loosely formal language approach, good at communicating the details of an application, and a "middle" approach, focused on modeling. Along those three purposes, a fourth purpose for formal languages can be added: learning tool.

The highly formal language approach, targeted at configuration of tools, lets aside readability in order to be as compatible as possible with the inner model of the human-machine dialog of a given tool; EMMA is a typical example of such a language, purely machine-oriented. The loosely formal approach aims at providing a frame of discussion in the shape of a common model; such a language is not necessarily to be interpreted, and focuses on readability. VoiceXML is an interesting example, in that it completely screens all platform-specific details, only to leave developers with the general user interaction code. The mid-level approach, focused on modeling, is one looking for a tool able to configure a platform, while still being able to be human-readable; UsiXML comes to mind as such a language. The "learning" purpose is an offspring of the "communication" purpose, with the goal of framing and directing users towards a specific model. In summary, formal languages can be used for the following purposes:

- configuration

- communication
- modeling
- learning

Hence, formal languages can help configure a multimodal system, thus working as scripting or programming languages; they can be used as communication tools to help exchange and structure ideas about a multimodal systems; formal languages with a thoroughly thought structure can encourage careful modeling of a multimodal application; finally, as potential communication and modeling tools, they can be interesting learning means to handle multimodal interaction. Thus, every description language is expected to find a balanced place on an axis running from highly formal to loosely formal, from configuration to communication (see Figure 24).

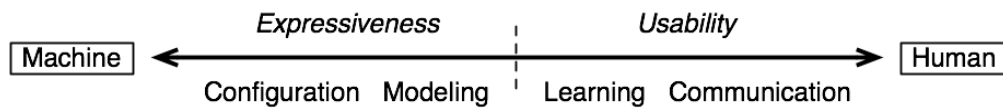


Figure 24. Different purposes for a multimodal user interaction description language.

As was already the case with architectures for multimodal interaction, formal languages for description of multimodal interaction can be approached from two different directions: either from expressiveness, or from usability (see De Boeck et al. [25]). Expressiveness covers technical features such as extensibility, completeness, reusability, or temporal aspects consideration; usability covers more human features such as programmability or readability. Any formal language will have to find its place between those two general requirements. Some languages will for example tend more toward expressiveness, letting all edition be done through a dedicated GUI (visual programming tool); others will focus on usability. In case of a visual programming tool, the usability of the description language is not that important, since it is hidden to users and to be read by the framework using the description. An interesting approach is to seek balance between usability and expressiveness: that is, a language able to configure a multimodal system, with high level modeling, and readable enough to be used as a learning tool, or even a communication tool. To achieve those objectives, such a language will need to have a number of clearly separated and specifically dedicated parts, closely tied together (see Windgrave [122]). We introduce three different levels that will be used in the rest of this chapter: a human-machine dialog-level, an input/output level, and a middle event-level in order to create a link between the human-centered part and the machine-centered part (see Figure 25; inspiration for this model can be traced to the ARCH Model [7]).

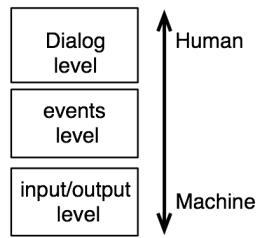


Figure 25. Different levels for a multimodal user interaction description language.

4.3 Multimodal Dialog Description: the Synchronisation Problem

The CASE and CARE models already mentioned the delicate problem of synchronization in the context of multimodal human machine interaction.

CARE properties, for instance, consider the four properties Complementarity – Assignment – Redundancy – Equivalence. Aside from the “assignment” property, the three other properties all take advantage of two or more modalities, and describe inter-modalities synchronizations schemes – or their lack thereof. Aside from the work of Nigay & Coutaz [82][24][80], a number of other authors also took interest into synchronization and concurrency issues between modalities [64][69]. This interest is easily explained by the nature of multimodal interfaces, and their assumed goal of crossing the gap between humans and machines. Using and managing concurrent modalities is a natural feature of human beings, as seen in subsection 2.1.3, and if this gap is to be crossed, applications making use of numerous modalities will have to be able to manage concurrently those modalities, too.

Concurrency on the software side is one part of the problem, and has to be handled at different levels. On the architecture level, multi-threading-based frameworks are necessary to help the interface manage different input modalities; classical architectures toward this goal include software agents-based or components-based architectures (see 3.6 for more details). Advanced fusion schemes are also necessary, for multimodal interfaces to be able to understand human input. Chapter 5 will deal more deeply with such fusion schemes.

On the description language side, things are a bit different. First, description language are on the frontline on bridging the multimodal gap between humans and machines. Indeed, they are expected to offer a machine-friendly description of the way humans will behave in front of a given application. Most evidently, concurrency and synchronicity aspects have an important role to play in

describing human behavior: by the very nature of multimodal interaction, input events have the potential to happen in a concurrent manner, and the time relationship between different events will have an influence on their interpretation. Thus, a description language targeted at multimodal human-machine dialog modelling will have to take concurrency and time synchronicity into account. In the work presented in this chapter, one of the driving principle was to be able to describe synchronicity issues in a simple way, taking as example the way the SMIL³⁰ language handles multimedia synchronous representation.

4.4 Guidelines For Languages for Describing Multimodal Dialog

Sire and Chatty [107] describe what one should want from a multimodal user interfaces programming language. The requirements they express in their article include features such as modality agnosticity, extensible event definition mechanisms or reusable components. From their proposal, the analysis of the state of the art languages referenced in Section 4.1 and Table 5, and open challenges described in section 1.2, the following guidelines for a multimodal description language have been derived. These guidelines are to be seen as a “checklist” of potential features a given multimodal interaction description language can provide. By no means should every language follow all of them. Guidelines should be used as design tools, or as language analysis criterias.

- G1 – *Abstraction levels*: different abstraction levels are advised, as multimodal interaction description can be huge: for example, a description language should separate description of the events and description of the human-machine dialog. Also, reusable parts or structures can greatly help programmability.
- G2 – *Modeling the human-machine dialog*: there should be some way to model the human-machine dialog, be it with a state machine, with an imperative approach with control structures, a declarative approach, or another approach.
- G3 – *Adaptability to context and user (input and output)*: as multimodal interfaces often offer equivalence between modalities, adaptability to context and user (also called plasticity) should be taken into account by a language dedicated to describing multimodal interaction. It is worth noting that adaptability can be considered from an input and an output point of view. On the input side, adaptability would focus on using user information and context to help recognition

³⁰ See <http://www.w3.org/TR/SMIL/> for more information on the W3C SMIL language (accessed 06.08.2010).

and fusion processes; on the output side, message selection, modalities and output coordination would be achieved according to user and context.

- G4 – *Control over fusion mechanism*: algorithms used to fuse multimodal input data can be quite complex and deliver different results according to the algorithm or its settings. Thus, description languages should take into account fusion parameters and ways to control them, for example by allowing choice between different algorithms, or by allowing management of fusion parameters.
- G5 – *Control over time synchronicity*: actual human-machine dialog description should give control over time synchronicity: when multiple events can all lead to a given action, how should the system fuse data if those events are activated at the same time? Thus, the fusion process would greatly benefit from control over time synchronicity, for example by taking into account the CARE properties presented in section 2.3.3.
- G6 – *Error handling*: error handling should be taken into account early on. Multimodal systems feature a large number of potential error sources, from the recognizers to the integration to the answer selection. Hence, a language for description of multimodal interaction should provide some way to handle errors and recognition mistakes, for example by allowing default choices to be specified, or encouraging the design of guided dialogues.
- G7 – *Events management*: a mechanism for events description and management should be taken into consideration, as events seem a natural way for people to think about how their multimodal application should work.
- G8 – *Input and output sources representation*: some way to represent the actual input and output sources can also be interesting, as the creator of the multimodal user interface wants to have control over which recognizer is used, and possibly be able to tune some parameters.

Table 6. *Eight guidelines for four purposes.*

	G1	G2	G3	G4	G5	G6	G7	G8
Communication	X	X	X	X	X			
Learning	X	X	X	X	X			
Modeling		X	X	X	X	X	X	
Configuration		X	X	X	X	X	X	X

- G* – *Find the right balance between usability and expressiveness*: this follows from the discussion in Section 2, and is maybe the single most important guideline: any description language should find its place (and role) between usability and expressiveness, between the

human and the machine. This guideline is at a higher level and thus is not integrated in the following tables.

Table 6 presents the eight guidelines above and matches them with the four different purposes of a description language identified in section 4.2. Guidelines were ranked from the most user-focused (abstraction levels) to the most system-focused (input/output data representation), hence the diagonal shape adopted by the crosses in the table. The next section presents SMUIML, a language for description of multimodal interaction, which will help illustrate most of the presented guidelines.

4.5 The SMUIML Description Language

SMUIML stands for *Synchronized Multimodal User Interaction Modeling Language*. As its name implies, the language seeks to offer developers a language for describing multimodal interaction, expressing in an easy-to-read and expressive way the modalities used, the recognizers attached to a given modality, the human-machine dialog modeling, the various events associated to this dialog, and the way those different events can be temporally synchronized.

Description languages can be used as configuration scripts for tools allowing creation of multimodal interfaces. The SMUIML language has been primarily designed with this goal in mind. Indeed, SMUIML is used as the scripting language to describe multimodal human-machine interaction in the context of the HephaïstTK framework, as explained in section 3.5. Besides the goal of configuration scripting, SMUIML follows the goal of multimodal interaction modeling as it tries to guide the user by giving her a pattern for the creation of their applications; SMUIML is also used as a tool in a course on multimodal interaction at the University of Fribourg, so, obviously, has also the goal of a learning tool.

This section will give a broad view of the SMUIML Language, beginning with an overview of the language structure, followed by a detailed view of the three different levels described by the language. XML Schema specification of SMUIML may be found in Appendix 1.

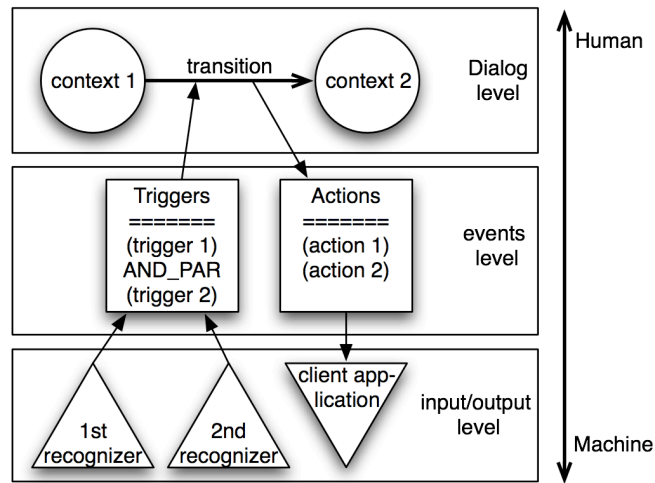


Figure 26. The three levels of SMUIML.

4.5.1 Structure of the Language

The way a SMUIML instance is split allows a clear separation between three levels necessary to the complete multimodal human machine dialog. As shows Figure 26, <recognizers> are at the lower, input/output level, <triggers> and <actions> form a middle level, devoted to events management, and the upper level contains the <dialog> description. This separation in different layers of abstraction is the foundation of the SMUIML language, and can be observed in state-of-the-art languages such as UsiXML or MIML (see Table 5). Moreover, the recognizer/trigger/action separation meets the device/language/task separation referred to in [82]. Indeed, recognizers are direct references to actual devices (and, in most cases, their corresponding software drivers or suites); triggers express the awaited events for a given recognizer, thus defining the set of all-possible well-formed expression for a specific application; actions depict the tasks the client application will have to handle; finally, the dialog part simply describes the relationships between these different pieces. This abstraction in three different levels allows components definition and reusability. In order to further enhance reusability, the upper dialog level allows definition of clauses that can be later used and extended.

A typical SMUIML document is divided in a number of sections (see Figure 27). First, the overall integration description is proper to a given client application. Whenever a new client application asks the HephaïstTK framework for a handle, it will have to give some identifier name (in the case of Figure 27, “client_app”). This identifier is used in the SMUIML file to identify which integration description is to be used for the current application. Hence, multiple applications can be

described in a same script, for example in a case where multiple small applications could access concurrently a number of available input modalities.

For a given client application, four main sections form the description of the multimodal interaction scenario. The first part, `<recognizers>`, indicates which particular recognizer will be tied to which modality. It also allows definition, per recognizer, of a number of variables that will be of use to the client application. The second section, `<triggers>`, lists the different events that will be of interest for the client application. The focus of this section is to model as generic triggers all events coming from the different recognizers. `<actions>` form the third section, and have the same goal of giving a generic model, but for to the output side of the framework. In particular, those `<actions>` will serve as a description of the framework-to-client application messages, as well as indicate the potential variables that would have to be transferred to the client application. `<triggers>` and `<actions>` form a set of building blocks, using the results of the various recognizers defined in the first section, and further called in the definition of the `<dialog>`. This dialog is described by means of a finite state machine. States are described by means of `<context>` elements, and to a given `<context>` are attached a number of `<transition>` elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description client="client_app">
    <recognizers>
      <!-- ... -->
    </recognizers>
    <triggers>
      <!-- ... -->
    </triggers>
    <actions>
      <!-- ... -->
    </actions>
    <dialog>
      <!-- ... -->
    </dialog>
  </integration_description>
</smuiml>
```

Figure 27. General SMUIML layout.

A complete SMUIML example is given in Figure 28. This example is taken from a multimodal drawing application realized with help of the Reactivision [51] computer vision framework and

Sphinx 4 speech recognition system. The example features two recognizers, three input triggers, one output action and part of the overall human-machine interaction dialog. All these elements will be detailed in the following subsections.

4.5.2 Recognizers

At the recognizers level, the goal is to tie the multimodal dialog scenario with the actual recognizers which the developer wishes to use for his application. In the context of the HephaïsTK framework, all recognizers are identified by a general name throughout the framework. This general identifier is thus used in SMUIML. The HephaïsTK framework keeps a list of recognizers, and their associated modality (or modalities). For example, if a number of speech recognizers are available, using one of them or the other is a matter of changing the “name” attribute of the speech-related `<recognizer>` element. As every event is defined afterwards in relation to the modality, and not a particular recognizer, this allows for more flexibility in the creation of the implementation. Moreover, design-wise, it enhances readability and allows developers to design their application before thinking about which recognizer they intend to use.

It is to be noted that the `<recognizers>` part is the only one where a tight link between the SMUIML language and a given tool (in this case, the HephaïsTK framework) appears. The modality the different recognizers provide is also indicated. The “modality” attribute is a string of characters, chosen from a list of keywords. This keywords list is created by the HephaïsTK framework at runtime against the available recognizers. In other words, the framework knows in real time which modalities it is able to offer.

SMUIML offers also the possibility to declare variables attached to a given recognizer. Those variables can be used in the other steps of the multimodal interaction description, for example when describing the actions to be sent to the client applications; the content of recognizers’ variables can be sent to an application, and the way a SMUIML script describes this variables passing uses the variables declaration feature of the SMUIML language. Data types of such variables vary from modality to modality, but are expected to be consistent within one modality; for example, every speech recognizer is expected to deliver text outputs describing the speech content, and thus datatype of such speech content will accordingly be set as strings of characters³¹. For a pointer-based

³¹ In the case of the HephaïsTK toolkit, it is not necessary for a given recognizer to directly output its results in the destined format; as every recognizer in HephaïsTK is managed by a software agent, the task of transforming the output of the recognizer into HephaïsTK-friendly data is the software agent’s responsibility. In fact, ensuring normalization of data format are one of the main tasks of the recognizers software agents, as discussed in subsection 3.5.2.

modality, such as a mouse, data type will either be x and y coordinates, or a sequence of coordinates in InkML.

Finally, the `<recognizers>` part of the SMUIML language can also shelter a number of properties to configure the HephaistTK framework. Those properties are simple name-value pairs, and override the configuration properties present in the standard XML configuration file of the framework. Those properties are not mandatory, and are not part of the standard SMUIML specification, as they relate to the specific use of SMUIML in the context of the HephaistTK framework. Nevertheless, they have been added into the language as a way to give indications to the framework without needing to resort to other configuration tools than the sole SMUIML script.

4.5.3 Triggers

Triggers are at the core of the transition mechanism of SMUIML. They describe a sub-set of interest from all the possible events coming from the different recognizers. A set of input events can hence be abstracted behind one trigger name, enhancing as much the script readability.

A standard trigger declaration is shown in Figure 28. In this example, two triggers are defined for the speech modality, regardless of the recognizer actually used. For example, the speech recognizer is simply assumed to send its results as strings of characters, and be scripted by means of a BNF-style grammar. A unique name or number identifies each `<trigger>`. As for the last trigger, attached to a Phidget (see Greenberg et al. [43]) RFID tags reader in this example, the three values declared are linked to actual tags numbers in the `<variable>` declarations in the recognizers; the tag numbers could also be used in place of variables. `<source>` elements are used to indicate the source modality of the trigger, as well as the awaited value which will trigger the event.

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description
    client="xpaint_client">

    <!-- declaration of recognizers -->
    <recognizers>
      <recognizer
        name="sphinx4" modality="speech"/>
      <recognizer name="reactivision"
        modality="reactivision">
        <variable name="posx"
          value="x" type="int"/>
        <variable name="posy"
          value="y" type="int"/>
      </recognizer>
    </recognizers>

    <!-- declaration of input triggers -->
    <triggers>
      <trigger name="return">
        <source modality="speech"
          value="return"/>
      </trigger>
      <trigger name="operation">
        <source modality="speech"
          value="rotate shape|move shape"/>
      </trigger>
      <trigger name="tools_one_hand">
        <source modality="rfid"
          value="select|line|freehand"/>
      </trigger>
    </triggers>

    <!-- declaration of output actions -->
    <actions>
      <action name="draw_action">
        <target name="xpaint_client" message=
```

```

        "draw $oper $shape $posx $posy"/>
    </action>
</actions>

<!-- declaration of H/M dialog -->
<dialog leadtime="1400">
    <context name="modification">
        <transition name="modif_clause">
            <par_and>
                <trigger name="operation"/>
                <trigger name="selected shape"/>
                <trigger name="position"/>
            </par_and>
            <result action="draw_action"/>
        </transition>
        <transition>
            <trigger name="return"/>
            <result context="start"/>
        </transition>
    </context>
</dialog>

</integration_description>
</smuiml>

```

Figure 28. A typical example of a SMUIML script.

4.5.4 Actions

<actions> are the equivalent of <triggers> for output. They describe the messages and their content that will form the communication channel between the Hephaïstos framework and its client application. A typical <action> declaration is shown on Figure 28.

The goal of those messages is to let the client application know in which state the framework finds itself in, by means of a clearly defined set of messages. Those messages can contain variables previously defined, such as the `\$posx` variable defined in Figure 28. The client application will then be able to know what content is to be restituted to the user. The choice was made not to offer extensive control over fission of modalities in SMUIML, for two reasons. First, having extensive

control over the way content is restituted can go as far as recreating a full rendering language, which was not the goal of this work. Second, mixing as little as possible the model and the view allows for better readability of the language. Hence, restitution of the content is up to the client application. This choice can however be a source of errors, as heavily modal applications would need some way to ensure synchronicity between the framework and the application.

4.5.5 Dialog

The `<dialog>` element describes the integration mechanisms of a SMUIML script. In essence, a `<dialog>` is a finite state machine, with transitions defined by the `<triggers>` and `<actions>` events that were presented in the former sections. States of the dialog are described by `<context>` elements, and `<transition>` elements define the rules for going from one context to the other. Each context has a unique name identifying it. One context must have a “start_context” attribute, defining it as the starting state. An “end_context” attribute also exists to describe a final state. A simple example of a `<dialog>` is represented in Figure 28.

When multiple triggers are present, the developer should be able to clarify time synchronicity issues, i.e. how the incoming multimodal triggers should be considered. The approach of SMUIML is to distinguish between parallel and sequential triggers, and between coupled and individual triggers. To denote those different cases, a set of keywords has been selected. Keyword “par” is for parallel triggers, “seq” for sequential triggers, “and” for coupled triggers, “or” for individual triggers.

Four elements to describe the different behaviors have been designed by mixing those four keywords, and linking them with the four CARE properties. `<par_and>` is to be used when multiple properties are to be fused together, as they all are necessary for the meaning extraction process. `<seq_and>` describes one or multiple individual triggers all necessary in sequence to trigger a transition. `<par_or>` describes redundant multimodal triggers having similar meanings. Each one is sufficient for the correct meaning to be extracted, but they all can be expressed at the same time by the user, increasing as such the robustness and recognition rate (for example, a user issuing a “play” vocal command and simultaneously pushing a play button). Finally, the `<seq_or>` element is to be used when multiple triggers can lead to the same result, but only one of them is to be provided. Those four integration describing elements can also be combined in order to express all kinds of multimodal interactions. In fact, as show comments in Figure 29, those four elements correspond to three of the four CARE properties of multimodal interactive systems (Complementarity, Redundancy and Equivalence) as defined in Coutaz et al. [24], with `<seq_or>` and `<seq_and>` both corresponding to

Complementarity, with a focus on ordering (or non-ordering) of the input events. The choice to drop the Assignment CARE property was made because we felt it was more meaningful to be able to differentiate sequenced and un-sequenced complementarity, than to differentiate equivalence of choice (one of a number of modalities is required, does not matter if two are selected at the same time) and assignment (one and only one of a number of modalities is required); In fact, assignment can be expressed by using one transition per modality. The CARE properties have revealed themselves a handy tool used by a number of tools to help formalize relationships between different modalities; choice was made to include them into SMUIML in order to answer the guideline of time synchronicity control.

```

<transition>
  <par_and>
    <!-- Complementarity -->
  </par_and>
  <seq_and>
    <!-- sequenced complementarity -->
  </seq_and>
  <par_or>
    <!-- Redundancy -->
  </par_or>
  <seq_or>
    <!-- Equivalence -->
  </seq_or>
</transition>

```

Figure 29. Triggers combination elements in SMUIML.

4.5.6 SMUIML language interpretation

As stated before, SMUIML is mainly used as the way to script the HephaisTK framework, presented in section 3.5. In consequence, both the HephaisTK framework and the SMUIML language share a strong bond. In fact, a good part of the way the HephaisTK dialog manager represents information is based on the structure of the SMUIML language.

Integration of HephaisTK into a given application also pinpoints the strong bonds between the framework and the language. This bond shows through directly in the way an application would make use of HephaisTK. Indeed, a binding declaration in Java would begin by the following line:


```
new HephaistkInitManager("client_app");
```

“client_app” refers to the application name every SMUIML script has to specify, and which is described in subsection 4.5.1. The referring architecture of HephaistK is described in Subsection 3.5.2. To enhance flexibility, callbacks to the client application are generated following the SMUIML script. Changing a few lines of the script can completely change the behavior of the multimodal framework, and as the callbacks are rather loosely tied, the developers are completely free to adapt their application, do testing on, or simply ignore, etc. those callbacks.

The callbacks themselves make reference to the <actions> part of the SMUIML script. Indeed, when writing the SMUIML script corresponding to his application, a developer will be asked to define a set of messages, which will be used for communication between the framework and the application. In practice, when the framework takes a fusion decision on a set of input data, those very messages will be used and sent to the client application. Upon reception, the client application is then completely free, regarding the actions it will take in answer to this message. This makes the link between the framework and potential applications rather loose, but in return the framework can be easily integrated in an already existing application.

4.6 Positioning of SMUIML

A qualitative evaluation of SMUIML has been achieved, and is detailed in the “Evaluations” chapter of this document, in section 6.1.

When confronted with the guidelines introduced in section 4.4, SMUIML takes into account abstraction levels by its three layers structure, events description with help of the <triggers> and <actions> elements, and representation of input sources with the <recognizers> elements. Control over the fusion process is allowed by specifying attributes in the <dialog> part: for example, the time frame in which a given meaning frame has to be fused can be specified with help of the “leadtime” attribute. Time synchronicity aspects are managed by the different <par_or>, <par_and>, <seq_or> and <seq_and> elements. On the case of data modeling, as SMUIML is tied to a framework, all data modeling is supposed to be managed inside this framework. Sire et Chatty [107] admit in this regard that data modeling can be present in multiple places, but could be present in XML modeling, for example to be compliant with EMMA. We believe data modeling could be based on EMMA description but should not be part of the multimodal interaction description itself, mainly for readability reasons. Control structures are a relevant point: basically, a state machine such as the one

present in SMUIML dialog description is able to describe any application process, however control structures such as “switch” or “while” statements can enhance the readability of a given language. But control structures imply an imperative programming-oriented language; the way SMUIML dialog description was designed would make unnatural the use of standard control structures such as switch statements or loops: control structures are inherent to the state machine model. A complete overhaul of the language would be needed for the integration of such control structures. Adaptability to user and context is not taken into account yet in HephaistTK and SMUIML, though mechanisms to ease plasticity integration are present. Finally, error handling is also not yet present in the language, as can be seen in Table 7, but is a planned work, that will take place jointly with the enhancement of the HephaistTK framework on error handling.

Table 7. The SMUIML language and the eight guidelines.

	SMUIML	<dialog>	<triggers>	<actions>	<recognizers>
G1. Abstraction levels	X	X	X	X	
G2. HMI modeling	X	X			
G3. Adaptability to context/user					
G4. Control over fusion process	X	X			X
G5. Time synchronicity	X	X			
G6. Error handling					
G7. Events management	X		X	X	
G8. I/O data representation	X				X

Finally, about guideline G* (Usability vs. expressiveness balancing), SMUIML was created with readability in mind, as well as keeping as much expressiveness as possible. Nonetheless, on the usability side, being able to use a Graphical User Interface to create a SMUIML script would go a long way toward better user-friendliness.

4.7 Conclusion

This chapter, dedicated to multimodal interaction modeling, presented our study of the design of a language targeted at multimodal human-machine dialog modeling. After presentation of the state of the art on languages targeted at multimodal human-machine dialog modeling, recurring features of such languages were extracted and analyzed. This analysis spawned the following features as keys for a successful multimodal human-machine dialog modeling language: abstraction layers, events

management, time synchronicity management, plasticity handling, web/desktop orientation, error handling, and data modeling. The spectrum of use of such description languages was then discussed, as well as the time synchronicity problem. Based on discussion of those two aspects of such description languages, and the analysis of the state of the art, eight guidelines for the good design of description languages for multimodal human-machine dialog modeling were proposed and discussed. Based on these guidelines, the SMUIML language (Synchronized Multimodal User Interaction Modeling Language) was created, and presented. General organization and specific features of a typical SMUIML script were then discussed, as well as the positioning of the language.

5 Multimodal Fusion: Dialog Management & Algorithms

"It worked."

Robert Oppenheimer, after the Trinity atomic bomb test.

5.1	Multimodal Fusion of Input Data: A State of the Art	96
5.1.1	Levels of Fusion of Input Modalities	96
5.1.2	An Historical View on Fusion Engines	98
5.1.3	Dialogue vs Fusion Management: a Discussion.....	98
5.1.4	Dialogue Management.....	100
5.1.5	Algorithms for Fusion of Input Modalities	101
5.2	Time Synchronization Issues.....	104
5.3	Fusion Handling in HephaistTK: Affected Components and General Organization	106
5.4	Fusion Algorithms in HephaistTK	108
5.4.1	Meaning Frame-based Fusion of Input Data	109
5.4.2	HMM-Based Fusion of Multimodal Input	113
5.5	A Benchmarking Tool for the Assessment of Fusion Algorithms	116
5.5.1	A testbed for fusion engines	117
5.5.2	Metrics and Software Requirements	120
5.6	Conclusion.....	121

Following Chapter 3 on architectures for frameworks dedicated to prototyping of multimodal interfaces, and the study of how these architectures can help fusion, and Chapter 4 on multimodal dialog modeling, this chapter will now focus on the inner workings of the Integration Committee, and in particular on the Fusion Manager. A state of the art on multimodal fusion will first help pinpoint what exactly can be considered as multimodal fusion, and which role play multimodal fusion algorithms, especially in regard to dialog management. Synchronization issues on the fusion side will be explored, and an example of a tricky fusion case, which will accompany this thesis in the two

following chapters, will be presented. Fusion handling in HephaistTK will then be detailed, in particular in relation with interpretation of the SMUIML scripts. Staying in the context of the HephaistTK framework, the particular fusion algorithms available in HephaistTK will subsequently be examined in details: first, a state-of-the-art-comparable meaning frames fusion algorithm, then an HMM-based fusion algorithm. Finally, a tool for benchmarking fusion algorithms will be presented.

5.1 Multimodal Fusion of Input Data: A State of the Art

Fusion of input modalities is one of the particular features that distinguish multimodal interfaces from unimodal interfaces. The goal of fusion is to extract meaning from a set of input modalities and pass it to a human-machine dialog manager. Fusion of different modalities is a delicate task, which can be executed at three different levels: at data level, at feature level and at decision level. Three different types of algorithms can in turn manage decision-level fusion: frame-based algorithms, unification-based algorithms or hybrid symbolic/statistical fusion algorithms. This section will detail each of those levels, algorithms and architectural foundations, as well as review the different roles played by the fusion modules and dialog manager modules in an architecture such as the one presented in section 3.4.

5.1.1 Levels of Fusion of Input Modalities

Sharma et al. [105] consider the data-, feature- and decision-levels for fusion of incoming data. Each fusion scheme operates at a different level of analysis of the same modality channel. As a classic illustration, consider the speech channel: data from this channel can be processed at the audio signal level, at the phoneme (feature) level, or at the speech (decision) level; in the same way, a video stream can be processed at the raw images (data) level, at a visual features level, or at a semantic, e.g. image content (decision) level. Fusion will mostly occur at a same level between different modalities (see Figure 30).

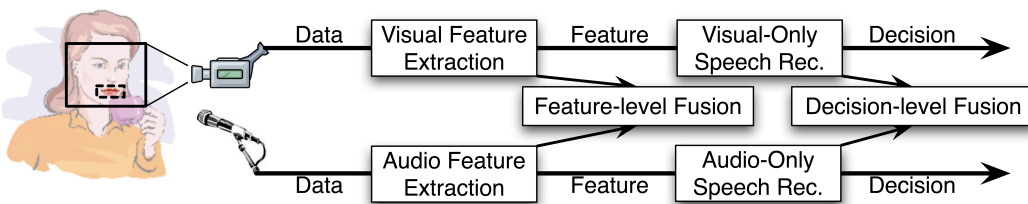


Figure 30. The various levels of multimodal fusion.

- *Data-level fusion* is used when dealing with multiple signals coming from modality sources producing very similar data (e.g., two webcams recording the same scene from different viewpoints). With this fusion scheme, no loss of information occurs, as the signal is directly processed. This benefit is also the main shortcoming of data-level fusion: due to the absence of pre-processing, it is highly susceptible to noise and failure.
- *Feature-level fusion* is a common type of fusion when tightly-coupled or strongly time synchronized modalities are to be fused. The standard example is the fusion of speech and lip movements. Feature-level fusion is susceptible to low-level information loss, although it handles noise better. The most classic architectures used for this type of fusion are adaptive systems like artificial neural networks, Gaussian mixture models, or hidden Markov models. The use of these types of adaptive architecture also means that feature-level fusion systems need numerous data training sets before they can achieve satisfactory performance.
- *Decision-level fusion* is the most common type of fusion in interactive multimodal applications. The main reason is its ability to manage loosely-coupled modalities like, for example, pen and speech interaction. Failure and noise sensitivity is low with decision-level fusion, since the data has been preprocessed. On one hand, this means that decision-level fusion has to rely on the quality of previous processing. On the other hand, unification-based decision-level fusion has the major benefit of improving reliability and accuracy of semantic interpretation, by combining partial semantic information coming from each input mode which can yield “mutual disambiguation” [85]. Typical algorithms for decision-level fusion are presented in subsection 5.1.5.

Table 8 summarizes the three fusion levels, their characteristics, sensitivity to noise, and usage contexts.

Table 8. Characteristics of fusion levels.

	Data-level fusion	Features-level fusion	Decision-level fusion
Typical input	Raw data of same type	Closely coupled modalities	Loosely coupled modalities
Level of information	Highest level of information detail	Moderate level of information detail	Mutual disambiguation by combining data from modes
Noise/failures sensitivity	Highly susceptible to noise or failures	Less sensitive to noise or failures	Highly resistant to noise or failures
Usage	Only used in same modalities combination	Used for fusion of closely coupled modalities	Most widely used type of fusion
Application examples	Fusion of two video streams	speech recognition from voice and lips	Pen/speech interaction

5.1.2 An Historical View on Fusion Engines

Lalanne et al. studied extensively in [61] the evolution of fusion engines, thus this subsection will mainly summarize their analysis. Lalanne et al. first consider a “breakthrough” phase within the development of fusion engine, portrayed by Bolt’s “Put-That-There” [10], although Bolt never made explicit reference to fusion. Then, the research field moved a bit, with a focus on the identification of problems raised by fusion. In particular, CUBRICON [78] is considered as the first explicit representation of fusion engine behaviour. Yet, this work only shed the light on further challenges, which had to wait a number of other projects before getting tackled. In particular, Koons et al. [56] integrated speech, gaze and hand gestures, the PAC-Amodeus architecture [80] introduced the “Melting Pot” fusion representation, Quickset [23] studied a number of different fusion algorithms (on which we will come in the following sections), and Johnston & Bangalore [50] used finite state machines in order to represent a context-free grammar. After these explorations which helped establish a firm groundwork on fusion, more recent publications helped explore the field: Latoschik [62] extends the work from Johnston & Bangalore in order to represent quantitative temporal aspects in the fusion engine. Flippo et al. [37] and Portillo et al. [95] combine techniques from Quickset and PAC-Amodeus to create a hybrid fusion engine exploiting both time-frame and unification mechanisms for solving ambiguities. Finally, Bouchet & Nigay [12][13] extend their early fusion work based on the PAC-Amodeus architectural model by defining a set of micro fusion engines as reusable and composable software components.

5.1.3 Dialogue vs Fusion Management: a Discussion

If the definition of what entails “fusion of input modalities” in a multimodal interactive system is largely accepted (see for more details subsection 2.1.1), in actual implementations, fusion management can take many different shapes, in particular when coupled with a dialog manager module, like in the architecture detailed in section 3.4. In fact, when talking about “fusion”, many multimodal interaction practitioners will refer either to the integration committee as a whole, or at least to the fusion manager-dialog manager pair; others will refer solely to the actual fusion algorithms present in the core of the fusion manager. We would like to dig a bit further on this topic, and discuss specific roles of the two entities.

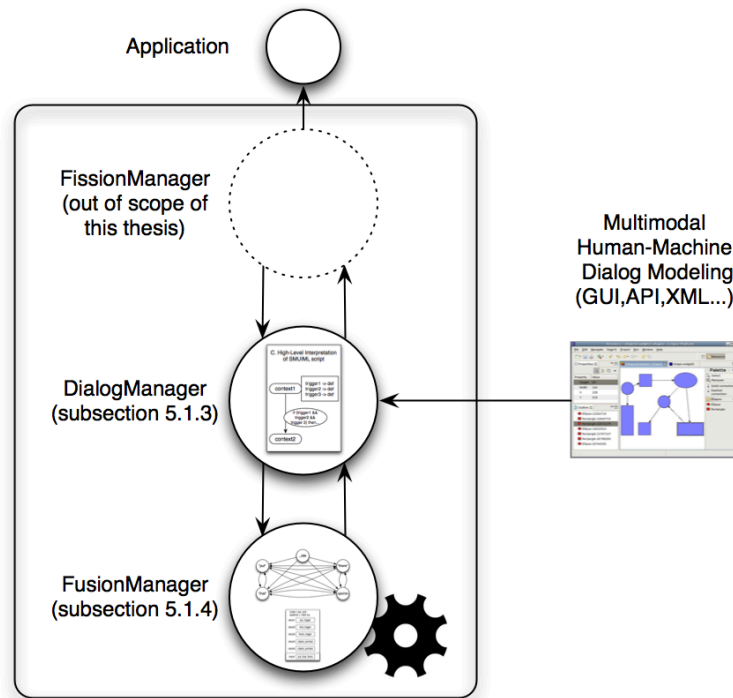


Figure 31. Roles and relationships of Dialog and Fusion Managers in a generic multimodal architecture.

Figure 31 gives a detailed view of the way a generic Integration Committee is structured. In the case of the architecture presented in Section 3.4, the role of the Fusion Manager is mainly to manage the different fusion algorithms, and literally frame them. Thus, the Fusion Manager encapsulates the algorithms, instantiates them based on information received from the Dialog Manager, dispatches data received from the different input recognizers, and informs the fusion algorithms of any relevant changes in the execution environment; however, initial notice of such changes would originate from the Dialog Manager. We will not discuss too deeply the specific roles of the Fission Manager, as it fell out of the scope of this thesis. Nonetheless, its overall working would be a mirror image of the Fusion Manager, with encapsulation of a number of fission algorithms, and decision of rendering results based on the decision taken by the fusion algorithms, and context information transmitted by the Dialog Manager.

As one could suspect, *the real conductor behind the inner workings of the whole Integration Committee is the Dialog Manager*. In fact, its main responsibility is to indicate to the others elements every change of context; but this means it is also responsible for parsing and/or interpretation of multimodal human-machine interaction dialog modeling, management of the high-level dialog description and communication of said description to the other modules, as well as transfer of all

fused data to other parts of the multimodal architecture, and to the client application. It also includes interpretation of all synchronization elements specified at a high-level, and transformation of those high-level description, to an algorithm-level description.

Thus, getting back to the question at the beginning of this subsection: how to define fusion, at a technical level? Our answer would be the following one: there are two different “fusions” one could talk about in a multimodal system:

- On one hand, fusion management, which, in the proposed multimodal architecture, would include both the Dialog Manager and the Fusion Manager;
- On the other hand, the actual fusion algorithms working in the core of the Fusion Manager.

5.1.4 Dialogue Management

Dialogue management in interactive multimodal applications, as its name implies, manages the information passing between results produced by the fusion engine and the applications interested in these results. Beyond the mere “message passing” duties, dialogue management modules in a multimodal application have frequently the task to filter out unknown or irrelevant results; they do so by getting knowledge on the current state of the multimodal system and of the client applications. The work of the dialogue management system is complicated by the possibility of different lags occurring at different stages of the multimodal dialogue process, as will be detailed in section 5.2. It is to be noted that dialog management is sometimes intermixed, or directly inspired, by either the fusion algorithm, or the overall architecture of the application. Thus, the careful reader should not be surprised to see recurring concepts between the approaches presented below and approaches presented in the former sections and chapters, the typical example being frames-based architectures, and/or fusion. We consider these recurring concepts as a good illustration of the confusion which is still prevalent in multimodal interaction research between the roles of fusion, dialog management and context management. Bui [17] considers four different approaches to dialog management:

- *Finite-state and frame-based approaches*: in this kind of dialog management approach, the dialog structure is represented in the form of a state machine. Frame-based models are an extension of finite-state models, using a slot-filling strategy in which a number of predefined information sources are to be gathered [22].
- *Information state-based and probabilistic approaches*: these approaches try to describe human-machine dialog following information states, consisting of five main components:

informational components, formal representations of those components, a set of dialog moves, a set of update rules and an update strategy [113].

- *Plan-based approaches*: the plan-based approaches are based on the plan-based theories of communicative action and dialog [22]. These theories claim that the speaker's speech act is part of a plan and that it is the listener's job to identify and respond appropriately to this plan [20].
- *Collaborative agents-based approaches*: these approaches view dialog as a collaborative process between intelligent agents. The agents work together to obtain a mutual understanding of the dialog. This induces discourse phenomena such as clarifications and confirmations [84].

Finally, it is worth mentioning machine learning approaches related to user, task and context modeling. Novel research fields related to machine learning, such as social signal processing [108], help building a refined representation of the user in her collaborative context. Adaptability can then be addressed with the help of machine learning, by watching the users' behavior in the sensed context [35].

5.1.5 Algorithms for Fusion of Input Modalities

Typical algorithms for decision-level fusion are frame-based fusion, unification-based fusion and hybrid symbolic/statistical fusion.

Frame-based fusion [116] uses data structures called frames or features for meaning representation of data coming from various sources or modalities. Meaning frames are an artificial intelligence concept first devised by Marvin Minsky in 1974 in his article "A framework for representing knowledge" [73]. In the original proposal from Minsky, frames are seen as artificial intelligence data structures which represent substructures of an idea, representing objects as attribute-value pairs. Frames are then connected together to form the whole idea.

The multimodal interaction version of meaning frames are a bit different, in that the "ideas" which frames represent are tightly linked to interaction operations. For example, one frame could correspond to a given computer operation, and all its substructures would represent the different interactions that would lead to that particular computer operation. Vo & Wood give in [116] a way of using frames for multimodal interaction, in which semantic interpretations are merged incrementally. Figure 32, taken from this reference, explains how Vo & Wood achieve fusion of input via meaning frames. In their system, frames are incrementally filled by the different sources, then merged recursively, until a satisfying hypothesis can be picked up by the Dialog Manager.

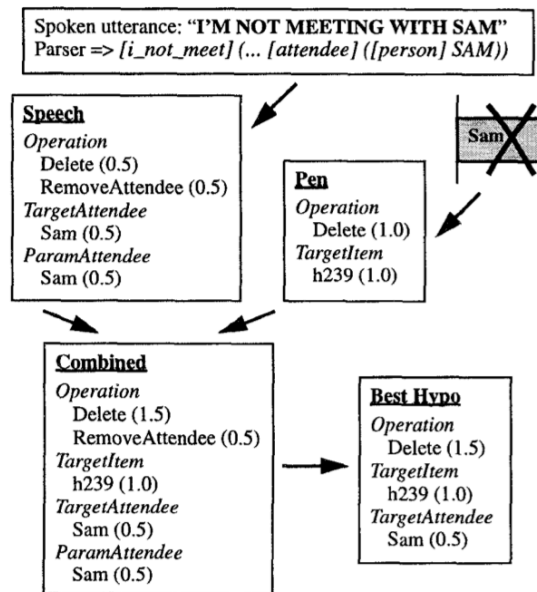


Figure 32. Example of meaning frames merging, in Vo & Wood [116].

Unification-based fusion [49] is based on recursively merging attribute-value structures to obtain a logical whole meaning representation. As an example, illustrated in Figure 33, consider a user interacting with a military planning application, and wishing to create a barbed wire line: the user will first utter "barbed wire" as an intent to create a barbed wire object in the application. The multimodal system will correctly get the intention, but will not execute the command, as it still needs a line object. The user will then use the pen modality to draw a line on the screen, which coordinates will be forwarded to the fusion module. The structure being now complete, the command can be executed by the system. As with frames, attribute-value structures are used, the main difference lying in the way conflicting data is to be resolved. Nonetheless, both fusion algorithms share the same conceptual grounds, as Johnston et al. [49] reckon: "*Vo and Wood 1996 present an approach to multimodal integration similar in spirit to that presented here [...]*".

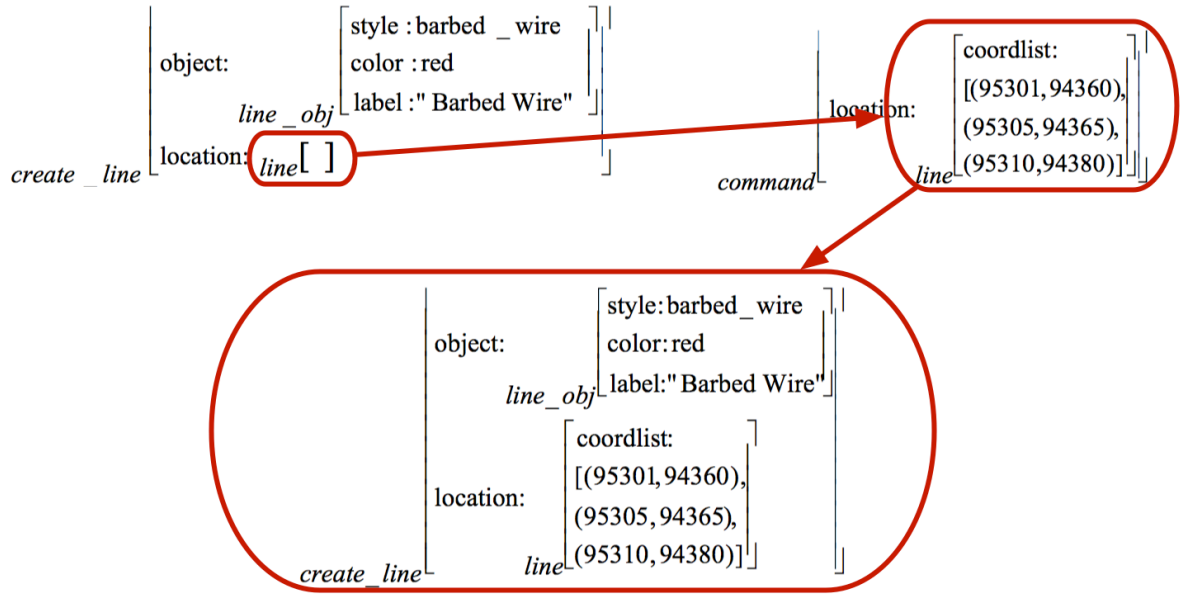


Figure 33. Example of unification-based fusion algorithm, in Johnston et al. [49].

Symbolic/statistical fusion [124] is an evolution of standard symbolic unification-based approaches, which adds statistical processing techniques to the fusion techniques described above. These kinds of “hybrid” fusion techniques have been demonstrated to achieve robust and reliable results. A classical example of a symbolic-statistical hybrid fusion technique is the Member-Team-Committee (MTC) architecture used in Quickset [123]. In this algorithm, speech and gesture inputs are organized in a so-called “associative map” (see Figure 34), which describes the speech-gestures pairs considered as valid. Next to this associative map is the actual statistical-based algorithm, which is based on three layers: the bottom layer is composed of recognizers members, each of which is a local posterior estimator with an assigned input variable subset, a specified model type and complexity, and a given training and validation data set. The upper layer is composed of “teams” of members, which cooperate to determine integration of the multiple members. There is a number of different teams in order to reduce integration uncertainty. Finally, the decisions of the different teams are submitted to a “committee”, which makes a final decision after comparing the empirical posterior distributions of the different teams.

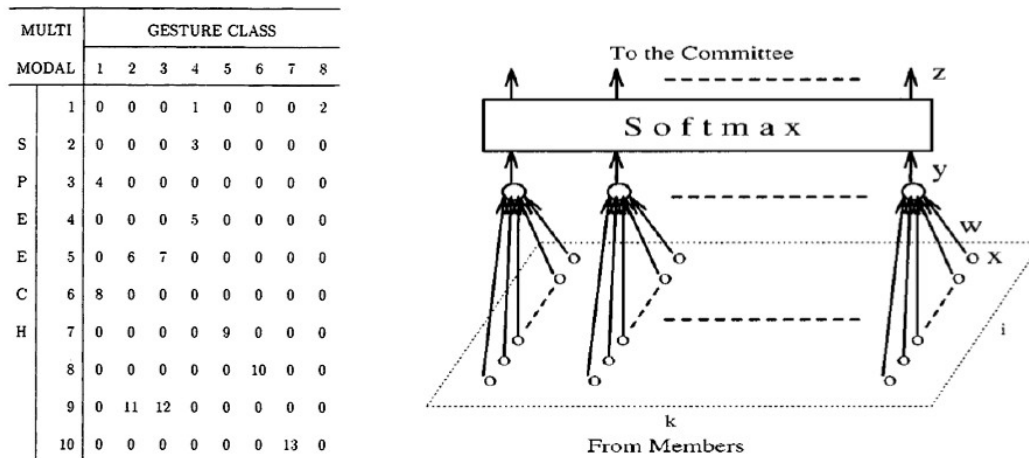


Figure 34. Associative map and general organization of Member-Team-Committee fusion algorithm, as in Wu et al. [123].

It is also interesting to note that machine learning has already been applied for fusion of input recognizers' data for non-interactive systems. Machine learning has been mainly applied at the feature level, with fewer works achieved on decision level fusion with assistance from machine learning. An example of such work is Pan et al. [92], who proposed context-dependent versions of Bayesian inference method for multisensory data fusion. Nonetheless, Jaimes & Sebe [48] reckon that "further research is still required to investigate fusion models able to efficiently use the complementary cues provided by multiple modalities".

As Jaimes & Sebe highlight, currently "most researchers process each channel (visual, audio) independently, and multimodal fusion is still in its infancy". Thus, multimodal interaction researchers have work to achieve in order to attain efficient multimodal fusion, with careful consideration of the different available modalities and the way modalities interlock. Machine learning will be of interest in order to attain such a goal. Besides multimodal fusion, machine learning will help multimodal applications take into account the affective aspect of communication – emotions based on their physiological manifestations [66], such as facial expressions, gestures [21], postures, tone of voice, respiration, etc.

5.2 Time Synchronization Issues

As already underlined in section 4.3, the time dimension is highly important in multimodal interaction. Section 4.3 talked about synchronization issues on the multimodal human-machine

dialog modeling side of a multimodal application, and this section focuses on synchronization issues on the dialog management and fusion algorithms side.

For a given multimodal command, the way modalities are synchronized will strongly impact the interpretation. As an illustration of some problems which can arise within even a simple multimodal application, consider a multimodal music player, which would allow users to control the different commands with a number of modalities, in a redundant or complementary way, depending from the command. Now, imagine a vocal command (“play next track”) combined with a pointing gesture, to play a musical track. The interpretation of this command can greatly vary depending on the time synchronicity and on the sequence in which commands have been produced.

For instance, in the following application, in which voice and gestures are used simultaneously to control this music player, depending on the order in which modalities are presented the interpretation varies:

- <pointing> “Play next track”: will result in playing the track following the one selected with a gesture;
- “Play” <pointing> “next track”: will result in first playing the manually selected track and then going to the following one at the time “next” is pronounced;
- In parallel <pointing> && “Play next track”: in this case, the system should interpret that the two modalities are used in a redundant way;
- “Play next track” <pointing>: In this case, the system can either interpret the commands as being redundant or as being complementary and, depending on its choice, will play a different track.

While the cases above seem not too much ambiguous, other cases can be imagined, in which it becomes unclear what the user wanted to say and what should a “perfect” multimodal fusion engine interpret.

Potential interpretation problems in multimodal fusion engine may also occur for technical reasons impacting on the precision of time synchronicity. For this reason, the fusion engine (and the related benchmark) should consider multiple potential causes of lag:

- Delay due to technology (ex.: speech recognition);
- Delay due to multimodal system architecture;
- User differences in their habitual multimodal integration pattern [88][91].
- Macro vs. micro-temporal fusion [81].

The fusion algorithms which were implemented in HephaistTK considered these different causes of lag, and sought to offer an answer to them, as will be seen in the coming sections.

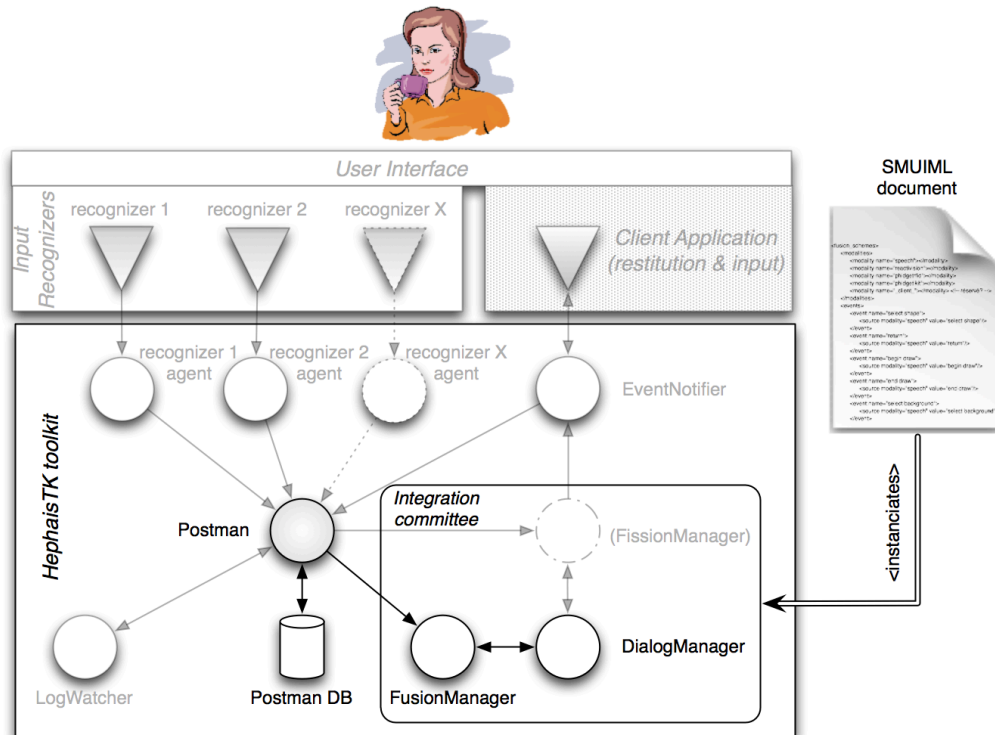


Figure 35. Parts of HephaistTK affected by fusion processes.

5.3 Fusion Handling in HephaistTK: Affected Components and General Organization

One of the goals of HephaistTK was to be able to test a number of different fusion algorithms, and compare them. Toward this goal, the architecture of the framework has been devised to enable implementation, design and “hot” swapping of different fusion algorithm, all this in a as seamless as possible way for the client application.

Management of fusion of input modalities in HephaistTK makes use of the following elements, highlighted in Figure 35:

- **Postman:** the postman serves two goals, in regard to fusion of input modalities. First, it serves as a repository of all data received by the framework. As such, fusion algorithms are able to tap into this stock of input data, most evidently to determine the current state of the human-

machine dialog, but also to study how input data evolved with time, do statistical queries on specific types of data, or get back to former results.

- **SMUIML script:** the SMUIML script is supposed to determine the way the fusion algorithm is tuned when launching the framework. As such, the description provided by a SMUIML script is supposed to be of a high enough level to be applied to the different fusion algorithms offered by HephaisTK³².
- **DialogManager:** while having to manage dialog between the framework and the client applications, the DialogManager in HephaisTK has also the task to respond to reception of the SMUIML scripts, interpret them, and forward all relevant information to the FusionManager, so that the latter is able to configure the currently chosen fusion algorithm. The DialogManager has also the duty to know and inform the other modules which are part of the Integration Committee of the current state and context in which the framework is supposed to be.
- **FusionManager:** obviously, the FusionManager module has the task to manage the fusion algorithms, configure them based on the data provided by the interpretation of the SMUIML script by the DialogManager, and offer the possibility to hot-swap fusion algorithms in the course of the activity of a given application. It also manages reception of input data coming from the different input recognizers, as well as subscriptions of the different fusion algorithms to relevant message categories.

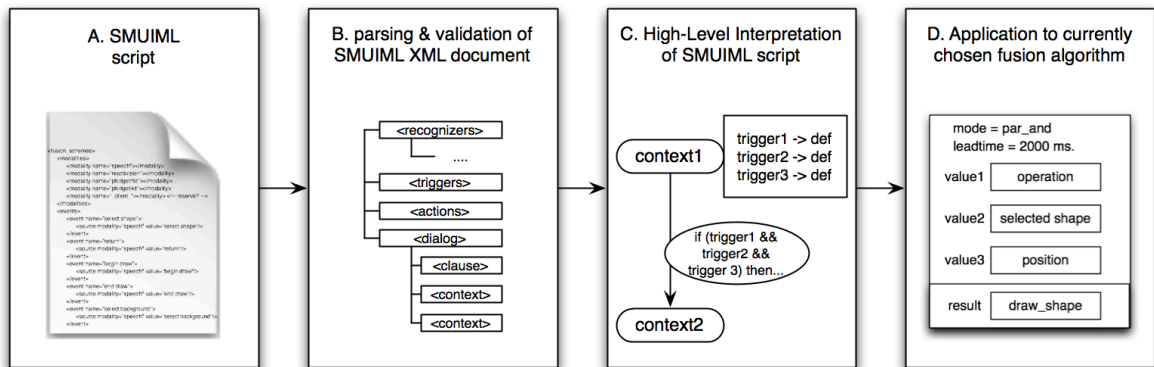


Figure 36. Steps of SMUIML interpretation in relation with fusion management in HephaisTK.

Figure 36 illustrates the different steps a SMUIML interaction description goes through when fed to the HephaisTK framework. First (stage A.) in the form of a XML file, the SMUIML document

³² The question of knowing whether the SMUIML language is able to script any type of fusion algorithm has been left aside as a future potential research direction. At the time of writing of this thesis, SMUIML is able to formulate an adequate description of the wished human-machine interaction for the fusion algorithms considered in section 5.4.

gets read and parsed by the framework, at which stage the basic XML structure is checked³³. Raw elements and attributes are then stored in memory for the Dialog Manager to process them (stage B.). Based on the current status of the framework, including which recognizers are currently available (or announced), the dialog manager will then handle the SMUIML elements and attributes, check their semantics and verify whether the provided description is runnable as is by the framework. Semantic errors such as discrepancies between data types used by triggers and announced by their corresponding recognizer, or unavailable recognizers, or unreachable contexts in the dialog description, would then be reported. If no errors arose at this stage, the SMUIML description is converted into a high-level state machine description stored in memory (stage C.). Contexts are converted as the states, triggers and fusion rules as transition rules, and action as transition results. Finally, this high-level state machine is sent to the Fusion Manager for fusion algorithms instantiation (stage D.). When added to the framework, each fusion algorithm is supposed to be able to instantiate itself based on the stored high-level description described in stage C, provided this description is error-free. The second precondition for a fusion algorithm to be able to run properly in HephaïsTK is to be able to describe multiple different states, so as to adapt to the <context> definitions of SMUIML. Thus, as all fusion algorithms are instantiated at the launch of the framework, and are supposed to be able to jump directly to a specific context, fusion algorithms in HephaïsTK can be hot-swapped during execution.

5.4 Fusion Algorithms in HephaïsTK

In its current state, the Fusion Manager in the HephaïsTK framework offers three different ways to handle input data. They are the following ones:

- “No” fusion, where input data is directly fed to the different client applications, without any processing or fusion. This is useful when a client application just needs the recognizers’ data without further addition, or wishes to manage fusion in its own terms. With a “no fusion” scheme, HephaïsTK acts as a simple aggregator of different input sources.

³³ When parsing the SMUIML XML file, it is to be noted that HephaïsTK does not achieve validation through a XML Schema or DTD description (although the XML Schema description for the SMUIML language does exist – and is included in Appendix 1 of this document). It was felt that, as HephaïsTK has to interpret the SMUIML file, it would be more efficient to do validation of the SMUIML syntax during this phase. Moreover, it allows to give more consistent error reports to the developer, due to the tight integration of SMUIML in the HephaïsTK toolkit. A typical example would be the availability of specific recognizers in front of the dialog description in SMUIML.

- “Meaning frames” based fusion, with a fusion algorithm based on state-of-the-art frame-based fusion (see subsection 5.1.5). More details on this fusion scheme and its implementation in HephaïsTK are given in subsection 5.4.1.
- “HMM-based” fusion, with a novel fusion algorithm devised in the context of this thesis. More details on this fusion scheme are given in subsection 5.4.2.

It is to be noted a fourth way to handle input data exists, although it barely qualifies as “fusion of input data”: It is a “Dummy” fusion, where no fusion or management of input data whatsoever takes place; this scheme is mainly used for test and debugging purposes, as input information stays in the framework, and is only outputted to the log window.

5.4.1 Meaning Frame-based Fusion of Input Data

Meaning frames fusion in HephaïsTK operates in much the same way than how Vo & Wood [116] present them, with a few differences. First, in HephaïsTK, frames depend solely on the fusion manager, and so are not directly dependant from the input sources. Second, in place of recursive merging, a subset of meaning frames are always present in memory, depending on the current context. These frames are continuously fed, until a satisfying hypothesis occur. Third, the way a hypothesis is considered “satisfying” rests on a set of rules, attached to every frame, and taking into account temporal and synchronicity features.

In detail, meaning frames in HephaïsTK are created at launch, depending of the SMUIML and its high-level interpretation, as explained in section 5.3. Frames in HephaïsTK are mainly composed of three different elements: slots, results and rules (see Figure 37).

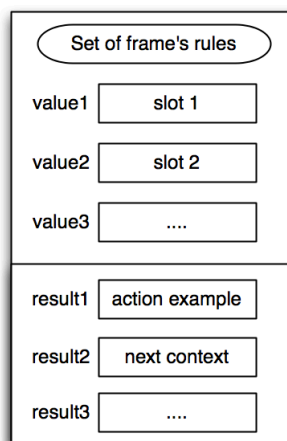


Figure 37. A typical meaning frame in HephaïsTK.

Slots constitute the body of meaning frames in HephaistTK. Each slot in a frame defines a potential condition which would lead to the frame being triggered. Each slot is focused on one and only one modality; but the condition defined can be very tight (like “pointer has to enter a zone on screen defined by the following coordinates”) or very permissive (like “a RFID tag with any value has to be read”), up to specific definitions (like “awaiting any of the following speech content: ‘hello’, ‘hi’, ‘good morning’, ‘howdy’...”). Slots are defined according to `<triggers>` elements in a SMUIML script, and adapted according to the current situation of the framework, and the available input recognizers. When data comes from given recognizers, it is sent to every slot in every meaning frame, which have a corresponding modality definition. Checking validity conditions defined within a slot is the responsibility of the slot itself; the meaning frame only knows whether the slot’s condition was met or not, and for which input.

Results list every action to be taken in the case the meaning frame is checked as valid. The number of actions attached to a meaning frame can range from 0 to 255. Actions are typically of two different categories:

- Context change: framework will activate the current set of meaning frames, and activate the set of meaning frames corresponding to the specified new context.
- Message to client application: any number of messages, specified by the developer in the `<actions>` part of a SMUIML script, will be sent to predefined client applications. These messages can contain the value of specific variables, but will also carry the specific events which led to the validation of their attached meaning frame; thus, the client application has the possibility to check how fusion was done – and decide to accept or reject the decision of the framework³⁴.

Rules specify the conditions for the meaning frame to be considered valid or not. Rules depend upon the synchronization specifications in the SMUIML script, and in particular the `<par_and>`, `<par_or>`, `<seq_and>` and `<seq_or>` elements, as well as specific settings, such as the lead time. Every time a new input is accepted by one of the meaning frame slots, the meaning frame will check its validity rules, and return the result to the fusion engine.

As the reader can suspect, meaning frames in HephaistTK are closely tied to SMUIML. Furthermore, the range of synchronization capabilities of SMUIML offer different possibilities when

³⁴ It is however to be noted that, in the case of meaning frames implementation in HephaistTK, rejection of a fusion engine’s decision will not influence future decisions. The HMM-based fusion algorithm presented in subsection 5.4.2, however, is expected to be able to take full advantage of the possibility to accept or reject its decisions, and adapt to this.

describing multimodal interaction. As an illustration, let us consider the classical “put that there” example, expressed in SMUIML.

```
<transition leadtime="1500">
  <par_and>
    <trigger name="put_trigger" />
    <trigger name="that_trigger" />
    <trigger name="there_trigger" />
    <trigger name="object_pointed_event" />
    <trigger name="object_pointed_event" />
  </par_and>
  <result action="put_that_there_action" />
</transition>
```

Figure 38. Classical “put that there” example expressed in SMUIML.

As shown in Figure 38, a first way to express the “put that there” would be to create one trigger event for each of the three words, one trigger event for pointing events, and ask all the three words and two pointing events to happen mandatorily (hence the `<par_and>` rule) in a 1500 ms time window; this transition would then result in a message sent to the client application, indicating the occurrence of a “put that there” event, along with the two pointed positions. The corresponding meaning frame to the SMUIML script of Figure 38 is illustrated in Figure 39. Five slots corresponding to each of the five expected triggers form the core of the meaning frame, rules are composed of the *par_and* synchronization mode, asking every one of the five slots to be filled for validation, all of this in a 1500 ms time frame. Finally, the meaning frame has only one result, i.e. the message to be sent to the client application.

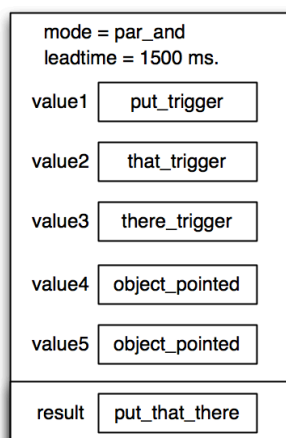


Figure 39. Meaning frame corresponding to a simple “put that there” example.

This first example has however a few flaws. First, all the five triggers are expected, but no order is specified with the “par_and” rule. Thus, a completely chaotic command such as *pointing* “there, that” *pointing* “put” would be accepted. Although one could think this shouldn’t present that big of a problem, recognizers mistakes coupled with numerous inputs could lead to false positives. A second flaw follows from the first: it is assumed that the first pointing event is related to “that”, and the second pointing event to “there”. Although it should be true in most cases, it is still a very broad assumption. A first way to correct these flaws would be to use a “seq_and” rule, asking for every trigger to be present, in the specified sequential order. But then, the benefit of multimodal interfaces to allow humans to express commands in their favorite style would be lost. A far better solution would be then to blend different fusion rules, such as in Figure 40.

```

<transition leadtime="1500">
  <seq_and>
    <trigger name="put_trigger" />
    <transition>
      <par_and>
        <trigger name="that_trigger" />
        <trigger name="object_pointed_event" />
      </par_and>
    </transition>
    <transition>
      <par_and>
        <trigger name="there_trigger" />
        <trigger name="object_pointed_event" />
      </par_and>
    </transition>
  </seq_and>
  <result action="put_that_there_action" />
</transition>

```

Figure 40. “Put that there” example, expressed in a slightly more complex way in SMUIML.

In the example of Figure 40, three different events are expected to happen sequentially: first, a “put” speech event, then, a complex event, formed by a “that” speech event and a gesture pointing event, and finally a second complex event, formed by the “there” speech event and a second gesture pointing event. As the two sub-events do not expect any particular order, any of the following user commands would be accepted by the system: “put that” *point* “there” *point*, put *point* “that there” *point*, put *point* “that” *point* “there”, or “put that” *point* *point* “there”. Such an

example is a good illustration of balancing users' expressive liberty with system's resistance to errors. The corresponding meaning frames to Figure 40 is depicted in Figure 41. As one can see, the whole fusion scheme is composed of one main meaning frame and two sub-frames; validation of each of the two subframes will be required for the main frame to be validated, in a recursive way.

Results of evaluation conducted with meaning frames fusion will be detailed in section 6.2.

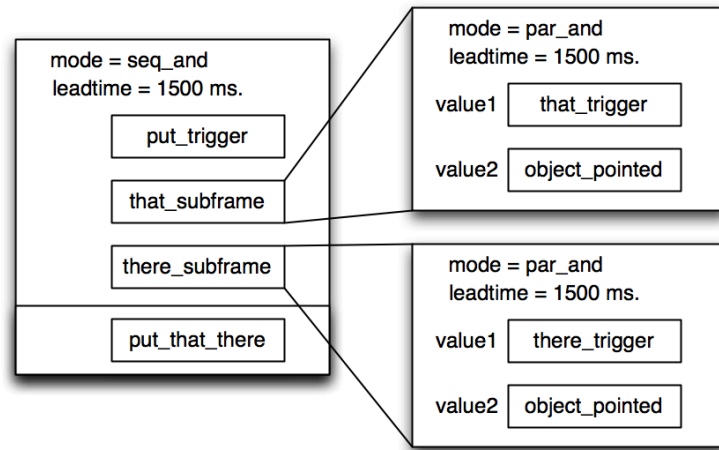


Figure 41. Meaning frames corresponding to the second “put that there” example.

5.4.2 HMM-Based Fusion of Multimodal Input

With fixed, precise situations, meaning frames present clear benefits. However, in presence of fuzzier situations, meaning frames fall short of giving correct results in numerous cases. Furthermore, as HephaisTK might be used in the future for studies on adaptivity to context, as well as ability to learn from fusion errors, work on a machine learning-based algorithm began. The goal when creating this fusion algorithm was to lay the ground for context management and user feedback studies; it is however to be noted that neither context management nor user feedback studies have been considered in this particular thesis. Results of evaluation conducted with the HMM-based fusion algorithm presented in this section will be detailed in section 6.2.

When considering which type of machine learning would be used, hidden Markov models were privileged because of their easy adaptation to time-related processes. Hidden Markov models (*HMMs*) have historically been used in temporal pattern recognition tasks, such as speech, handwriting or gesture recognition. As fusion of input data also focuses on time-dependant patterns, HMMs were seen as the most obvious choice when considering the different alternatives among

statistical models. The interested reader will find deeper information on hidden Markov models in numerous academic references, including [44][96][97].

The first challenge when integrating HMMs as fusion algorithms for multimodal interaction is to map features and states to the actual human-machine interaction model. In our case, the SMUIML modeling (and its high-level counterpart) was used as basis. A second challenge was to minimize, or even set aside, the need for training of HMMs before being able to actually use them. As will be explained in this section, training in our system is achieved through simulation of expected inputs described in SMUIML.

Going from the high-level human-machine interaction description depicted in section 5.3, to the actual implementation of the HMM-based fusion algorithm is done in the following way. Contexts are modeled with help of one HMM each: so, changing contexts in HephaistK is equivalent to switching from one HMM to another. This also means that adapted training done after the initialization phase will not transfer directly during contexts switching, although this could be done, as will be seen afterwards. Triggers (i.e. input events) are modeled as individual output states (observations) of the HMM. Basically, one input event described in the SMUIML script corresponds to one output state of the HMM. Furthermore, one more “idle” state is added in all context HMMs. Transitions, as described in SMUIML, and which form the core of the multimodal human-machine dialog description, are defined as sequences of output states.

As an example, let’s consider the “put that there” example once again. As our example comprises only one interaction, there is only one context, thus one HMM to consider. Four different input events are expected, i.e. three speech-related triggers (“put”, “that” and “there”), and one gesture-related trigger, i.e. pointing events. Thus, when adding an “idle” state, the basic “outer” HMM architecture would look like Figure 42.

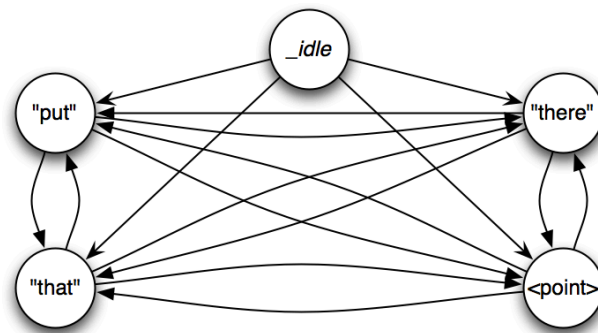


Figure 42. HMM for the “put that there” example.

Then, every time an input event is taken into account by the HephaïsTK framework, it is sent to the fusion manager, which in turn considers, for the list of input events which happened in a specified time window, potential combinations. Those combinations are then injected in the HMM. The Viterbi algorithm is subsequently used on the HMM to extract the most probable state sequence. This state sequence is compared to a set of expected observations sequences, defined by the transitions defined in the SMUIML script. For example, the following succession of input events: “put” *point* “that” *point* “there” would generate a state sequence like the one in Figure 43. If a match is found, the messages and/or context changes defined in the SMUIML document for the given context are applied.

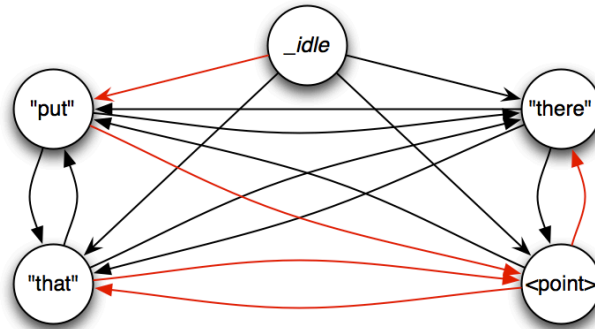


Figure 43. Finding a match for a sequence of input events.

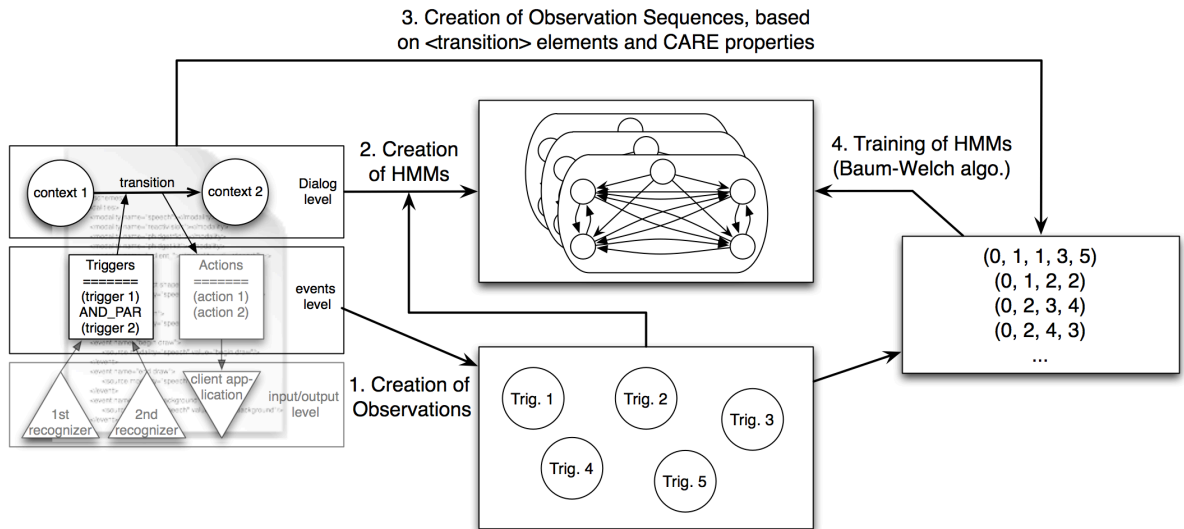


Figure 44. Instantiation and preliminary training of HMM-based fusion in HephaïsTK, based on the dialog description in SMUIML.

Instantiation of HMM-based fusion is done as depicted in Figure 44. when the SMUIML script is transformed into the high-level description (see section 5.4), first, all triggers defined in the

SMUIML document are parsed, and observations are created, based on this list of triggers (step 1). Basically, one observation corresponds to one trigger definition, with the exception of the “idle” observation state. Then, for each context defined, one HMM is instantiated, using only observations relevant to the currently processed context (step 2). All <transition> elements declared in the SMUIML script are then parsed, and, based on the synchronicity rules, a set of all acceptable observations sequences is generated for each context (step 3). Finally, each HMM undergoes a preliminary training stage, with the sets of acceptable observations sequences injected into their applicable HMM; the Baum-Welch algorithm is used to find the unknown parameters of the HMM, hence allowing the fusion algorithm to be directly used, without explicit training from the user (step 4).

As pointed out in the introduction of this subsection, the main goal when creating this HMM-based fusion algorithm was to lay the foundations for adding in HephaistTK the ability to adapt to context and user feedback. Preliminary tests of adaptability already occurred in the context of the HephaistTK framework, with the possibility for users to give feedback to the framework. Specific method calls in the ‘HephaistkInitManager’ class were thus added. User feedback would then be propagated in the rest of the framework, and in particular to the FusionManager. In the case of the HMM-based fusion algorithm, a first, but effective, user feedback currently handled is for the user to tell “no” to the system, indicating the latest fusion result is not what was expected. When a “no” feedback comes to the FusionManager, the latest used observations sequence is considered invalid, and the HMM which did provide the unwished for result is retrained, with less weight attributed to the seemingly invalid observation sequence. Results obtained with this first implementation have been considered encouraging, yet too preliminary for inclusion into this thesis.

5.5 A Benchmarking Tool for Assessment of Fusion Algorithms

This section³⁵ proposes a benchmark to measure the performance of multimodal fusion engines in a replicable and controlled way. For this purpose, we discuss in the following paragraphs about the possibility to set up a benchmark, a software infrastructure, and a metric of performance, in order to compare precisely the quality and efficiency of multimodal fusion engines.

³⁵ This section is a reworked and extended version of the following publication: Dumas, B., Ingold, R., and Lalanne, D. 2009. Benchmarking fusion engines of multimodal interactive systems. In ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 2009). ACM, pp. 169-176 [27].

5.5.1 A Benchmarking Tool for Fusion Engines

To allow replicable testing in the case of multimodal fusion engines, the challenge is to create a set of problematic use-cases supporting their quantitative evaluations. In order to bypass the problems related to recognition errors introduced by each modality recognizers (speech, gesture, emotions, etc.), which occur before the multimodal fusion itself, we propose to simulate the recognizers outputs and feed these outputs directly to the fusion engines. The goal of this benchmarking tool is to focus on fusion algorithms and rules; furthermore, by simulating the recognizers output, we are also able to simulate incorrect outputs, and assess how fusion engines react to recognizers failures.

As illustrated on Figure 45, for a given benchmark, i.e. a temporal and multimodal events' stream resulting from simulated multimodal recognizers, a fusion engine will generate a series of interpretations in time.

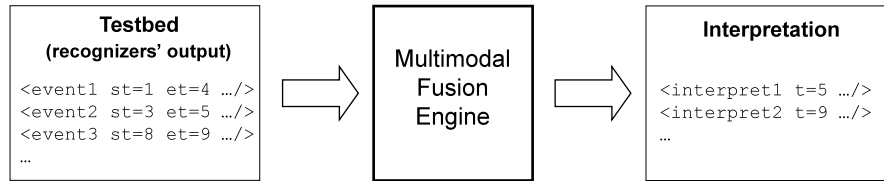


Figure 45. A multimodal fusion engine processes a series of multimodal events to generate a series of interpretations.

The resulting interpretation by the multimodal fusion engine can then be compared with a ground truth associated with the benchmark, in order to measure its performance, as illustrated on Figure 46. As discussed in the following section, various factors will help computing a performance metric (response time, confidence and efficiency).

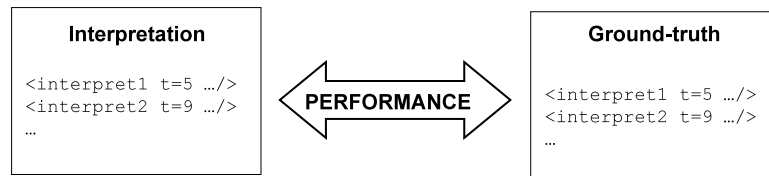


Figure 46. A ground truth allows to rate performances of the fusion engine, according to the interpretations it gave.

As mentioned above, the benchmark we propose simulates the output of various recognizers. For the sake of standardization, the representation of this stream of events uses the Extensible MultiModal Annotation markup language (EMMA) [36]. EMMA is an XML markup language used for containing and annotating the interpretation of user input by multimodal recognizers. Examples

of interpretation of user input are a transcription into words of a raw signal, for instance derived from speech, pen or keystroke input, a set of attribute/value pairs describing their meaning, or a set of attribute/value pairs describing a gesture. The interpretation of the user's input is expected to be generated by signal interpretation processes, such as speech, gesture and ink recognition. By using EMMA for the representation of the benchmark and ground truth data, we slightly divert the language from its original role envisioned by the W3C Multimodal Interaction Working Group. However, we feel that the EMMA language perfectly fits this new role of simulation and ground truth data representation.

In order to be relevant, the benchmark should allow testing most of the major difficulties related to multimodal fusion. This opens an important research question: what are the issues related with multimodal fusion engines in interactive systems? Which difficult use cases or combination of events can generate interpretation errors?

In search for answers, two formal representations targeted at modeling multimodal human machine interaction could provide us with a relevant framework for the benchmark [11][34]:

- The CASE model (see section 2.3.2), focusing on modality combination possibilities at the fusion engine level;
- The CARE model (see section 2.3.3), giving attention to modality combination possibilities at the user and system level.

Since the goal of the benchmark can be reworded as the task of measuring how well fusion engines are able to interpret the intention of the user and its usage of multimodality, the CARE model seems the most suited to structure the benchmark; The CARE properties model the various usage of multimodality that a user can intentionally achieve to control an interactive system.

To illustrate our point, we propose to encode with EMMA the “play next track” example presented in section 5.2 as an illustrative benchmark. The following EMMA is composed of two sequences, the first corresponding to the benchmark description, and the second sequence corresponding to the ground truth related to the benchmark. The EMMA code for the benchmark description would read like depicted in Figure 47.

```

<emma:emma version="1.0">
  <emma:one-of id="main">
    <emma:sequence id="testbed">
      <!--    <pointing>+"Play next track" case benchmark    -->
      <emma:interpretation id="gesture1" emma:medium="tactile"
        emma:mode="video">
        <content>track_pointed</content>
      </emma:interpretation>
      <emma:interpretation id="speech1" emma:medium="acoustic"
        emma:mode="voice" emma:time-ref-uri="#gesture1"
        emma:offset-to-start="300">
        <content>play</content>
      </emma:interpretation>
      <emma:interpretation id="speech2" emma:medium="acoustic"
        emma:mode="voice" emma:time-ref-uri="#gesture1"
        emma:offset-to-start="400">
        <content>next track</content>
      </emma:interpretation>
      <!--    other benchmark elements would come here    -->
    </emma:sequence>
  </emma:one-of>
</emma:emma>

```

Figure 47. Testbed description in EMMA.

Figure 48 shows the corresponding ground truth interpretation (as presented in Figure 46). `<emma:derived-from>` elements allow to connect the ground truth interpretation to the different benchmark input elements that should lead to it. The message corresponds to the value that the fusion engine is supposed to produce. In the whole EMMA benchmark file, we believe this is the only value dependant from the fusion engine, i.e. which would have to be adapted from one engine to another. Even when taking into account this `<message>` element (which is considered as application-specific instance data), the whole EMMA benchmark file is fully compliant to the W3C EMMA 1.0 recommendation [36].

Of course, the benchmark presented above is a toy example and a more complete and serious benchmark should be discussed and agreed on by the overall community. In particular, if the CARE properties model brings to light a number of problematic fusion cases as shown above, it certainly does not deliver an exhaustive catalog of every difficult or tricky cases a fusion engine might encounter; such a catalog can only be achieved by collecting the real-world experience of many multimodal interaction practitioners.

```

<emma:sequence id="groundtruth">
  <!-- <pointing>+"Play next track" case groundtruth -->
  <emma:interpretation id="message1">
    <emma:derived-from resource="#gesture1" composite="true"/>
    <emma:derived-from resource="#speech1" composite="true"/>
    <emma:derived-from resource="#speech2" composite="true"/>
    <message>play_next_of_pointed_track</message>
  </emma:interpretation>
  <!-- other groundtruth elements would come here -->
</emma:sequence>
</emma:one-of>
</emma:emma>

```

Figure 48. Ground truth interpretation part of benchmark specification.

5.5.2 Metrics and Software Requirements

We propose the following quantitative and qualitative metrics to measure the quality of a given multimodal engine according to a series of multimodal recognized events. Quantitative metrics are measured by means of the machine, whereas qualitative metrics necessitate the intervention of humans in order to be completed. For each multimodal event we plan to measure in a quantitative way:

- *Response time*: time that the fusion engine takes to return an interpretation after receiving multimodal inputs.
- *Confidence*: Level of confidence of the machine response, based for example on confidence scores indicated in the EMMA benchmark interpretation elements.
- *Efficiency*: success or failure of the fusion engine to interpret correctly the benchmark entries. Efficiency is measured by confronting the machine interpretation against the ground truth data.

An efficient fusion engine should answer reliably and quickly to the requests of the users; furthermore, an efficient fusion engine should be extensible and easy to use. Although such characteristics are much harder to measure dynamically, they are nonetheless important. Thus, other features of fusion engines to be measured in a more qualitative way would be the following ones:

- *Adaptability*: whether a fusion engine is able to adapt itself to context and user.
- *Extensibility*: how much a fusion engine can be extended to new or different input sources.

On a related dimension, the benchmark itself should be characterized, thus helping developers outline shortcomings of their fusion engines. Characterization would be achieved by developers and practitioners on a set of guidelines to be defined. Such characteristics would be:

- *Expressive power, or ability to be used by non-programmers*: the developer-, or user-friendliness (usability) of the mechanisms used to configure the fusion engine.
- *Level of complexity*: level of complexity (low/medium/high) of the test, following a number of criteria. For example, a test requiring complementary fusion based on incomplete or noisy data would be more complex than straightforward equivalence of two input modalities.
- *Problem type*: each test could be characterized by a number of keywords, describing particular goals or features to be tested, such as temporal constraints, presence of noisy data, etc.

Logging is a common mechanism for most development teams in order to debug software, to trace its usage, or to record and analyze users' behaviors. In multimodal systems, the time constraint is highly important and all the modalities should be properly time-stamped and synchronized. Time-sensitive architectures need to establish temporal thresholds for time-stamping start and end of each input signal piece, so that two commands sequences can be identified. Indeed, when two commands are performed in parallel, it is important to know in which order the commands have been entered because the interpretation will vary accordingly, as seen in the previous section. Therefore, logging mechanisms are required for multimodal system benchmarking. In particular, as too important delays between the user input and the resulting output can ruin the user experience, ways to log data and timestamps passing through the fusion engine are recommended.

Any multimodal system able to log input events and fused multimodal events, as well as their timestamps, should be able to implement the proposed benchmark. Input events can be generated either directly from the EMMA file, if the multimodal system already uses EMMA as data transfer format, or by means of a tailored component taking the EMMA file as input. As for the results analysis, it can be achieved either on the fly, as will be presented in Section 6.2, or after running tests, by analyzing log files.

5.6 Conclusion

This chapter was dedicated to multimodal fusion algorithms, in particular elicitation of the difference between fusion and dialog management in a multimodal architecture, the design of a novel HMM-based fusion algorithm, as well as proposal of a benchmarking tool for replicable evaluation

multimodal fusion algorithms. The chapter began with a review of the state of the art in fusion algorithms for interactive systems. In particular, specificities of fusion for multimodal human-machine interaction were explored: level of fusion, as well as their characteristics, differences between dialog and fusion management, and typical algorithms for fusion of input data and dialogue management. Synchronization issues on the fusion side were also discussed.

The chapter went on with a discussion on how fusion algorithms are handled in the core of the HephaïsTK framework, and in particular the roles taken by the SMUIML script and the dialog manager. The different fusion schemes available in the HephaïsTK framework were then presented, with a particular stress on the two main fusion algorithms: meaning frame-based fusion and HMM-based fusion. Finally, a benchmarking tool to measure the performance of multimodal fusion engines in a replicable and controlled way was detailed.

6 Evaluations

*“C’est le marteau qui a l’avantage sur le maçon...
parce que l’outil sait exactement comment il doit être manié,
tandis que celui qui le manie ne peut le savoir qu’à peu près.”*

Milan Kundera, “Le livre du rire et de l’oubli”.

6.1	SMUIML Modeling Evaluation	124
6.2	Fusion Evaluation.....	126
6.3	HephaisTK Evaluation Through Use Cases.....	130
6.3.1	HephaisTK Programming Tutorial Examples	131
6.3.2	Proof of Concepts	132
6.4	Conclusion.....	135

This chapter presents about evaluations of the different topics discussed in the three former chapters. As the different topics tend to ask for very different evaluation styles, one section will be devoted to each of the three axes considered in this thesis. A few attributes tie the evaluations together, though. First, be it architectures for fusion, modeling of human-machine dialog or fusion algorithms, each of these topics is delicate to evaluate in a very thorough manner. In the case of architectures, evaluating them is more a question of capabilities, performances and robustness, as well as the usability vs. expressiveness duality discussed in section 3.4. In the case of modeling languages for multimodal interaction, evaluations would focus on one hand on sets of specific features like, for example, the guidelines discussed in section 4.4, and, on the other hand, on user evaluations to validate its usability. In the case of fusion algorithms, the metrics used would be the ones discussed in subsection 5.5.2. As the reader can suspect, if some specific quantitative measures can be assessed on each of these three topics, they can by no means give a complete picture of the specific qualities and shortcomings of a given architecture, modeling language or fusion algorithm. Thus, evaluations on each of these topics followed a more qualitative approach which will be detailed below.

In detail, evaluations on modeling languages for multimodal interaction will be first presented. As stated before, evaluation for a modeling language would follow two different trails: on one hand, assessment in front of a set of guidelines, and on the other hand, user evaluations to examine the usability of the modeling language. Assessment of SMUIML in front of specific guidelines has been already discussed in section 4.6; thus, section 6.1 below will focus on a user evaluation conducted with a batch of students. This evaluation is nonetheless purely qualitative, and should reflect opinions of a set of developers mildly familiarized with multimodal interaction and XML languages, in front of SMUIML.

Evaluations of the two fusion algorithms discussed in chapter 5 follow in section 6.2. Evaluations of these algorithms were conducted with help of the benchmark tool presented in section 5.5. In the absence of a broadly accepted catalog of problematic fusion cases, the algorithms were evaluated with a few identified problematic cases, thus once again, the overall evaluation of these fusion algorithms is by no means a full quantitative evaluation, and should be considered as a qualitative comparison on the specific capabilities of both algorithms.

Finally, assessment of the capabilities of the HephaïsTK framework and of its architecture took the form of a series of use cases. Four different use cases are discussed in section 6.3, from a simple “Hello World” example, to an application used in the context of Smart Meeting Rooms. Once again, this form of evaluation is seen as a way to demonstrate the capabilities of HephaïsTK, but should not be taken as a full quantitative evaluation.

6.1 SMUIML Modeling Evaluation

The SMUIML language allowed modeling of a number of different multimodal use cases, from a music player with simple multimodal commands, to an application allowing classification and visualization of documents based on different criterias, to a drawing table with speech and tangible input. Thus, the ability to model and manage different multimodal applications was empirically verified. Furthermore, the SMUIML language was checked against the set of eight guidelines presented in Section 4.4, thus giving a better picture of its capabilities and shortcomings. Nevertheless, the user-friendliness of SMUIML had still to be considered.

In order to compare the language’s expressiveness and usability, master degree students from a course on multimodal interaction were asked to devise their own version of a language allowing description of multimodal interaction. These students had already a bachelor degree in computer

science, and were pursuing their studies to get a master degree. The course on multimodal interaction delivered at the University of Fribourg (Switzerland) is an optional course in their curriculum and amongst the 30 students which were present the year this evaluation was done, six of them chose to delve deeper in multimodal interaction modeling. They already had an introductory course on multimodal interaction, but no extended experience. Thus, they represented developers with a strong interest and passing knowledge on multimodal interaction. These students had to first imagine a multimodal application, draw a storyboard detailing the main use cases of their application, and they were then given three weeks to invent a formalization to describe more deeply their application. They had no knowledge of SMUIML or any other description language on multimodal interaction, although they had already followed a course on multimedia-aimed description languages such as SMIL. The idea behind this task was to see how developers think about modeling multimodal interaction when they only have passing knowledge about multimodality.

First, most of the students addressed the problem by describing what “happens” in the system, i.e. events. Some of them built their language proposal only around events, others made a difference between “input” events and “result” events, and still others built chain of events. Nonetheless, non-specialists of multimodal interfaces, faced with the problem of describing multimodal human-machine interaction, show a tendency to first think about the actual events and their expected results. Thereafter, most proposals offered a way to model human-machine interaction dialog, either by “knitting” events and actions to and from the system, or by describing fixed chains of events. Then, some students tried to give a description of the actual hardware used; some others did not see the interest of giving this level of detail. It is nonetheless to be noted that the students were to create a high-level formalization of multimodal human-machine interaction. In a more “complete” approach, some link to the actual hardware would have to be specified, and some of the students paid attention to this. Students did sadly not have enough time to complete implementation to fully validate their scripts, but those scripts were nonetheless fed to HephaïstTK toolkit to verify their syntactic and semantic validity.

Finally, the students were confronted with SMUIML, and give a qualitative evaluation of it. Once again, the basic idea behind it was to get the opinion of non-experts in the field of multimodal interaction about the language. Students were asked to give a grade on the *expressiveness* and *usability* of the SMUIML language, with 1 as the lowest grade and 5 as the highest grade. They were also asked to provide comments justifying their grades. In regard to expressiveness, all students gave the SMUIML language a grade of 5, and justified their grades by explaining that the language met all their expectations. In regard to usability, half of the students gave the SMUIML language a grade of

4, and the other half a grade of 5. In particular, two remarks were raised about the `<dialog>` part: first, with complex application, the whole context diagram could become tedious to analyze and read; second, every trigger, action or context is identified with a unique name, without distinction, which can easily lead to duplicate names. Thus, as usable as a description language can be, producing such documents for large-sized applications can become tedious.

6.2 Fusion Evaluation

We used the benchmark presented in section 5.5 to compare the two algorithms presented in Chapter 5: meaning frame-based fusion algorithm, and HMM-based fusion algorithm. Moreover, we tested them with two different strategies of fusion, i.e. with and without temporal ordering constraints. HephaistK was configured for the benchmark using SMUIML. We described in SMUIML and EMMA the different “play next track” examples introduced in section 5.2, as well as their corresponding ground truth. In Figure 49 is a portion of the SMUIML script related to the first case of the benchmark. Complete SMUIML and EMMA scripts for the evaluations detailed below can be found in Appendix 2.

```
<dialog>
  <context name="start">
    <transition leadtime="1000">
      <seq_and>
        <trigger name="track_pointed_event"/>
        <trigger name="play_trigger"/>
        <trigger name="nexttrack_trigger"/>
      </seq_and>
      <result action="play_next_of_point_action"/>
    </transition>
  </context>
</dialog>
```

Figure 49. Extract of the `<dialog>` part of the SMUIML script used for the evaluation.

The three example cases “play next track” with the pointing gesture event happening respectively before, in the middle of and after the speech event were modeled. Modeling in SMUIML was achieved by decomposing the speech acts in two different sub-sentences (“play” and “next track”), although a speech act for each word would have been possible as well.

Results of feeding these three examples to the HephaïsTK framework are shown below, starting at Figure 50. Start and delay times are expressed in milliseconds. The trigger events are listed in the order they were fed to the framework, in three groups of three events, with only the order in which they were sent changing. For each group of three events, the ground truth (“awaited answer”) is indicated. The actual answer received from the HephaïsTK fusion engine is then reported, and coloured in green if it corresponded to the ground truth answer, in red otherwise. These events were generated empirically, with a few users mimicking the different events, and start time of each event being deduced from these tests.

The first batch of tests was done with the meaning frames algorithm detailed in subsection 5.4.1, and no sequential constraint whatsoever. Every test was constituted of three different events, for nine consecutive events in all. As one can see in Figure 50, the first example case of pointing a track, and asking to play the next track of it, led to a first false answer from the fusion engine, which assumed that the user was asking to play the pointed track; then, with the “next track” command, the system finally returned the awaited answer. The second case was a priori correctly understood, but the “next track” event lead the fusion system to believe in a complex “play next of pointed track” event. The third test is exactly the repetition of the second one. All in all, without temporal restrictions, in front of ambiguous inputs, the “best match” strategy of meaning frames shows its limits, as it has no real trace of input history, as well as no assumption on potential following inputs.

Start ti...	Modality	Content	Awaited Answer	Actual Answer	Correct?	Delay
0	gesture	track_pointed	play_next_of_pointed_track		false	413
300	speech	play	play_next_of_pointed_track	play_pointed_track	false	111
400	speech	next track	play_next_of_pointed_track	play_next_of_pointed_track	true	11
3000	speech	play	play_pointed_track		false	278
3250	gesture	track_pointed	play_pointed_track	play_pointed_track	true	27
3500	speech	next track	next	play_next_of_pointed_track	false	23
6000	speech	play	next		false	421
6100	speech	next track	next	next	true	320
6400	gesture	track_pointed	next	play_next_of_pointed_track	false	20

Figure 50. Meaning frames, no sequential constraints ($\langle \text{par_and} \rangle$).

When adding temporal constraints, results get definitely better, as shows Figure 51. The reason is quite simple: temporal constraints drastically limit the number of possible input sequences expected by the system. As can be seen, the constraints allow all but one command to return correct answers. The mistake can be traced to a double definition of the “next” multimodal command, with at least two meaning frames resident in memory having the possibility to lead to it. Overall, meaning

frames with sequential time order constraints behave far better than without any constraints; once again, this should come as no surprise, as sequential constraints offer to meaning frames the possibility to make assumptions on future input events.

Delay times are computed as the time between the input event dispatch and the fused message reception by the client application. It should be noted though that the value indicated is for the latest message received by the system, thus, values for intermediate results may be overwritten. With that in mind, in the case of meaning frames algorithm, one can see that delay between the moment the last input is fed into the system, and the moment the application receives the fusion results, is between 5 and 37 ms, with a mean value of 15.6 ms.

Start time	Modality	Content	Awaited Answer	Actual Answer	Correct?	Delay
0	gesture	track_pointed	play_next_of_pointe...		true	439
300	speech	play	play_next_of_pointe...		true	137
400	speech	next track	play_next_of_pointe...	play_next_of_pointed_track	true	37
3000	speech	play	play_pointed_track		true	265
3250	gesture	track_pointed	play_pointed_track	play_pointed_track	true	14
3500	speech	next track	next	next	true	5
6000	speech	play	next		true	420
6100	speech	next track	next	next	true	320
6400	gesture	track_pointed	next	!!! sent too late !!! next	false	19

Figure 51. Meaning frames, sequential constraints (<seq_and>).

The same test procedure has been applied with the HMM-based fusion algorithm, also with a first sequence of inputs without any sequential constraints, then with a second sequence of inputs with sequential constraint. Results of tests without sequential constraints are shown in Figure 52. As can be immediately observed, results are far superior compared to those of the meaning frames algorithm in the same conditions. In particular, only the first case expecting a “play next of pointed track” result fails, but the answer is still a viable one. It is to be noted the same test has been run with a first version of user feedback correction of fusion errors, and, when asked to revise its decision on this first case, the HMM-based algorithm would afterwards correctly infer the awaited answer.

Start time	Modality	Content	Awaited Answer	Actual Answer	Correct?	Delay
0	gesture	track_pointed	play_next_of_pointe...		false	412
300	speech	play	play_next_of_pointe...	play_pointed_track	false	111
400	speech	next track	play_next_of_pointe...	next	false	10
3000	speech	play	play_pointed_track		false	266
3250	gesture	track_pointed	play_pointed_track	play_pointed_track	true	16
3500	speech	next track	next	next	true	13
6000	speech	play	next		false	420
6100	speech	next track	next		false	319
6400	gesture	track_pointed	next	next	true	19

☐ mute

Figure 52. HMM-based algorithm, no sequential constraints (<par_and>).

If sequential time constraints are introduced, results are even better, as shown in Figure 53. Every test is correctly passed. Once again, it should not come as a big surprise, as sequential constraint drastically reduce the solutions space of the fusion algorithm. In our view however, the results of sequentially unconstrained HMM-based algorithm are the more interesting ones, as they demonstrate already good results with better conditions for users, as the solution space is bigger, and with possibility for supervised, real-time improvement.

Start time	Modality	Content	Awaited Answer	Actual Answer	Correct?	Delay
0	gesture	track_pointed	play_next_of_pointe...		false	433
300	speech	play	play_next_of_pointe...		false	132
400	speech	next track	play_next_of_pointe...	play_next_of_pointe...	true	31
3000	speech	play	play_pointed_track		false	267
3250	gesture	track_pointed	play_pointed_track	play_pointed_track	true	16
3500	speech	next track	next	next	true	25
6000	speech	play	next		false	441
6100	speech	next track	next		false	341
6400	gesture	track_pointed	next	next	true	40

☐ mute

Figure 53. HMM-based algorithm, sequential constraints (<seq_and>).

Performance-wise, delays between the last input event from recognizers for a given multimodal command, and the corresponding fused command being sent to the client application varies between 10 and 40 ms, with a mean value of 17 ms. This is slightly higher than delay times achieved with help of meaning frames, yet such delay times are still completely acceptable, from the user point of view, as they stay well below the 100 ms mark required for a user to feel that the system is answering instantaneously [72].

6.3 HephaistK Evaluation Through Use Cases

HephaistK being a framework for the creation of multimodal interfaces, it was only logical to do qualitative evaluations of the framework by creating a number of multimodal applications with help of HephaistK. This section will detail some of the creations done in the context of this thesis, and will begin with two simple examples, targeted at demonstrating the use of the framework. Two more complex “proof of concept” examples directly follow.

```
public HelloHephaistKWorld() {
    HephaistkInitManager hephaistManager = new
        HephaistkInitManager("helloworld_client");
    hephaistManager.addSMUIMLConfigFile(smuimlFileLocation);
    hephaistManager.startFramework();
    //end of framework instantiation

    hephaistManager.addMultimodalInputFusedEventListener(new
        MultimodalInputFusedEventListener() {
    public void fusedEvent(MultimodalInputFusedEvent event) {
        if(event == null) {
            System.out.println("received a 'null' event");
        }
        else{
            System.out.println("HelloWorld received an event with
                "+event.getMessages().size()+" messages!");
            Message mess = null;
            for(int i = 0; i < event.getMessages().size(); i++){
                mess = event.getMessages().get(i);
                System.out.println("Message "+i+" - message type: "
                    +mess.getMessageType()+" | message: "+mess.getMessage());
            }
        }
    }
    });
}
```

Figure 54. “Hello World” – Java part of example.

6.3.1 HephaistK Programming Tutorial Examples

The first example application that will be detailed in this subsection is the infamous “Hello World” example, found in nearly every computer science tool and language, and which helps beginner users to quickly obtain their first results. To use HephaistK in an application, one needs to provide two different elements: a SMUIML script detailing how the framework is supposed to behave, and a Java file with handles to the framework. This Java file, in the case of a “Hello World”-style application, would look something like in Figure 54. As one can see, the actual instantiation of the framework is done in about three lines of code. The first one (*HephaistKInitManager hephaisManager = new HephaistKInitManager("helloworld_client");*) creates the framework manager and asks it to take into consideration that it will have to communicate with an application named “helloworld_client”. The second line of code (*hephaisManager.addSMUIMLConfigFile(smuimlFileLocation);*) specifies to the framework the location of the SMUIML config file. Finally, the third line of code (*hephaisManager.startFramework();*) actually starts HephaistK. At this point, the agent framework is loaded, recognizers and framework agents are fired up, the SMUIML file is parsed, its high-level representation generated, the high-level representation is passed to the fusion manager, and the framework is ready to work. In the example of Figure 54, all lines of codes except the first few ones are dedicated to receiving, managing and displaying the events fired by the framework’s fusion engine.

Apart from this first example dedicated to showing beginner users how to launch HephaistK, another simple example application was developed. This one was a multimodal music player, with “play”, “pause”, “forward” and “backward” commands. All four commands could be triggered either with the mouse, with speech commands, or RFID tags. Figure 55 shows the GUI of this simple multimodal music player. The SMUIML and Java code for this example can be found in Appendix 3. A condensed version of the Java code is available in Figure 56. The curious reader will be able to confirm the readability of the code for such use cases.



Figure 55. Second simple example: a multimodal music player.


```

public class HephaistKMusicPlayer extends JFrame implements
MultimodalInputFusedEventListener, ActionListener{

    //{...}

    //here comes instantiation of HephaistK
    HephaistkInitManager hephaisManager = new
        HephaistkInitManager("musicplayer");
    hephaisManager.addSMUIMLConfigFile(SMUIML_FILE_PATH);
    hephaisManager.startFramework();

    //declaration of class as listeners implementer

    hephaisManager.addMultimodalInputFusedEventListener(this);
    // {...}
}

//as this class implements MultimodalInputFusedEventListener,
//we have to override this class
//@Override
public void fusedEvent(MultimodalInputFusedEvent event) {
    //'events' are messages such as declared in the <actions>
    //part of your SMUIML script
    for(Message message : event.getMessages()){
        if(message.getMessage().equals("previous")){
            previous();
        }else if(message.getMessage().equals("play")){
            play();
        }else if(message.getMessage().equals("pause")){
            pause();
        }else if(message.getMessage().equals("next")){
            next();
        }else{
            System.out.println("received new message");
        }
    }
}

```

Figure 56. Condensed Java code corresponding to Music player example.

6.3.2 Proof of Concepts

Proof of concepts examples were more complex examples, designed to show the capabilities of the HephaïsTK framework at creating real-world applications. The first one of these proofs of concepts was not directly created with help of HephaïsTK, but was initially a project developed from scratch by students in the context of the course on multimodal interaction given at the University of Fribourg. This project was a drawing table named the XPaint drawing table. On this table, two physical artefacts allow the users to draw on the table a set of shapes or tools selected by means of RFID-tagged tangible objects. Commands can also be selected by means of vocal commands, recognized with help of the Sphinx speech recognition framework. Additionally, specific commands as selection of colour or line width are expressed through specific hardware input devices like Phidgets sliders.

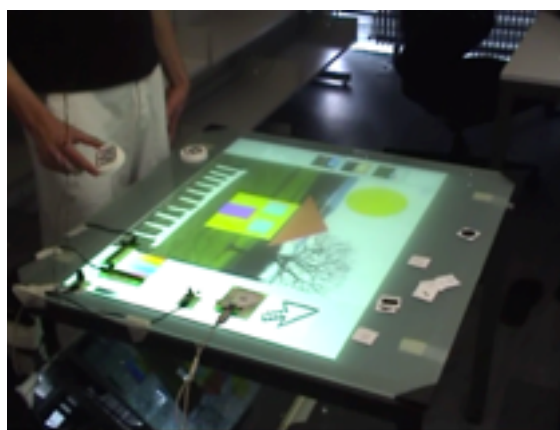


Figure 57. The XPaint drawing table.

The XPaint drawing table was re-modeled with help of HephaïsTK and the SMUIML language. One of the interesting features of the XPaint table is its strong modal architecture: there are three different “modes” of interaction, in which specific commands can be used, with the exclusion of the others. Figure 58 shows the different interactions and change of modes of use according to the different commands, as well as the different modalities used. In such a use case, the dialog modeling capabilities of SMUIML show their full strength, as such a high-level multimodal application can be modeled in no more than 150 lines of script programming. The complete SMUIML script modeling the XPaint table can be found in Appendix 4.

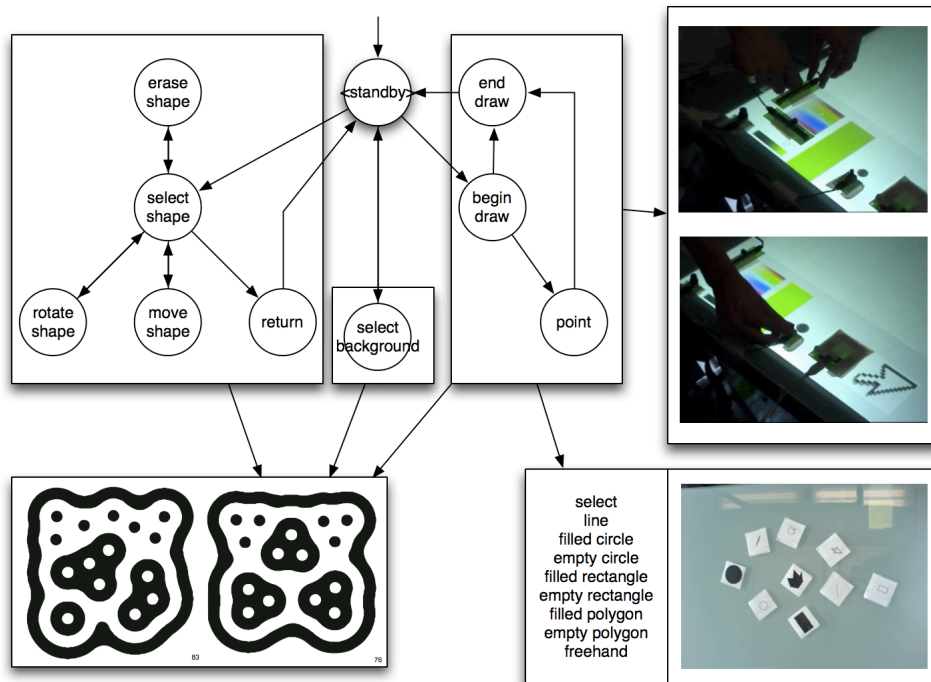


Figure 58. The three modes and different input modalities of XPaint.

The second proof of concept is a tool for document management in Smart Meeting Rooms. This application, named Docobro (see Figure 59), allows a user to visualize a number of documents, sort them by type or theme, add a previously RFID-tag paper document to the virtual cloud of electronic documents, and open and display the electronic representation of a paper document. Docobro can be manipulated by means of standard keyboard/mouse, speech commands or RFID tagged objects. In a typical use case, a user would come to the computer with an RFID-tagged document; a RFID reader would detect the presence of the document, match its identity against a set of identities stored in a database, and add it to the cloud of documents already present in the application. The cloud of documents can then be visualized by type and/or theme; selection of the visualization is done with help of speech, mouse, or RFID tags. Once added to the documents cloud, the virtual version of a specific document can be opened by a « open this » command, for example to be projected.

The interest of the Docobro application lies in the fact it was not an application specifically created to demonstrate the capabilities of HephaisTK, but was a tool needed for the management of the Smart Meeting Room at the University of Fribourg. It was thus a good candidate to demonstrate

the qualities of the HephaïstTK framework when needing to create quickly a working multimodal application. Docobro has been also demonstrated at IHM'08³⁶ and ICMI-MLMI'09³⁷.

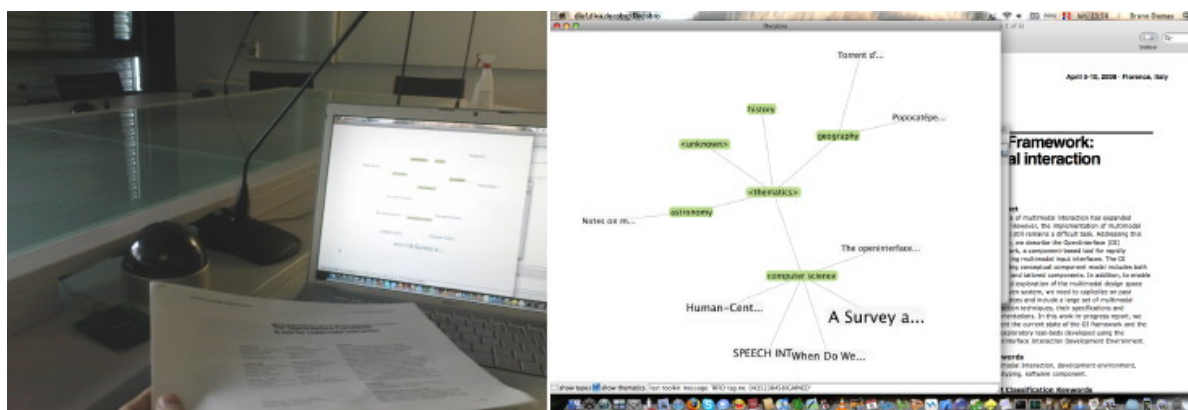


Figure 59. The Docobro documents management application.

6.4 Conclusion

This chapter was dedicated to the evaluations of the three different axes explored in the course of this thesis. Evaluations done with students of the SMUIML language were presented first. Results obtained when comparing the two fusion algorithms of Chapter 5 followed. Finally, a set of programming tutorials and proof-of-concepts applications helped illustrate the capabilities of the HephaïstTK framework.

Evaluations in the course of this thesis mainly took the form of qualitative assessments. In the case of the study of architectures for fusion of input data, as well in the case of modeling languages for multimodal dialog description, formal observation and study of the state of the art were the preferred methods of evaluating the capabilities of our solutions, with qualitative evaluations coming next as demonstrations of the validity of our approaches. Nonetheless, we are perfectly aware these forms of assessment are not able to give a complete picture of the capabilities of our work. More evaluation should then be executed for this picture to be thorough.

³⁶ Dumas, B., Lalanne, D., and Ingold, R. 2008. Démonstration: HephaïstTK, une boîte à outils pour le prototypage d'interfaces multimodales. In IHM '08: Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine (Metz, France, September 2008). ACM, pp. 215-216 [30].

³⁷ Dumas, B., Lalanne, D., and Ingold, R. 2009. HephaïstTK: a toolkit for rapid prototyping of multimodal interfaces. In ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 2009). ACM, pp. 231-232 [31].

In the case of modeling languages for multimodal interaction, a stepped-up user evaluation, with developers proficient in multimodal interaction, should be organized, in order to assess the real usability and expressiveness of SMUIML. In the case of HephaïsTK and its architecture, the problematic is about the same, with the need for “real” developers to use the framework to create applications, and thus assess the usability and expressiveness of the framework as a whole. As the framework might benefit in the near future from a graphical user interface for the creation of SMUIML scripts, such an evaluation has been for the moment held back, as a complete evaluation of the language would surely benefit of the comparison with a graphical tool; in the same vein, allowing developers to choose their preferred method for the scripting of the framework, between a graphical tool and a scripting language, would help an evaluation better grasp the advantages and shortcomings of the architecture of the HephaïsTK framework itself, independently from the specific features of the SMUIML language. Thus, only qualitative evaluations of those two axes of this thesis were achieved.

The problem with the evaluation of the fusion algorithms is a bit different. A benchmarking tool allowing to evaluate the resistance of a given fusion algorithm in front of tricky multimodal input cases has been proposed in section 5.5. Yet, as of today, no general consensus exists on a set of problematic cases that a “good” fusion algorithm should be able to handle successfully. Thus, the evaluation presented in section 6.2 focuses on a subset of particular problematic cases when considering time synchronicity issues. Of course, this subset is by no means complete, and a real quantitative evaluation of the algorithms should consider a far wider range of tricky multimodal fusion cases. In its present state, this evaluation should be considered as groundwork for comparison of both fusion algorithms present in HephaïsTK.

7 Conclusions & Perspectives

“Tous pour un, un pour tous, c’est notre devise.”
Alexandre Dumas, “Les Trois Mousquetaires”.

7.1	The Three-Headed Hydra of Multimodal Input Fusion	137
7.1.1	The First Head: Software Architecture for Fusion of Input Data.....	138
7.1.2	The Second Head: Multimodal Human-Machine Dialog Modeling	139
7.1.3	The Third Head: Fusion Engine Algorithms	139
7.2	Perspectives & Future Work	140
7.2.1	The Hephaïstos framework: Perspectives.....	140
7.2.2	Multimodal Dialog Modeling: Perspectives.....	141
7.2.3	Fusion Algorithms: Perspectives.....	141
7.3	Wrap Up	142

7.1 The Three-Headed Hydra of Multimodal Input Fusion

This thesis, as highlighted in Chapter 1, focused from the beginning on fusion of multimodal input data. As studies of fusion went on, we realized that focusing only on algorithms, or dialog modeling, or architectures, was a mistake. For the picture of fusion of multimodal input to be complete, one had to consider architectures, modeling and algorithms as a whole. If anything, we consider the main contribution of this thesis to be the demonstration of the way those three themes are deeply intertwined, and have to be considered as such, to be effectively and efficiently handled.

In fact, talking about “intertwined” themes is even an understatement. Architectures, modeling and algorithms are three distinctive parts of a same body; if the reader would allow this Greek

mythology-induced comparison, the developer wishing to effectively tackle the three-headed hydra of fusion of multimodal input data will have to “cut” its three heads, lest the uncut ones make the others grow back and strike at the unsuspecting practitioner.

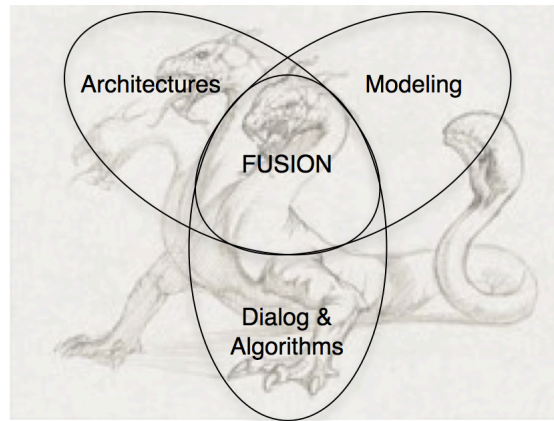


Figure 60. The three-headed hydra of multimodal input fusion management.

7.1.1 The First Head: Software Architecture for Fusion of Input Data

Choosing and defining a valid architecture is the first step toward meticulous management of input data. As Chapter 3 demonstrated, low-level features of a given architecture, such as stream-based or event-driven management of input data, go a long way toward favoring given classes of fusion management. Considering at the earliest stages from findings such as the CASE design space or CARE properties also helps smooth the integration of advanced fusion schemes.

In this thesis, we presented our exploration in architectural solutions targeted at rapid prototyping of multimodal interfaces. Using usability and expressiveness as characterizations of the qualities and drawbacks of said solutions, we demonstrated interest for a dedicated, high-level tool. We proposed a generic architecture for multimodal interaction, integrating all necessary components for management of advanced fusion schemes, as well as fusion-related challenges, such as time synchronicity issues handling. Based on this generic architecture, we proposed the HephaïsTK framework as a tool geared toward rapid prototyping of multimodal interfaces.

A number of applications created with help of HephaïsTK were then presented. On one hand, applications intended as programming tutorials helped demonstrate the accessibility of the programming requirements needed to use the HephaïsTK framework, as well as the clarity of its

approach; on the other hand, a second set of applications seen as proof-of-concepts helped show the capabilities of the HephaistTK framework for the development of multimodal applications integrating advanced concepts.

7.1.2 The Second Head: Multimodal Human-Machine Dialog Modeling

The second of the three themes defining multimodal input data fusion is dialog modeling. As discussed in subsection 5.1.2, dialog and fusion are deeply bound, and multimodal human-machine dialog modeling has to take into account the attributes of fusion in order to be successful. This thesis tackled the issue of multimodal human-machine dialog modeling by first extracting primary features of multimodal dialog modeling languages, following a study of the state of the art in said languages. Among these features, a good proportion of them are directly or indirectly linked to attributes of the fusion engine, thus strengthening our stance.

These features led us to the definition of a set of eight guidelines for the design of description languages for multimodal human-machine dialog modeling. With help of these guidelines, the SMUIML language was designed, as the tool dedicated to modeling multimodal human-machine dialog for HephaistTK. This language was then positioned in relation to other state of the art languages for multimodal dialog modeling.

Finally, the SMUIML language was evaluated with the help of a set of MSc-level students knowledgeable in the field of multimodal interaction. These students were first asked to imagine multimodal applications, and, based on their experience in the design of these applications, define what they would expect from a tool allowing modeling of multimodal dialog. They were then asked to confront their expectations with the SMUIML language, and all of them considered the language as a complete tool which met their expectations.

7.1.3 The Third Head: Fusion Engine Algorithms

At the core of the integration committee in a multimodal application or tool lies a fusion algorithm. This algorithm is tasked with integrating incoming input data and extracting from them a suitable interpretation for client applications. Having discussed in Chapters 3 and 4 a number of characteristics one should encounter in a fusion engine, Chapter 5 was thus dedicated to studies of suitable algorithms for integration of all these characteristics. After discussion of the status of fusion and dialogue, as well as detailing the way a SMUIML dialog modeling script is interpreted in

HephaisTK as a practical illustration of the ties between dialog description and fusion engine, two fusion algorithms implemented in HephaisTK were described. The first one is based on a standard scheme used in multimodal interaction, inspired by meaning frames; the second fusion algorithm is based on Hidden Markov Models (HMMs), with the intent of merging the time modeling capabilities and machine learning properties of HMMs with decision-level fusion of input data. Finally, a benchmarking tool for measuring performance and capabilities of fusion algorithms for multimodal input data was sketched.

This benchmarking tool was then put to use to compare the two fusion algorithms presented in Chapter 5 on the basis of a set of problematic cases, in particular concerning time synchronicity issues. These cases clearly demonstrated the superiority of the HMM fusion algorithm over the “classic” meaning frames algorithm, with or without time constraints specified in the SMUIML script. This should come as no surprise: HMMs have the ability to model time characteristics of input data, whereas this same time modeling in meaning frames is unsatisfactory at best.

7.2 Perspectives & Future Work

The work executed in the course of this thesis was primarily exploratory; in particular, a lot of the topics presented in the former chapters would benefit greatly from consolidation work. In this section, we present perspectives and future work for each of the three axes which constituted the backbone of this thesis: first, we will speak about architectural perspectives, in particular concerning the HephaisTK framework. Then, multimodal dialog modeling perspectives, and finally perspectives on fusion algorithms.

7.2.1 The HephaisTK framework: Perspectives

We have demonstrated that the HephaisTK framework is an effective tool for building multimodal applications. However, while its event-oriented architecture lets it manage advanced fusion schemes with relative ease, the same cannot be said about stream-heavy applications. For example, a relatively classic case of a multimodal application such as the Google Earth exploration application, based on multitouch surfaces, could be built with HephaisTK in its present state, but would suffer from efficiency problems due to the overhead introduced by the multiple event-managing states that are the postman, postman database, integration committee and event notifier agents. This practical

problem raises an interesting software engineering question: would it be possible to define an architecture able to merge both streams and events, thus benefiting from the best of both worlds?

Another future work will be the thorough evaluation of the framework, which could be achieved following two different axes. First, implementation of a number of use cases in a set of state-of-the-art tools for rapid prototyping of multimodal interfaces, so as to be able to further distinguish the strengths and weaknesses of different software architecture choices. Second, creation of a number of multimodal applications by a set of developers not involved in the project, so as to be able to measure more precisely the usability vs. expressiveness trade-off in the HephaïsTK framework. This last evaluation task would however greatly benefit from the future work presented in the multimodal dialog modeling axis.

7.2.2 Multimodal Dialog Modeling: Perspectives

In addition to the eight guidelines presented in section 4.4, we mentioned a ninth, informal, guideline: “readability”. We assume however, that readability, when speaking about a scripting language, is a poor man’s usability feature compared to full-fledged graphical editor tools geared toward scripting of tools for creation of multimodal interfaces. However, SMUIML was designed with this “graphical editor tool” step in mind. We therefore propose that creating a graphical editor tool using SMUIML as its abstract modeling representation, as well as its output, should not only be feasible, but far less complicated than building such a graphical editor tool from scratch. Work has begun on such a graphical editor tool, and we should be able to soon verify whether this is the case or not.

When such a graphical editor tool will be available for generation of SMUIML scripts, it will be interesting to evaluate SMUIML generation from scratch against SMUIML generation using the graphical editor, in particular in terms of usability and expressiveness. A thorough user evaluation of both the language and the graphical tool would thus be welcome. Finally, such a tool would greatly help the evaluation of the HephaïsTK framework, as it would offer a choice of development means to HephaïsTK users.

7.2.3 Fusion Algorithms: Perspectives

Potential perspectives are particularly promising in this axis of the thesis. Indeed, the creation of an HMM-based algorithm for fusion of multimodal input data was a first step towards adaptation of

multimodal fusion to the user. First trials of user-induced error corrections were done, with a simple vibration-based device which would react to hits from the user. This device would then indicate this negative feedback from the user to the fusion engine, which would then try to correct itself, based on the assumption that the last returned result did not please the user. This simple setup already returned encouraging, yet far too preliminary results to be included in this thesis document. Nonetheless, we intend to go on in this research direction, foraging deeper into error management and plasticity issues for multimodal interaction.

Finally, the benchmarking tool presented in section 5.5 will only be useful with a thorough testbed accepted broadly by the multimodal interaction community. Needless to say, laying the groundwork for such a testbed is a challenge that will need to be tackled if one expects research on multimodal fusion algorithms to blossom.

7.3 Wrap Up

Going back to section 1.2 in the introduction listing the challenges of multimodal interaction, we can confirm that Garofolo [40] as well as Oviatt et al. [90] advocate adaptivity to user and context (plasticity) as well as advanced error management, as definite goals for which the multimodal interaction community should strive. Both of these goals require advanced fusion management; indeed, it is our firm belief that, in order to attain these two goals, one has to first lay solid foundation for the whole fusion management process. This solid groundwork can be described as a three-axis body: dedicated architecture, solid dialog modeling and efficient fusion algorithms. This thesis was devoted to the strengthening of this very body, through a study of architectures for multimodal interaction, the definition of eight guidelines for description languages for multimodal dialog modeling, and the creation of a novel fusion algorithm as well as a benchmark for assessment of performances and capabilities of fusion algorithms. Each of these three axes was validated through implementation via, respectively, the HephaisTK framework, the SMUIML language, and the inner Integration Committee of HephaisTK. Qualitative evaluations of each of the three axes were conducted for further validation.

In addition to a better knowledge on fusion underlying mechanisms, we hope that this thesis will have helped clear the way for future work on fusion-related adaptability and error management research.

Bibliography

- [1] Abrilian, S., Martin, J.-C., and Buisine, S. 2003. Algorithms for controlling cooperation between output modalities in 2D embodied conversational agents. In ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces (Vancouver, British Columbia, Canada, 2003). ACM, pp. 293-296.
- [2] Allen, J.F., Perault, C.R. 1980. Analyzing Intentions in Dialogues. *Artificial Intelligence*, 15(3), pp. 143-178 (1980).
- [3] André, E. 2000. The generation of multimedia documents. In: Dale, R., Moisl, H., Somers, H. (eds), *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*, pp. 305-327. Marcel Dekker Inc. (2000).
- [4] Araki, M. and Tachibana, K. 2006. Multimodal Dialog Description Language for Rapid System Development. In Proc. Of the 7th SIGdial Workshop on Discourse and Dialogue (Sydney, Australia, July 2006). pp. 109-116.
- [5] Baddeley, A.D. 1992. Working Memory. *Science*, 255, pp. 556-559 (1992).
- [6] Baddeley, A.D. 1974. Working Memory. In Bower, G.A. (ed.), *Recent advances in learning and motivation*, vol. 8. New York: Academic Press (1974).
- [7] Bass L., Pellegrino R., Reed S., Sheppard S., and Szczur M. 1991. The Arch Model: Seeheim Revisited. In CHI 91 User Interface Developer's Workshop, 1991.
- [8] Bellifemine, F., Caire, G. Poggi, A. and Rimassa, G. 2003. JADE: A While Paper. In EXP magazine, Telecom Italia, Vol. 3, No. 3, September 2003.
- [9] Benoit, C., Martin, J.-C., Pelachaud, C., Schomaker, L., Suhm, B. 2000. Audio-visual and multimodal speech-based systems. In Gibbon, D., Mertins, I., Moore, R. (Eds.), *Handbook of Multimodal and Spoken Dialogue Systems: Resources, Terminology and Product Evaluation*, pp. 102-203, Kluwer (2000).
- [10] Bolt, R.A. 1980. Put-that-there: voice and gesture at the graphics interface. *Computer Graphics*, 14(3), pp. 262-270 (1980).
- [11] Bouchet, J., Madani, L., Nigay, L., Oriat, C., and Parissis, I. 2008. Formal Testing of Multimodal Interactive Systems. In *Engineering interactive Systems: EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, March 22-24, 2007. Selected Papers*, J. Gulliksen, M. B. Harning, P. Palanque, G. C. Veer, and J. Wesson, Eds. *Lecture Notes In Computer Science*, vol. 4940. Springer-Verlag, Berlin, Heidelberg, 36-52.
- [12] Bouchet, J., Nigay, L., and Balzagette, D. 2004. ICARE: a component-based approach for multimodal interaction. In *Proceedings of the 1st French-Speaking Conference on Mobility and Ubiquity Computing (Nice, France, June 01 - 03, 2004). UbiMob '04*, vol. 64. ACM, New York, NY, 36-43.

- [13] Bouchet, J., Nigay, L., and Ganille, T. 2004. ICARE software components for rapidly developing multimodal interfaces. In ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces (State College, PA, USA, 2004). ACM Press, pp. 251-258.
- [14] Bourguet, M.-L. 2002. A Toolkit for Creating and Testing Multimodal Interface Designs. In Companion Proceedings of UIST'02, 15th Annual Symposium on User Interface Software and Technology (Paris, France, October 2002).
- [15] Bourguet, M.-L. 2006. Towards a taxonomy of error-handling strategies in recognition-based multimodal human-computer interfaces. In *Signal Processing*, vol. 86, no. 12, 2006. pp. 3625-3643.
- [16] Brooke, N.M., Petajan, E.D. 1986. Seeing speech: Investigations into the synthesis and recognition of visible speech movements using automatic image processing and computer graphics. In *Proceedings of the International Conference on Speech Input and Output: Techniques and Applications (1986)*, 258, pp. 104-109 (1986).
- [17] Bui T.H. 2006. Multimodal Dialogue Management - State of the Art. CTIT Technical Report series No. 06-01, University of Twente (UT), Enschede, The Netherlands (2006).
- [18] Buxton, W. 1983. Lexical and pragmatic considerations of input structures. In *SIGGRAPH Comput. Graph.*, vol. 17, no. 1, 1983. pp. 31-37.
- [19] Card, S. K., Mackinlay, J. D., and Robertson, G. G. 1991. A morphological analysis of the design space of input devices. In *ACM Trans. Inf. Syst.*, vol. 9, no. 2, 1991. pp. 99-122.
- [20] Churcher, G., Atwell, E., Souter, C. 1997. Dialogue management systems: a survey and overview. 1997.
- [21] Clay, A. 2009. La branche émotion, un modèle conceptuel pour l'intégration de la reconnaissance multimodale d'émotions dans des applications interactives : application au mouvement et à la danse augmentée. PhD Thesis, ESTIA, Université Bordeaux 1. 2009. 204 pages.
- [22] Cohen, P. 1998. Dialogue Modeling. In Cole, R., Mariani, J., Uszkoreit, H., Varile, G.B., Zaenen, A., Zampolli, A.(eds), *Survey of the State of the Art in Human Language Technology*, pp. 204-209, Cambridge University Press (1998).
- [23] Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J. 1997. QuickSet: multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM international Conference on Multimedia*, Seattle, USA, pp. 31-40, (1997).
- [24] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. Young, R. 1995. Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties. In: *Proceedings of INTERACT'95*, Lillehammer, Norway, June 1995, pp. 115-120, Chapman & Hall Publ. (1995).
- [25] De Boeck, J., Vanacken, D., Raymaekers, C., and Coninx, K. 2007. High-Level Modeling of Multimodal Interaction Techniques Using NiMMiT. In *Journal of Virtual Reality and Broadcasting*, vol. 4, no. 2, 2007.
- [26] Dragicevic, P. and Fekete, J.-D. 2004. Support for input adaptability in the ICON toolkit. In ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces (State College, PA, USA, 2004). ACM, pp. 212-219.
- [27] Dumas, B., Ingold, R., and Lalanne, D. 2009. Benchmarking fusion engines of multimodal interactive systems. In ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 2009). ACM, pp. 169-176.
- [28] Dumas, B., Lalanne, D., Guinard, D., Koenig, R., and Ingold, R. 2008. Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction* (Bonn, Germany, February 2008). ACM, pp. 47-54.

-
- [29] Dumas, B., Lalanne, D., and Ingold, R. 2008. Prototyping Multimodal Interfaces with SMUIML Modeling Language. In *Proceedings of CHI 2008 Workshop on UIDLs for Next Generation User Interfaces* (Florence, Italy, April 2008).
 - [30] Dumas, B., Lalanne, D., and Ingold, R. 2008. Démonstration: HephaïstosTK, une boîte à outils pour le prototypage d'interfaces multimodales. In *IHM '08: Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine* (Metz, France, September 2008). ACM, pp. 215-216.
 - [31] Dumas, B., Lalanne, D., and Ingold, R. 2009. HephaïstosTK: a toolkit for rapid prototyping of multimodal interfaces. In *ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces* (Cambridge, Massachusetts, USA, November 2009). ACM, pp. 231-232.
 - [32] Dumas, B., Lalanne, D., and Ingold, R. 2010. Description languages for multimodal interaction: a set of guidelines and its illustration with SMUIML. In *Journal on Multimodal User Interfaces: "Special Issue on The Challenges of Engineering Multimodal Interaction"*, vol. 3, no. 3, 2010. pp. 237-247.
 - [33] Dumas, B., Lalanne, D., Oviatt, S. 2009. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In Denis Lalanne, Jürg Kohlas eds. *Human Machine Interaction*, LNCS 5440, Springer-Verlag, pp. 3-26 (2009).
 - [34] Dupuy-Chessa, S., du Bousquet, L., Bouchet, J., and Ledru, Y. 2006. Test of the ICARE Platform Fusion Mechanism. In *Interactive Systems*, LNCS vol. 3941, 2006. pp. 102-113.
 - [35] Duric, Z., Gray, W., Heishman, R., Li, F., Rosenfeld, A., Schoelles, M., Schunn, C., Wechsler, H. 2002. Integrating perceptual and cognitive modeling for adaptive and intelligent human-computer interaction. In *Proc. of the IEEE*, 90(7), pp. 1272-1289 (2002).
 - [36] EMMA: Extensible MultiModal Annotation markup language: W3C recommandation. <http://www.w3.org/TR/emma/> (accessed 06.08.2010).
 - [37] Flippo, F., Krebs, A., and Marsic, I. 2003. A framework for rapid development of multimodal interfaces. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces* (Vancouver, British Columbia, Canada, 2003). ACM Press, pp. 109-116.
 - [38] Foster, M.E. 2002. State of the art review: Multimodal fusion. COMIC project Deliverable 6.1. September 2002.
 - [39] Fuentes, M., Mostefa, D., Kharroubi, J., Garcia-Salicetti, S., Dorizzi, B., and Chollet, G. 2002. Identity Verification by Fusion of Biometric Data: On-Line Signature and Speech. In *Proc. COST 275 Workshop on The Advent of Biometrics on the Internet* (Rome, Italy, November 2002). pp. 83-86.
 - [40] Garofolo, J. 2008. Overcoming Barriers to Progress in Multimodal Fusion Research. In *Multimedia Information Extraction: Papers from the 2008 AAAI Fall Symposium* (Arlington, Virginia, November 2008). The AAAI Press, pp. 3-4.
 - [41] Gilroy, S. W., Cavazza, M., Chaignon, R., Mäkelä, S.-M., Niranen, M., André, E., Vogt, T., Urbain, J., Seichter, H., Billinghurst, M., and Benayoun, M. 2008. An affective model of user experience for interactive art. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology* (Yokohama, Japan, 2008). ACM, pp. 107-110.
 - [42] Grant, K. W. and Greenberg, S. 2001. Speech intelligibility derived from asynchronous processing of auditory-visual information. In *Workshop on Audio-Visual Speech Processing (AVSP-2001)* (Scheelsminde, Denmark, 2001), pp. 132-137.
 - [43] Greenberg, S. and Fitchett, C. 2001. Phidgets: easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology* (Orlando, Florida, 2001). ACM, pp. 209-218.

- [44] Huang, X., Ariki, Y., and Jack, M. 1990 Hidden Markov Models for Speech Recognition. Columbia University Press.
- [45] Humm, A., Hennebert, J., and Ingold, R. 2009. Combined Handwriting And Speech Modalities For User Authentication. In *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 1, 2009. pp. 25-35.
- [46] Ishii, H. and Ullmer, B. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. In *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems* (Atlanta, Georgia, United States, 1997). ACM Press, pp. 234-241.
- [47] Jacob, R. J., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T., and Zigelbaum, J. 2008. Reality-based interaction: a framework for post-WIMP interfaces. In *Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy, April 05 - 10, 2008). CHI '08. ACM, New York, NY, 201-210.
- [48] Jaimes, A., Sebe, 2007. N. Multimodal human-computer interaction: A survey. In *Computer Vision and Image Understanding* 108, 1-2 (Oct. 2007), Elsevier, pp.116-134 (2007).
- [49] Johnston, M., Cohen, P. R., McGee, D., Oviatt, S. L., Pittman, J. A., and Smith, I. 1997. Unification-based multimodal integration. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics* (Morristown, NJ, USA, 1997). Association for Computational Linguistics, pp. 281-288.
- [50] Johnston, M. and Bangalore, S. 2005. Finite-state multimodal integration and understanding. *Nat. Lang. Eng.* 11, 2 (Jun. 2005), 159-187.
- [51] Kaltenbrunner, M. and Bencina, R. 2007. ReacTIVision: a computer-vision framework for table-based tangible interaction. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction* (Baton Rouge, Louisiana, 2007). ACM, pp. 69-74.
- [52] Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. 2005. TUIO: A protocol for table-top tangible user interfaces. In *Proc. of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005.
- [53] Katsurada, K., Nakamura, Y., Yamada, H., and Nitta, T. 2003. XISL: a language for describing multimodal interaction scenarios. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces* (Vancouver, British Columbia, Canada, 2003). ACM Press, pp. 281-284.
- [54] Klemmer, S. R., Li, J., Lin, J., and Landay, J. A. 2004. Papier-Mache: toolkit support for tangible input. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems* (Vienna, Austria, 2004). ACM Press, pp. 399-406.
- [55] König, W. A., Rädle, R., and Reiterer, H. 2009. Squidy: a zoomable design environment for natural user interfaces. In *CHI '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems* (Boston, MA, USA, 2009). ACM, pp. 4561-4566.
- [56] Koons, D., Sparrell, C., Thorisson, K. 1993. Integrating simultaneous input from speech, gaze, and hand gestures. In M. Maybury (Ed.), *Intelligent Multimedia Interfaces*. Cambridge, MA: MIT Press, pp. 257-276 (1993).
- [57] Krahnstoever, N., Kettebekov, S., Yeasin, M., and Sharma, R. 2002. A Real-Time Framework for Natural Multimodal Interaction with Large Screen Displays. In *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces* (ICMI 2002). pp. 349-354.
- [58] Ladry, J.-F., Navarre, D., and Palanque, P. 2009. Formal description techniques to support the design, construction and evaluation of fusion engines for sure (safe, usable, reliable and evolvable) multimodal interfaces. In *ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces* (Cambridge, Massachusetts, USA, 2009). ACM, pp. 185-192.

-
- [59] Ladry, J.-F., Palanque, P., Basnyat, S., Barboni, E. and Navarre, D. 2008. Dealing with Reliability and Evolvability in Description Techniques for Next Generation User Interfaces. In Proceedings of User interface description languages for next generation user interface workshop at CHI08, Florence, Italy, 2008.
 - [60] Lalanne, D., Evequoz, F., Rigamonti, M., Dumas, B., and Ingold, R. 2007. An ego-centric and tangible approach to meeting indexing and browsing. In Machine Learning for Multimodal Interaction (MLMI' 07) (Brno, Czech Republic, march 2007). Springer Berlin / Heidelberg, pp. 84-95.
 - [61] Lalanne, D., Nigay, L., Palanque, P., Robinson, P., Vanderdonckt, J., and Ladry, J. 2009. Fusion engines for multimodal input: a survey. In Proceedings of the 2009 international Conference on Multimodal interfaces (Cambridge, Massachusetts, USA, November 02 - 04, 2009). ICMI-MLMI '09. ACM, New York, NY, pp. 153-160.
 - [62] Latoschik, M.E., 2002. Designing transition networks for multimodal VR-interactions using a markup language. Multimodal Interfaces, 2002. In Proceedings of the Fourth IEEE International Conference on Multi-modal Interfaces. IEEE, 411-416.
 - [63] Lawson, J.-Y. L., Al-Akkad, A.-A., Vanderdonckt, J., and Macq, B. 2009. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems (Pittsburgh, PA, USA, 2009). ACM, pp. 245-254.
 - [64] Lewin, I. 1998. Formal Design, Verification and Simulation of Multi-modal Dialogues. In TWLT 13: Formal Semantics and Pragmatics of Dialogue; Enschede, Universiteit Twente, Faculteit Informatica, pp. 173-184.
 - [65] McKeown, K. 1985. Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text. Cambridge University Press (1985).
 - [66] McNeill, D. 1992. Hand and Mind: What Gestures Reveal About Thought, Univ. of Chicago Press, Chicago, IL (1992).
 - [67] Mansoux, B., Nigay, L., and Troccaz, J. 2006. Output Multimodal Interaction: The Case of Augmented Surgery. In Proceedings of HCI 2006, Human Computer Interaction, People and Computers XX, The 20th BCS HCI Group conference in co-operation with ACM (London, UK), pages 177-192. 2006.
 - [68] Martin, D., Cheyer, A., Moran, D. 1999. The Open Agent Architecture: A Framework for Building Distributed Software Systems. In Applied Artificial Intelligence, Volume 13, Number 1-2, January-March 1999, pp. 91-128.
 - [69] Martin, J.-C. 1997. TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces. In Intelligence and Multimodality in Multimedia Interfaces, AAAI Press (1997).
 - [70] Mayer, R.E., Moreno, R. 1998. A split-attention effect in multimedia learning: evidence for dual processing systems in working memory. In Journal of Educational Psychology, 1998. 90(2), pp. 312-320 (1998).
 - [71] Melichar, M. 2008. Design of Multimodal Dialogue-Based Systems. PhD Thesis, no. 4081, Lausanne, EPFL, June 2008.
 - [72] Miller, R.B. 1968. Response time in man-computer conversational transactions. Proc. AFIPS Fall Joint Computer Conference Vol. 33, pp. 267-277 (1968).
 - [73] Minsky, M. 1974 *A Framework for Representing Knowledge*. Technical Report. UMI Order Number: AIM-306., Massachusetts Institute of Technology.
 - [74] Moore, J.D. 1995. Participating in Explanatory Dialogues: Interpreting and Responding to Questions in Context. MIT Press, Cambridge, Massachusetts (1995).

-
- [75] Mousavi, S.Y., Low, R., Sweller, J. 1995. Reducing cognitive load by mixing auditory and visual presentation modes. In *Journal of Educational Psychology*, 1995, 87(2), pp. 319-334 (1995).
 - [76] Mugellini, E., Lalanne, D., Dumas, B., Evequoz, F., Gerardi, S., Le Calvé, A., Boder, A., Ingold, R., Abou Khaled, O. 2009. Memmodules as tangible shortcuts to multimedia information. In Denis Lalanne, Jürg Kohlas eds. *Human Machine Interaction*, LNCS 5440, Springer-Verlag, Berlin/Heidelberg, pp. 103-132 (2009).
 - [77] Neal, J.G., Shapiro, S.C. 1991. Intelligent multimedia interface technology. In: Sullivan, J., Tyler, S. (eds.), *Intelligent User Interfaces*, ACM Press, New York, pp. 11-43 (1991).
 - [78] Neal, J. G., Thielman, C. Y., Dobes, Z., Haller, S. M., and Shapiro, S. C. 1989. Natural language with integrated deictic and graphic gestures. In *Proceedings of the Workshop on Speech and Natural Language. Human Language Technology Conference. Association for Computational Linguistics*, Morristown, NJ, 410-423.
 - [79] Nigay, L. 1994. Conception et modélisation logicielles des systèmes interactifs: application aux interfaces multimodales. PhD dissertation, 315 pages.
 - [80] Nigay, L., Coutaz, J.A. 1993. 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, April 24 - 29, 1993). ACM, New York, NY, pp. 172-178 (1993).
 - [81] Nigay, L. and Coutaz, J. 1995. A generic platform for addressing the multimodal challenge. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems* (Denver, Colorado, United States, 1995). ACM Press/Addison-Wesley Publishing Co., pp. 98-105.
 - [82] Nigay, L. and Coutaz, J. 1997. Multifeature Systems: The CARE Properties and Their Impact on Software Design. In *Multimedia Interfaces: Research and Applications*, chapter 9, AAAI Press, 1997.
 - [83] Norman, D.A. 1998. *The Design of Everyday Things*. New York: Basic Book (1988).
 - [84] Novick, D. G., Ward, K. 1993. Mutual Beliefs of Multiple Conversants: A computational model of collaboration in Air Traffic Control. In *Proceedings of AAAI'93*, pp. 196-201 (1993).
 - [85] Oviatt, S.L. 2003. Advances in Robust Multimodal Interface Design. In: *IEEE Computer Graphics and Applications*, vol. 23, september 2003 (2003).
 - [86] Oviatt, S. L. 2006. Human-centered design meets cognitive load theory: designing interfaces that help people think. In *Proceedings of the 14th Annual ACM international Conference on Multimedia* (Santa Barbara, CA, USA, October 23-27, 2006). ACM, New York, NY. pp. 871-880.
 - [87] Oviatt, S.L. 1997. Multimodal interactive maps: Designing for human performance. In: *Human-Computer Interaction*, vol. 12, pp. 93-129 (1997).
 - [88] Oviatt, S.L. 1999. Ten myths of multimodal interaction. In *Communications of the ACM*, 42(11), New York: ACM Press, pp. 74-81 (1999).
 - [89] Oviatt, S.L. 2008. Multimodal interfaces. In J. Jacko, J., Sears, A. (eds), *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, 2nd edition, CRC Press, 2008, chap. 14, pp. 286-304.
 - [90] Oviatt, S. L., Cohen, P. R., Wu, L., Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J., Ferro, D. 2000. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and research directions. In: *Human Computer Interaction*, 2000, vol. 15, no. 4, pp. 263-322 [Reprinted in *Human-Computer Interaction in the New Millennium* (ed. J. Carroll), Addison-Wesley Press, Reading, MA, 2001; chapter 19, pp. 421-456].

-
- [91] Oviatt, S. L., Coulston, R., Tomko, S., Xiao, B., Lunsford, R., Wesson, R. M., and Carmichael, L. 2003. Toward a theory of organized multimodal integration patterns during human-computer interaction. In *Proceedings of the 5th International Conference on Multimodal Interfaces, ICMI 2003* (Vancouver, British Columbia, Canada, november 2003). ACM Press, pp. 44-51.
 - [92] Pan, H., Liang, Z.P., Anastasio, T.J., Huang, T.S. 1999. Exploiting the dependencies in information fusion. In *CVPR*, vol. 2, pp. 407-412 (1999).
 - [93] Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., and Sansone, S. 2008. Authoring pervasive multimodal user interfaces. In *Int. J. Web Eng. Technol.*, vol. 4, no. 2, 2008. pp. 235-261.
 - [94] Petajan, E.D. 1984. Automatic Lipreading to Enhance Speech Recognition, PhD thesis, University of Illinois at Urbana-Champaign (1984).
 - [95] Portillo, P. M., García, G. P., and Carredano, G. A. 2006. Multimodal fusion: a new hybrid strategy for dialogue systems. In *Proceedings of the 8th international Conference on Multimodal interfaces* (Banff, Alberta, Canada, November 02 - 04, 2006). ICMI '06. ACM, New York, NY, 357-363.
 - [96] Rabiner, L. R. 1990. A tutorial on hidden Markov models and selected applications in speech recognition. In *Readings in speech recognition* (Morgan Kaufmann Publishers Inc.), 1990. pp. 267-296.
 - [97] Rabiner, L. and Juang, B. 1993. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc.
 - [98] Reeves, L. M., Lai, J., Larson, J. A., Oviatt, S., Balaji, T. S., Buisine, S. p., Collings, P., Cohen, P., Kraal, B., Martin, J.-C., McTear, M., Raman, T., Stanney, K. M., Su, H. Wang, Q.Y. 2004. Guidelines for multimodal user interface design. In *Communications of the ACM* 47(1), pp. 57-59 (2004).
 - [99] Ross, A. and Jain, A. 2003. Information fusion in biometrics. In *Pattern Recognition Letters*, vol. 24, no. 13, 2003. pp. 2115-2125.
 - [100] Schyn, A., Navarre, D., Palanque, P., and Porcher Nedel, L. 2003. Formal description of a multimodal interaction technique in an immersive virtual reality application. In *IHM 2003: Proceedings of the 15th French-speaking conference on human-computer interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine* (Caen, France, 2003). ACM, pp. 150-157.
 - [101] Serrano, M. 2010. *Interaction multimodale en entrée: Conception et Prototypage*. PhD Thesis. Université Joseph Fourier, Grenoble, France. 2010.
 - [102] Serrano, M., Juras, D., and Nigay, L. 2008. A three-dimensional characterization space of software components for rapidly developing multimodal interfaces. In *ICMI '08: Proceedings of the 10th international conference on Multimodal interfaces* (Chania, Crete, Greece, 2008). ACM, pp. 149-156.
 - [103] Serrano, M. and Nigay, L. 2009. Temporal aspects of CARE-based multimodal fusion: from a fusion mechanism to composition components and WoZ components. In *ICMI-MLMI '09: Proceedings of the 2009 international conference on Multimodal interfaces* (Cambridge, Massachusetts, USA, 2009). ACM, pp. 177-184.
 - [104] Serrano, M., Nigay, L., Lawson, J.-Y. L., Ramsay, A., Murray-Smith, R., and Denef, S. 2008. The openinterface framework: a tool for multimodal interaction. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems* (Florence, Italy, 2008). ACM, pp. 3501-3506.
 - [105] Sharma, R., Pavlovic, V. I., and Huang, T. S. 1998. Toward Multimodal Human-Computer Interface. In *Proceedings of the IEEE*, vol. 86, no. 5, 1998. pp. 853-869.
 - [106] Sinha, A. K. and Landay, J. A. 2003. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces* (Vancouver, British Columbia, Canada, 2003). ACM, pp. 117-124.

-
- [107] Sire, S., Chatty, C. 2002. The Markup Way to Multimodal Toolkits, W3C Multimodal Interaction Workshop, 2002.
 - [108] SSPNet: Social Signal Processing Network, <http://www.sspnet.eu> (accessed 06.08.2010).
 - [109] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., and Montero, F. 2005. A transformational approach for multimodal web user interfaces based on UsiXML. In ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces (Toronto, Italy, 2005). ACM Press, pp. 259-266.
 - [110] Sweller, J., Chandler, P., Tierney, P., Cooper, M. 1990. Cognitive Load as a Factor in the Structuring of Technical Material. In *Journal of Experimental Psychology: General*, 119, pp. 176-192 (1990).
 - [111] Thevenin, D., Coutaz, J. 1999. Plasticity of User Interfaces: Framework and Research Agenda, Proc. of INTERACT'99 (Edinburgh, August 1999), IOS Press.
 - [112] Tindall-Ford, S., Chandler, P., Sweller, J. 1997. When two sensory modes are better than one. In *Journal of Experimental Psychology: Applied*, 3(3), pp. 257-287 (1997).
 - [113] Traum D., Larsson, S. 2003. The Information State Approach to Dialogue Management. In Van Kuppevelt, J.C.J, Smith, R.W. (eds.): *Current and New Directions in Discourse and Dialogue*, pp. 325-353 (2003).
 - [114] Ullmer, B., Ishii, H., and Jacob, R. J. 2005. Token+constraint systems for tangible interaction with digital information. *ACM Trans. Comput.-Hum. Interact.* 12, 1 (Mar. 2005), 81-118.
 - [115] Vernier, F. 2001. La multimodalité en sortie et son application à la visualisation de grandes quantités d'information PhD Thesis. Université Joseph Fourier, Grenoble, France. 2001.
 - [116] Vo, M. T. and Wood, C. 1996. Building an Application Framework for Speech and Pen Input Integration in Multimodal Learning Interfaces. In IEEE Conf. on Acoustics, Speech, and Signal Processing; Proceedings of ICASSP'96 (Atlanta, USA, May 1996). pp. 3545-3548.
 - [117] Wagner, J., Andre, E., and Jung, F. 2009. Smart sensor integration: A framework for multimodal emotion recognition in real-time. In 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops (ACII 2009) (Amsterdam, The Netherlands, sept. 2009). pp. 1-8.
 - [118] Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. 2004. Sphinx-4: a Flexible Open Source Framework for Speech Recognition. Technical Report. UMI Order Number: SERIES13103., Sun Microsystems, Inc.
 - [119] Wellner, P. 1993. Interacting with paper on the DigitalDesk. In *Commun. ACM*, vol. 36, no. 7, 1993. pp. 87-96.
 - [120] Wickens, C. 2002. Multiple resources and performance prediction. In *Theoretical Issues in Ergonomic Science*, 3(2), pp. 159-177 (2002).
 - [121] Wickens, C., Sandry, D., Vidulich, M. 1983. Compatibility and resource competition between modalities of input, central processing, and output. In *Human Factors*, 25(2), pp. 227-248 (1983).
 - [122] Windgrave, C. 2008. Chasm: A Tiered Developer-Inspired 3D Interface Representation. In Proceedings of User interface description languages for next generation user interface workshop at CHI'08, Florence, Italy, 2008.
 - [123] Wu, L., Oviatt, S., and Cohen, P. R. 1999. Multimodal Integration - A Statistical View. In *IEEE Transactions on Multimedia*, vol. 1, no. 4, 1999. pp. 334-341.
 - [124] Wu, L., Oviatt, S. L., and Cohen, P. R. 2002. From members to teams to committee-a robust approach to gestural and multimodal recognition. In *IEEE Trans Neural Netw*, vol. 13, no. 4, 2002. pp. 972-982.

Table of Figures

Figure 1.	Structure of this thesis.	20
Figure 2.	The component structure of Wickens' multiple-resource model of attention (taken from [121]).	30
Figure 3.	Bolt's "Put-that-there" seminal multimodal interface.	31
Figure 4.	Description of the multimodal human machine loop.	34
Figure 5.	The TYCOON Theoretical Framework for studying multimodality (taken from [69]). ..	36
Figure 6.	Different levels of abstraction for a same input source.	37
Figure 7.	The CASE design space (figure based on reference [80]).	37
Figure 8.	CARE properties as a framework to characterize multifeature user interfaces (figure taken from reference [82]).	39
Figure 9.	Examples of purely stream-oriented graphical tools: ICO vs. Squidy.	49
Figure 10.	OpenInterface tools: Skemmi and OIDE.	50
Figure 11.	Typical example of an event-driven graphical tool: the IMBuilder tool.	50
Figure 12.	Canonical architecture of a multimodal interactive system.	54
Figure 13.	Java Swing MM Extension: MMIButton.	56
Figure 14.	A sample automata modeling the interaction flow.	58
Figure 15.	The SCS framework basic architecture.	58
Figure 16.	The Smart Librarian use case.	59
Figure 17.	Basic architecture of the HephaisTK framework.	62
Figure 18.	Example illustrating the operation of HephaisTK toolkit in presence of two inputs events.	64
Figure 19.	Typical startup log in HephaisTK.	66
Figure 20.	Event notification in HephaisTK.	67
Figure 21.	An extract of a XML EMMA script, illustrating how EMMA represents data.	76

Figure 22. An extract of a XML XISL script, illustrating the way XISL represents human-machine dialog.	76
Figure 23. An extract of a XML NiMMiT script, illustrating how NiMMiT handles abstraction layers.	77
Figure 24. Different purposes for a multimodal user interaction description language.	79
Figure 25. Different levels for a multimodal user interaction description language.	80
Figure 26. The three levels of SMUIML.	84
Figure 27. General SMUIML layout.....	85
Figure 28. A typical example of a SMUIML script.....	89
Figure 29. Triggers combination elements in SMUIML.	91
Figure 30. The various levels of multimodal fusion.	96
Figure 31. Roles and relationships of Dialog and Fusion Managers in a generic multimodal architecture.	99
Figure 32. Example of meaning frames merging, in Vo & Wood [116].	102
Figure 33. Example of unification-based fusion algorithm, in Johnston et al. [49].....	103
Figure 34. Associative map and general organization of Member-Team-Committee fusion algorithm, as in Wu et al. [123].	104
Figure 35. Parts of HephaisTK affected by fusion processes.	106
Figure 36. Steps of SMUIML interpretation in relation with fusion management in HephaisTK.....	107
Figure 37. A typical meaning frame in HephaisTK.....	109
Figure 38. Classical “put that there” example expressed in SMUIML.....	111
Figure 39. Meaning frame corresponding to a simple “put that there” example.	111
Figure 40. “Put that there” example, expressed in a slightly more complex way in SMUIML.	112
Figure 41. Meaning frames corresponding to the second “put that there” example.....	113
Figure 42. HMM for the “put that there” example.	114
Figure 43. Finding a match for a sequence of input events.	115
Figure 44. Instantiation and preliminary training of HMM-based fusion in HephaisTK, based on the dialog description in SMUIML.	115
Figure 45. A multimodal fusion engine processes a series of multimodal events to generate a series of interpretations.	117
Figure 46. A ground truth allows to rate performances of the fusion engine, according to the interpretations it gave.	117

Figure 47. Testbed description in EMMA.....	119
Figure 48. Ground truth interpretation part of benchmark specification.....	120
Figure 49. Extract of the <dialog> part of the SMUIML script used for the evaluation.....	126
Figure 50. Meaning frames, no sequential constraints (<par_and>).	127
Figure 51. Meaning frames, sequential constraints (<seq_and>).	128
Figure 52. HMM-based algorithm, no sequential constraints (<par_and>).	129
Figure 53. HMM-based algorithm, sequential constraints (<seq_and>).	129
Figure 54. “Hello World” – Java part of example.	130
Figure 55. Second simple example: a multimodal music player.	131
Figure 56. Condensed Java code corresponding to Music player example.....	132
Figure 57. The XPaint drawing table.....	133
Figure 58. The three modes and different input modalities of XPaint.	134
Figure 59. The Docobro documents management application.	135
Figure 60. The three-headed hydra of multimodal input fusion management.	138

Table of Tables

Table 1.	Differences between GUIs and MUIs.	28
Table 2.	10 myths of multimodal interaction [88].....	32
Table 3.	Architecture traits of current multimodal systems.	46
Table 4.	Architecture traits of current multimodal systems, with works presented in this thesis. ...	71
Table 5.	State of the art multimodal interaction description languages and their features.....	75
Table 6.	Eight guidelines for four purposes.....	82
Table 7.	The SMUIML language and the eight guidelines.	93
Table 8.	Characteristics of fusion levels.....	97

Appendix 1:

XML Schema Definition of SMUIML

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.unifr.ch/smuiml"
xmlns="http://www.unifr.ch/smuiml" elementFormDefault="qualified">
  <xsd:element name="smuiml">
    <xsd:complexType>
      <xsd:sequence maxOccurs="1" minOccurs="1">
        <xsd:element name="integration_description"
          type="integration_description" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="integration_description">
    <xsd:sequence>
      <xsd:element name="recognizers" type="recognizers" />
      <xsd:element name="triggers" type="triggers" />
      <xsd:element name="actions" type="actions" />
      <xsd:element name="dialog" type="dialog" />
    </xsd:sequence>
    <xsd:attribute name="client" type="xsd:string" use="required" />
  </xsd:complexType>
  <xsd:complexType name="recognizers">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1"
        name="recognizer" type="recognizer" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="triggers">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1"
        name="trigger" type="trigger" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="actions">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1"
                name="action" type="action" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="dialog">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1"
                name="context" type="context" />
        </xsd:sequence>
        <xsd:attribute name="leadtime" type="xsd:int" />
    </xsd:complexType>
    <xsd:complexType name="context">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1"
                name="transition" type="transition">
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" />
        </xsd:complexType>
    <xsd:complexType name="transition">
        <xsd:sequence>
            <xsd:choice maxOccurs="1">
                <xsd:element minOccurs="0" name="trigger"
                    type="trigger_a" maxOccurs="1" />
                <xsd:element minOccurs="0" name="par_and"
                    type="par_and" maxOccurs="1" />
                <xsd:element minOccurs="0" name="par_or"
                    type="par_or"/>
                <xsd:element minOccurs="0" name="seq_and"
                    type="seq_and" />
                <xsd:element minOccurs="0" name="seq_or"
                    type="seq_or" />
            </xsd:choice>
            <xsd:element maxOccurs="unbounded" minOccurs="1"
                name="result" type="result" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" />
        <xsd:attribute name="leadtime" type="xsd:int" />
    </xsd:complexType>
```

```

<xsd:complexType name="result">
  <xsd:attribute name="context" type="xsd:anyURI" />
  <xsd:attribute name="action" type="xsd:anyURI" />
</xsd:complexType>
<xsd:complexType name="seq_or">
  <xsd:choice>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element minOccurs="0" name="trigger"
        type="trigger_a" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:element minOccurs="0" name="par_and" type="par_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="par_or" type="par_or"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_and" type="seq_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_or" type="seq_or"
      maxOccurs="unbounded" />
  </xsd:choice>
  <xsd:attribute name="leadtime" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="trigger_a">
  <xsd:attribute name="name" type="xsd:anyURI" />
</xsd:complexType>
<xsd:complexType name="seq_and">
  <xsd:choice>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element minOccurs="0" name="trigger"
        type="trigger_a" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:element minOccurs="0" name="par_and" type="par_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="par_or" type="par_or"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_and" type="seq_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_or" type="seq_or"
      maxOccurs="unbounded" />
  </xsd:choice>
  <xsd:attribute name="leadtime" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="par_or">
  <xsd:choice>

```

```
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
  <xsd:element minOccurs="0" name="trigger"
    type="trigger_a" maxOccurs="unbounded" />
</xsd:sequence>
<xsd:element minOccurs="0" name="par_and" type="par_and"
  maxOccurs="unbounded" />
<xsd:element minOccurs="0" name="par_or" type="par_or"
  maxOccurs="unbounded" />
<xsd:element minOccurs="0" name="seq_and" type="seq_and"
  maxOccurs="unbounded" />
<xsd:element minOccurs="0" name="seq_or" type="seq_or"
  maxOccurs="unbounded" />
</xsd:choice>
<xsd:attribute name="leadtime" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="par_and">
  <xsd:choice>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element minOccurs="0" name="trigger"
        type="trigger_a" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:element minOccurs="0" name="par_and" type="par_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="par_or" type="par_or"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_and" type="seq_and"
      maxOccurs="unbounded" />
    <xsd:element minOccurs="0" name="seq_or" type="seq_or"
      maxOccurs="unbounded" />
  </xsd:choice>
  <xsd:attribute name="leadtime" type="xsd:int" />
</xsd:complexType>
<xsd:complexType name="recognizer">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0"
      name="variable" type="variable" />
    <xsd:element maxOccurs="unbounded" minOccurs="0"
      name="parameter" type="parameter" />
    <xsd:element minOccurs="0" name="translate_value"
      type="translate_value" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="modality" type="xsd:string" />
```

```

</xsd:complexType>
<xsd:complexType name="action">
  <xsd:sequence>
    <xsd:element name="target" type="target" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:complexType name="trigger">
  <xsd:sequence>
    <xsd:element name="source" type="source" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:complexType name="variable">
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="value" type="xsd:string" use="required" />
  <xsd:attribute name="type" type="xsd:string" use="optional" />
</xsd:complexType>
<xsd:complexType name="parameter">
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="value" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:complexType name="translate_value">
  <xsd:attribute name="from" type="xsd:string" use="required" />
  <xsd:attribute name="to" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:complexType name="source">
  <xsd:attribute name="modality" type="xsd:anyURI" />
  <xsd:attribute name="Variable" type="xsd:anyURI"></xsd:attribute>
  <xsd:attribute name="value" type="xsd:string" use="required" />
  <xsd:attribute name="condition" type="xsd:string" />
</xsd:complexType>
<xsd:complexType name="target">
  <xsd:attribute name="name" type="xsd:anyURI" use="required" />
  <xsd:attribute name="message" type="xsd:string" use="required" />
</xsd:complexType>
</xsd:schema>

```


Appendix 2:

Benchmark SMUIML and EMMA Files

SMUIML Configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description client="musicplayer">
    <recognizers>
      <recognizer name="fakespeech" modality="speech">
        <parameter name="emma_file"
value="diuf/diva/hephaistk/benchmark/musicplayer/playnexttrack_emma.xml"
/>
      </recognizer>
      <recognizer name="fakegesture" modality="gesture">
      </recognizer>
      <recognizer name="phidget_interface_kit"
        modality="phidget_ikit">
      </recognizer>
    </recognizers>

    <triggers>
      <trigger name="play_trigger">
        <source modality="speech" value="play | play track" />
      </trigger>
      <trigger name="nexttrack_trigger">
        <source modality="speech" value="next | next track" />
      </trigger>
      <trigger name="track_pointed_event">
        <source modality="gesture" value="track_pointed" />
      </trigger>
    </triggers>

    <actions>
```



```
<action name="play_next_of_point_action">
  <target name="musicplayer"
    message="play_next_of_pointed_track" />
</action>
<action name="play_pointed_track_action">
  <target name="musicplayer" message="play_pointed_track" />
</action>
<action name="next_action">
  <target name="musicplayer" message="next" />
</action>
</actions>

<dialog>
  <context name="start">
    <transition leadtime="1000">
      <par_and>
        <trigger name="track_pointed_event" />
        <trigger name="play_trigger" />
        <trigger name="nexttrack_trigger" />
      </par_and>
      <result action="play_next_of_point_action" />
    </transition>
    <transition leadtime="1000">
      <par_and>
        <trigger name="play_trigger" />
        <trigger name="track_pointed_event" />
      </par_and>
      <result action="play_pointed_track_action" />
    </transition>
    <transition leadtime="1000">
      <trigger name="nexttrack_trigger" />
      <result action="next_action" />
    </transition>
    <transition leadtime="1000">
      <par_and>
        <trigger name="play_trigger" />
        <trigger name="nexttrack_trigger" />
        <trigger name="track_pointed_event" />
      </par_and>
      <result action="next_action" />
    </transition>
  </context>
</dialog>
```

```

    </integration_description>
</smuiml>

```

EMMA Configuration file for the testbed:

```

<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2003/04/emma
    http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
  xmlns="http://www.example.com/example">
  <emma:one-of id="main">
    <emma:sequence id="testbed">

      <!-- <pointing> "Play next track" -->
      <emma:interpretation id="gesture1" emma:medium="tactile"
        emma:mode="video">
        <content>track_pointed</content>
      </emma:interpretation>
      <emma:interpretation id="speech1" emma:medium="acoustic"
        emma:mode="voice"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="300">
        <content>play</content>
      </emma:interpretation>
      <emma:interpretation id="speech2" emma:medium="acoustic"
        emma:mode="voice"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="400">
        <content>next track</content>
      </emma:interpretation>

      <!-- "Play" <pointing> "next track" -->
      <emma:interpretation id="speech3" emma:medium="acoustic"
        emma:mode="voice"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="3000">
        <content>play</content>
      </emma:interpretation>
      <emma:interpretation id="gesture2" emma:medium="tactile"
        emma:mode="video"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="3250">
        <content>track_pointed</content>
      </emma:interpretation>
      <emma:interpretation id="speech4" emma:medium="acoustic"
        emma:mode="voice"

```

```
        emma:time-ref-uri="#gesture1" emma:offset-to-start="3500">
        <content>next track</content>
    </emma:interpretation>

    <!-- "Play next track" <pointing> -->
    <emma:interpretation id="speech5" emma:medium="acoustic"
        emma:mode="voice"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="6000">
        <content>play</content>
    </emma:interpretation>
    <emma:interpretation id="speech6" emma:medium="acoustic"
        emma:mode="voice"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="6100">
        <content>next track</content>
    </emma:interpretation>
    <emma:interpretation id="gesture3" emma:medium="tactile"
        emma:mode="video"
        emma:time-ref-uri="#gesture1" emma:offset-to-start="6400">
        <content>track_pointed</content>
    </emma:interpretation>
</emma:sequence>

<emma:sequence id="groundtruth">

    <!-- <pointing> "Play next track" -->
    <emma:interpretation id="message1">
        <emma:derived-from resource="#gesture1" composite="true"/>
        <emma:derived-from resource="#speech1" composite="true"/>
        <emma:derived-from resource="#speech2" composite="true"/>
        <message>play_next_of_pointed_track</message>
    </emma:interpretation>

    <!-- "Play" <pointing> "next track" -->
    <emma:interpretation id="message2">
        <emma:derived-from resource="#speech3" composite="true"/>
        <emma:derived-from resource="#gesture2" composite="true"/>
        <message>play_pointed_track</message>
    </emma:interpretation>
    <emma:interpretation id="message3">
        <emma:derived-from resource="#speech4" composite="false"/>
        <message>next</message>
    </emma:interpretation>
```

```
<!-- "Play next track" <pointing> -->
<emma:interpretation id="message4">
  <emma:derived-from resource="#speech5" composite="true"/>
  <emma:derived-from resource="#speech6" composite="true"/>
  <emma:derived-from resource="#gesture3" composite="true"/>
  <message>next</message>
</emma:interpretation>
</emma:sequence>
</emma:one-of>
</emma:emma>
```


Appendix 3:

Multimodal Music Player Example: SMUIML and Java Files

Multimodal Music Player Example: SMUIML configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description client="musicplayer">
    <recognizers>
      <recognizer name="speech" modality="speech">
        <parameter name="lang" value="fr"/>
      </recognizer>
      <recognizer name="phidgetrfid" modality="rfid"/>
    </recognizers>
    <triggers>
      <trigger name="play_trigger">
        <source modality="speech" value="lecture"/>
      </trigger>
      <trigger name="pause_trigger">
        <source modality="speech" value="pause"/>
      </trigger>
      <trigger name="previous_trigger">
        <source modality="speech" value="précédent |
          précédente piste"/>
      </trigger>
      <trigger name="next_trigger">
        <source modality="speech" value="prochain |
          prochaine piste"/>
      </trigger>
      <trigger name="rfid_trigger">
```

```
        <source modality="rfid" value="_ANY"
              condition="valid"/>
    </trigger>
</triggers>
<actions>
    <action name="play_action">
        <target name="musicplayer" message="play"/>
    </action>
    <action name="pause_action">
        <target name="musicplayer" message="pause"/>
    </action>
    <action name="previous_action">
        <target name="musicplayer" message="previous"/>
    </action>
    <action name="next_action">
        <target name="musicplayer" message="next"/>
    </action>
    <action name="rfid_action">
        <target name="musicplayer" message="rfid"/>
    </action>
</actions>
<dialog>
    <context name="start">
        <transition>
            <trigger name="play_trigger"/>
            <result action="play_action"/>
        </transition>
        <transition>
            <trigger name="pause_trigger"/>
            <result action="pause_action"/>
        </transition>
        <transition>
            <trigger name="previous_trigger"/>
            <result action="previous_action"/>
        </transition>
        <transition>
            <trigger name="next_trigger"/>
            <result action="next_action"/>
        </transition>
        <transition>
            <trigger name="rfid_trigger"/>
            <result action="rfid_action"/>
        </transition>
    </context>
</dialog>
```

```

        </context>
    </dialog>
</integration_description>
</smuiml>

```

Corresponding Java file:

```

package diuf.diva.hephaistk.demo.musicplayer;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

import diuf.diva.hephaistk.config.HephaistkInitManager;
import diuf.diva.hephaistk.dialog.Message;
import diuf.diva.hephaistk.events.MultimodalInputFusedEvent;
import diuf.diva.hephaistk.events.MultimodalInputFusedEventListener;

public class HephaistTKMusicPlayer extends JFrame implements
MultimodalInputFusedEventListener, ActionListener{
    private static final long serialVersionUID = 2089067929312863959L;
    private static final String SMUIML_FILE_NAME =
"musicplayerconfig_fr.xml";
    private static final String SE =
System.getProperty("file.separator");
    private static final String SMUIML_FILE_PATH =
System.getProperty("user.dir")+SE+"diuf"+SE+"diva"+SE+"hephaistk"+SE+"demo
"+SE+"musicplayer"+SE+SMUIML_FILE_NAME;

    private JPanel bigPanel, littlePanel = null;
    private JButton previous, play, pause, next = null;
    private JTextField infoField = null;

    public HephaistTKMusicPlayer(){

        //here comes instanciation of GUI window

```



```
bigPanel = new JPanel();
bigPanel.setLayout(new BoxLayout(bigPanel, BoxLayout.Y_AXIS));

littlePanel = new JPanel();
littlePanel.setLayout(new BoxLayout(littlePanel,
    BoxLayout.X_AXIS));

previous = new JButton("previous");
previous.addActionListener(this);
littlePanel.add(previous);

play = new JButton("play");
play.addActionListener(this);
littlePanel.add(play);

pause = new JButton("pause");
pause.addActionListener(this);
littlePanel.add(pause);

next = new JButton("next");
next.addActionListener(this);
littlePanel.add(next);

bigPanel.add(littlePanel);

infoField = new JTextField();
bigPanel.add(infoField);

add(bigPanel);

//here comes instanciacion of HephaistK
HephaistkInitManager hephaisManager = new
    HephaistkInitManager("musicplayer");
hephaisManager.addSMUIMLConfigFile(SMUIML_FILE_PATH);
hephaisManager.startFramework();

//declaration of class as listeners implementer
hephaisManager.addMultimodalInputFusedEventListener(this);

//final steps of GUI instanciacion
pack();
setVisible(true);
}
```

```

    public static void main(String args[]){
        new HephaistTKMusicPlayer();

    }

    //as this class implements MultimodalInputFusedEventListener,
    //we have to override this class
    //@Override
    public void fusedEvent(MultimodalInputFusedEvent event) {
        //'events' are messages such as declared in the <actions>
        //part of your SMUIML script
        for(Message message : event.getMessages()){
            if(message.getMessage().equals("previous")){
                previous();
            }else if(message.getMessage().equals("play")){
                play();
            }else if(message.getMessage().equals("pause")){
                pause();
            }else if(message.getMessage().equals("next")){
                next();
            }else{
                System.out.println("received new message !!!!
                \""+message.getMessage()+"
                "+message.getNrOfVariables()+" variable(s)");
                if(message.getNrOfVariables() > 0){
                    for(int i = 0; i < message.getNrOfVariables(); i++){
                        System.out.println("variable no. "+i+":
                        '"+message.getVariableName(i)+"' of class
                        "+message.getVariableClass(i)+"; value:
                        "+message.getVariableValue(i));
                    }
                }
            }
        }
    }

    //as this class implements ActionListener,
    //we have to override this class
    //@Override
    public void actionPerformed(ActionEvent arg0) {
        JButton src = (JButton)(arg0.getSource());
        if(src.getText().equals("previous")){

```

```
        previous();
    }else if(src.getText().equals("play")){
        play();
    }else if(src.getText().equals("pause")){
        pause();
    }else if(src.getText().equals("next")){
        next();
    }
}

private void play(){
    infoField.setText("playing...");
}

private void pause(){
    infoField.setText("on pause");
}

private void previous(){
    infoField.setText("previous track");
}

private void next(){
    infoField.setText("next track");
}
}
```

Appendix 4:

XPaint Drawing Table SMUIML

Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<smuiml>
  <integration_description client="xpaint_client">

    <recognizers>
      <recognizer name="speech" modality="speech"/>
      <recognizer name="reactivision" modality="reactivision">
        <variable name="posx" value="xpos"/>
        <variable name="posy" value="ypos"/>
        <variable name="fiducial" value="sourceId"/>
      </recognizer>
      <recognizer name="phidgetrfid" modality="rfid">
        <variable name="shape" value="source"/>
        <variable name="oper" value="source"/>
        <translate_value from="2342111" to="filled circle"/>
        <!--other values of tags follow. -->
        <variable name="rfid_value" value="tagID"/>
      </recognizer>
      <recognizer name="phidget_ikit" modality="phidget_ikit"/>
      <recognizer name="xpaint_client" modality="xpaint_client">
        <variable name="background_selected" value="data"
          type="string"/> <!-- boolean-->
      </recognizer>
    </recognizers>

    <triggers>
      <trigger name="select shape">
        <source modality="speech" value="select shape"/>
      </trigger>
    </triggers>
  </integration_description>
</smuiml>
```

```
</trigger>
<trigger name="return">
  <source modality="speech" value="return"/>
</trigger>
<trigger name="begin draw">
  <source modality="speech" value="begin draw"/>
</trigger>
<trigger name="end draw">
  <source modality="speech" value="end draw"/>
</trigger>
<trigger name="select background">
  <source modality="speech" value="select background"/>
</trigger>
<trigger name="operation">
  <source modality="speech" value="erase shape |
    rotate shape | move shape"/>
</trigger>
<trigger name="selected shape">
  <source modality="xpaint_client"
    value="$shape_selected"/>
</trigger>
<trigger name="position">
  <source variable="fiducial" value="1 | 2"
    condition="valid"/>
</trigger>
<trigger name="position1">
  <source variable="fiducial" value="1"
    condition="valid"/>
</trigger>
<trigger name="position2">
  <source variable="fiducial" value="2"
    condition="valid"/>
</trigger>
<trigger name="tools_one_hand">
  <source modality="rfid" value="select | line |
    freehand"/>
</trigger>
<trigger name="tools_two_hands">
  <source modality="rfid" value="filled circle |
    empty circle | filled rectangle | empty rectangle
    | filled polygon | empty polygon"/>
</trigger>
<trigger name="color">
```

```

        <source modality="phidget_ikit" condition="valid"/>
    </trigger>
    <trigger name="thickness">
        <source modality="phidget_ikit" condition="valid"/>
    </trigger>
    <trigger name="background_type">
        <source variable="fiducial" value="1 | 2"
            condition="valid"/>
    </trigger>
    <trigger name="background_selected">
        <source variable="data" value="$background_selected"/>
    </trigger>
</triggers>

<actions>
    <action name="draw_operation">
        <target name="xpaint_client" message="draw $oper
            $shape $posx $posy"/>
    </action>
    <action name="selection_of_background">
        <target name="xpaint_client" message="background
            selected"/>
    </action>
    <action name="modif_operation">
        <target name="xpaint_client" message="modify $oper
            $shape $posx $posy"/>
    </action>
</actions>

<dialog leadtime="2100">
    <context name="start">
        <transition leadtime="2200">
            <trigger name="select shape"/>
            <result context="modification"/>
        </transition>
        <transition>
            <trigger name="begin draw"/>
            <result context="drawing"/>
        </transition>
        <transition>
            <trigger name="select background"/>
            <result context="background"/>
        </transition>
    </context>
</dialog>

```

```
</context>
```

```
<context name="drawing">
  <transition name="drawing_one_hand">
    <par_and> <!-- others : par_or, seq_and, seq_or -->
      <trigger name="tools_one_hand"/>
      <trigger name="color"/>
      <trigger name="thickness"/>
      <trigger name="position"/>
    </par_and>
    <result action="draw_operation"/>
  </transition>
  <transition name="drawing_two_hands">
    <par_and>
      <trigger name="tools_two_hands"/>
      <trigger name="color"/>
      <trigger name="thickness"/>
      <trigger name="position1"/>
      <trigger name="position2"/>
    </par_and>
    <result action="draw_operation"/>
  </transition>
  <transition>
    <trigger name="end draw"/>
    <result context="start"/>
  </transition>
</context>
```

```
<context name="background">
  <transition name="background_frame">
    <trigger name="background_type"/>
    <result action="selection_of_background"/>
    <result context="start"/>
  </transition>
</context>
```

```
</dialog>
```

```
</integration_description>
```

```
</smuiml>
```