The SEMAINE API

—

A Component Integration Framework for a Naturally Interacting and Emotionally Competent Embodied Conversational Agent

Dissertation

zur Erlangung des Grades des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Marc Schröder

Saarbrücken, 2011

Tag des Kolloquiums: 21.12.2011

Dekan der Fakultät: Prof. Dr.-Ing. Holger Hermanns

Mitglieder des Prüfungsausschusses:

- Prof. Dr. Dietrich Klakow (Vorsitzender)
- Prof. Dr. Hans Uszkoreit (Erstgutachter)
- Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster (Zweitgutachter)
- Dr. Patrick Gebhard (Beisitzer)

Kurze Zusammenfassung

Die vorliegende Dissertation behandelt das Thema der virtuellen Agenten mit Fähigkeiten zur natürlichen Benutzer-Interaktion sowie emotionaler Kompetenz bzgl. der Wahrnehmung und Generierung emotionalen Ausdrucks. Der Schwerpunkt der Arbeit liegt auf den technologischen Grundlagen für die Implementierung eines echtzeitfähigen Systems mit diesen Fähigkeiten, das aus wiederverwendbaren Komponenten erstellt werden kann. Die Arbeit umfasst drei Kernaspekte. Zum Einen beschreibt sie ein neues Framework zur Komponenten-Integration, die SEMAINE API: Diese erleichtert die Erstellung von Emotions-orientierten Systemen aus Komponenten, die untereinander mittels Standard- oder Prä-Standard-Repräsentationen kommunizieren. Zweitens wird eine Systemarchitektur vorgestellt, welche Vorbereitung und Auslösung von Systemverhalten entkoppelt und so zu einer substanziellen Beschleunigung der Generierungszeit führt, wenn Systemäußerungen im Voraus geplant werden können. Drittens beschreibt die Arbeit die W3C Emotion Markup Language, einen werdenden Web-Standard zur Repräsentation von Emotionen in technologischen Systemen. Es werden kritische Aspekte der Systemperformanz untersucht, wodurch gezeigt wird, dass das Framework eine gute Basis für die Implementierung echtzeitfähiger interaktiver Agentensysteme darstellt. Anhand von drei Beispielen wird illustriert, dass mit der SEMAINE API leicht neue Emotions-orientierte Systeme aus neuen und existierenden Komponenten erstellt werden können.

Short Summary

The present thesis addresses the topic area of Embodied Conversational Agents (ECAs) with capabilities for natural interaction with a human user and emotional competence with respect to the perception and generation of emotional expressivity. The focus is on the technological underpinnings that facilitate the implementation of a real-time system with these capabilities, built from re-usable components. The thesis comprises three main contributions. First, it describes a new component integration framework, the SEMAINE API, which makes it easy to build emotion-oriented systems from components which interact with one another using standard and pre-standard XML representations. Second, it presents a prepare-and-trigger system architecture which substantially speeds up the time to animation for system utterances that can be pre-planned. Third, it reports on the W3C Emotion Markup Language, an upcoming web standard for representing emotions in technological systems. We assess critical aspects of system performance, showing that the framework provides a good basis for implementing real-time interactive ECA systems, and illustrate by means of three examples that the SEMAINE API makes it is easy to build new emotion-oriented systems from new and existing components.

Contents

Acknowledgements

Writing a PhD thesis is tough, writing a *second* one is crazy – as many people have pointed out to me in recent years. They were of course right. I have done it nevertheless, and I must say that writing gets easier over the years, but time to do it gets scarcer even more. So my first thanks go to my family who have supported me in sparing off family time and sitting in front of a computer instead, typing.

Second, I would like to thank the entire SEMAINE consortium for the excellent collaboration, notably Björn Schuller, Florian Eyben, Martin Wöllmer (München), Catherine Pelachaud, Elisabetta Bevacqua, Etienne de Sevin (Paris), Dirk Heylen, Mark ter Maat (Twente), Maja Pantic, Michel Valstar, Hatice Gunes (London), Roddy Cowie, Gary McKeown (Belfast), and Sathish Pammi (DFKI). Your positive and constructive attitude has made the double task of coordinator and WP leader for system integration manageable.

Furthermore, I would like to thank the members of the Emotion Incubator groups and the Multimodal Interaction working group at the W3C, in particular the following people: Paolo Baggia and Enrico Zovato (Loquendo, Turin), Felix Burkhardt (Deutsche Telekom Laboratories, Berlin), Christian Peter (Fraunhofer IGD-R, Rostock), and Catherine Pelachaud (ParisTech). You have shown true perseverence, in an endeavour whose long-term nature only very few of us foresaw when we started.

Special thanks to Catherine Pelachaud for suggestions on how to improve an earlier draft version of the ECA state-of-the-art chapter.

Thanks are also due to my colleagues at DFKI who have kindly provided relevant suggestions for structuring the thesis, notably Hendrik Zender, Günter Neumann, and Hans-Ulrich Krieger.

Last but not least, I would like to thank my supervisor Hans Uszkoreit for encouraging me to pursue this PhD in computer science, as well as for his helpful comments and suggestions regarding an earlier draft version of this thesis.

Chapter 1

Introduction

Making the interaction with computers more natural for humans requires computers to acquire multiple capabilities. Alongside many others, these capabilities include aspects of communication that are emotion-related and non-verbal. This thesis is part of a sustained effort to bring together, in one consistent framework, many of the technologies required to endow a computer with such capabilities, and describes how these technologies are put to use to implement a specific type of dialogue system: a fully autonomous implementation of 'Sensitive Artificial Listeners' (SAL).

We consider a one-to-one dialogue situation where, at any given time, one user is having a conversation with one Embodied Conversational Agent (ECA) character. The interaction is multimodal, involving speech, head movements, and facial expressions. One of the key features of human interactions that we expect to reproduce is that the dialogue will involve some emotional colouring – not in terms of episodes of intense "basic" emotions, but in the sense of emotion "pervading everyday life" (Cowie, 2010).

The goal of having a natural conversation in this setting sets technology a number of substantial challenges. The computer must be able to perceive the user's verbal and non-verbal behaviour, i.e. have some awareness of the words spoken, the prosody with which they are spoken, and the user's head movements and facial expressions. While the user is talking, the computer must exhibit suitable *listener behaviour* – notably multimodal *backchannels* (Yngve, 1970) which signal that the listener is still present and following what is being said, and which may at the

same time provide *feedback* to the speaker about the listener's reaction (Allwood et al., 1992). Examples of such listener behaviour are head nods, smiles, or short vocalisations such as "uh-huh" or "wow", which may be produced individually or in combination. The computer must determine when is a good moment to *take the turn* (Sacks et al., 1974) and become the speaker itself; it must then produce a verbal utterance which must fit the dialogue context, and which has to be spoken with a suitable voice and synchronous facial animation, including lip movements, facial expressions and head movements. In doing all of this, it must be aware of any emotional colouring in the user's behaviour, and react appropriately. If any of these processes is not performed, does not have the right timing, or in some other way does not match the user's sense of natural behaviour, the quality of the interaction is degraded.

It is important to notice that the requirements formulated above are quite distinct from the various requirements arising from the task-oriented interactions that are often studied. For example, an accurate interpretation of the user's words depends on high-quality Automatic Speech Recognition (ASR) (Jurafsky et al., 2000); efficiently achieving a dialogue goal depends on suitable dialogue structures and on modelling task domains (Allen et al., 2001b); and having a common experiential basis for an interaction requires grounding of a machine's knowledge about the world (Kruijff et al., 2007). We recognise the importance of these goals, but they are not the goals that we address in this work. They represent one of the streams that need to converge to produce a competent artificial interactant; our work represents another, which has received much less attention in the computational community, but which psychological research suggests is at least equally important (Mehrabian and Ferris, 1967; Birdwhistell, 1970).

The scenario that we call "Sensitive Artificial Listeners" was developed specifically to let us concentrate on the emotional and non-verbal competences involved in conversation. Its raison d'être is to avoid the difficulties of task-oriented dialogues and instead address directly the emotion-related and non-verbal capabilities that a system needs in order to have a naturally flowing conversation. A good deal of experience indicates that the two aspects can operate rather independently (for example, when a party is too noisy for people to hear most of each other's words, and yet interaction flourishes). If so, it makes sense to expect that capabilities developed in the SAL scenario can later be integrated into task-oriented interactions.

Since an ECA-based interactive dialogue system requires multiple input and output processing functionalities, it is typically built from components. Each component provides some specialised functionality; components need to communicate representations of their processing results to one another. At the most coarse-grained level, three building blocks of dialogue systems are usually distinguished: (1) detection and analysis of user behaviour; (2) interpretation, deliberation and planning; and (3) generation of system behaviour. Each of these blocks typically has a substructure. For example, we can expect to require individual components for detecting the emotions and non-verbal behaviour from the voice and from the face of the user; we may want to separate the generation of the ECA's behaviour while speaking from the generation of its listener behaviour; and we will require separate components for generating the synthetic voice and the visual behaviour of the ECA. In order to provide a consistent system behaviour, these components need to communicate with one another using jointly understood representations.

The effort needed for system integration is often underestimated (Thórisson et al., 2004; Herzog et al., 2004). The naive assumption that it is sufficient to bring components with the required functionality together is likely to lead to problems – not at the beginning of the work, but at a later project stage. When integration is required, for example to build a demonstrator system, issues related to, e.g., inter-component communication, coordination, or traceability are likely to surface. At that stage, usually under time pressure, these problems will be difficult to resolve in a clean way. Making integration of software components a well-organised and high priority task can avoid or at least reduce these problems.

Whereas each specific ECA system will be somewhat different from others, many of the functionalities required to build them are actually the same. The burden on integrating components could be somewhat lifted if clear conventions for their interfaces were defined. Specifically, if the representations used for various processing results followed some *standard* format, it would be much easier to reuse components when building new systems.

The present thesis reports on an effort to integrate the components needed for a naturally interacting and emotionally competent ECA into one software architecture, in order to provide the building blocks needed for studying natural one-to-one conversations between an ECA and a human user. The specific emphasis of the work lies on the component integration framework as such, as well as on the rep-

resentations used for inter-component communication – notably, a standard for representing emotions.

1.1 The SEMAINE project

This thesis has been carried out in the context of the EU-funded project "SEMAINE: Sustained Emotionally coloured Machine-human Interaction using Nonverbal Expression" (2008–2010)[1]. The project has built a "Sensitive Artificial Listener" (SAL) system: a multimodal dialogue system with an emphasis on nonverbal skills – detecting and emitting vocal and facial signs of emotion and signs related to the interaction, such as backchannel signals, in order to register and express information such as continued presence, attention or interest, an evaluation of the content, or an emotional connotation. The system has strong real-time constraints, because it must react to the user's behaviour while the user is still speaking.

The project's approach is strongly oriented towards making basic technology for emotion-oriented interactive systems available to the research community, where possible as open source. While the primary goal is to build a system that can engage a human user in a conversation in a plausible way, it is also an important aim to provide high-quality audiovisual recordings of human-machine interactions, as well as software components that can be reused by the research community.

1.2 Contribution of the present thesis

In order to build a system with the capabilities described above, a framework is needed for integrating the different components into a coherent system. We can distinguish two key requirements for this work.

1. **Infrastructure:** Components written in different programming languages and running on different operating systems must work together to provide the requested functionality. A suitable integration framework and system

[1] `http://www.semaine-project.eu`

architecture must be set up to enable the real-time constraints imposed by the targeted system functionality.

2. **Communication:** Suitable representation formats are needed to enable system components to communicate to one another the concepts that are important for a natural interaction between the ECA and the human. Where possible, standard representation formats should be used to facilitate the future reuse of components and subsystems. Of particular importance in the context of emotion-oriented systems is a computer-readable representation of emotion.

The present thesis describes how we have addressed these requirements. As a solution to the infrastructure requirements, we have created the SEMAINE API, a component integration framework based on the principles of an asynchronous messaging middleware. In order to enable communication in a re-usable way, we have enabled support for a number of relevant representation languages. In particular, we have created a specification for an Emotion Markup Language.

We assess these solutions in terms of the performance obtained and in terms of ease of re-use of the SEMAINE API for building new emotion-oriented systems.

The results of the present thesis have been used for system integration in the SEMAINE project, thus providing the integration and communication backbone for the system components provided by the SEMAINE partners.

1.3 Thematic delimitation of the present thesis

The topic of the thesis is the technological framework underpinning a naturally interacting and emotionally competent interactive system.

The emphasis hereby is on the *infrastructure-level technology* needed to endow an interactive system with natural and emotion-related behaviours.

Obviously there are numerous other important aspects needed in order to realise a system with the required capabilities. There need to be components that can analyse expressive behaviour from the human user's face and voice; components to decide whether the system should speak or listen; components to decide which behaviour to perform; and components capable of rendering the intended

behaviour through face, voice, and gestures. Furthermore, there needs to be an evidential basis for determining the appropriate system behaviour. Corpora need to be collected and annotated, machine learning algorithms need to be trained, rule-based algorithms need to be tuned, etc. Finally, the naturalness of the behaviour of any given interactive system needs to be evaluated in order to verify the effectiveness of the implemented behaviour.

All of these aspects of system design, training, tuning and evaluation are undoubtedly necessary to create an interactive system that can claim to be emotionally competent or interacting naturally. In fact, all of these steps have been carried out in the SEMAINE project by various project partners.

In the context of the present thesis, however, these aspects are considered peripheral and out of scope of the intended work. For our component integration framework, the aim is to provide proper integration of the components needed for this type of system. We describe the components of the SEMAINE system only briefly, in order to illustrate the difference between the two system architectures we provide (Chapter 7). Similarly, we do not carry out an interactive system evaluation, since it would not allow us to verify whether the requirements identified above (see Section 1.2) have been met. Instead, we assess the framework in terms of its *performance* as measurable in milliseconds, and its *ease of use* in terms of the steps needed to build different emotion-oriented systems on the basis of the framework.

1.4 Publications

The work presented in this thesis is based in parts on the following scientific and technical publications.

Journal papers

- Schröder, M., Bevacqua, E. Cowie, R., Eyben, F., Gunes, H., Heylen, D., ter Maat, M., McKeown, G., Pammi, S., Pantic, M., Pelachaud, C., Schuller, B., de Sevin, E., Valstar, M., and Wöllmer, M. (submitted): Building Autonomous Sensitive Artificial Listeners. Submitted to IEEE Transactions on Affective Computing.

- Schröder, M. (2010). The SEMAINE API: Towards a standards-based framework for building emotion-oriented systems. Advances in Human-Machine Interaction, Volume 2010, Article ID 319406. doi:10.1155/2010/319406.

Book chapters

- Schröder, M., Pirker, H., Lamolle, M., Burkhardt, F., Peter, C., and Zovato, E. (2011). Representing emotions and related states in technological systems. In P. Petta, R. Cowie, and C. Pelachaud (Eds.), Emotion-Oriented Systems – The Humaine Handbook (pp. 367-386). Springer.

Conference and workshop proceedings papers

- Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., and Zovato, E. (submitted): EmotionML – an upcoming standard for representing emotions and related states. Submitted to Affective Computing and Intelligent Interaction, Memphis, TE, USA.

- Schröder, M., and McKeown, G. (2010). Considering Social and Emotional Artificial Intelligence. Proc. AISB 2010 Symposium "Towards a Comprehensive Intelligence Test", Leicester, UK.

- Schröder, M., Wilson, I., Jarrold, W., Evans, D., Pelachaud, C., Zovato, E. and Karpouzis, K. (2008). What is most important for an Emotion Markup Language? Proc. Third Workshop Emotion and Computing, KI 2008, Kaiserslautern, Germany.

- Schröder, M., Devillers, L., Karpouzis, K., Martin, J.-C., Pelachaud, C., Peter, C., Pirker, H., Schuller, B., Tao, J. and Wilson, I. (2007). What should a generic emotion markup language be able to represent? Proc. 2nd International Conference on Affective Computing and Intelligent Interaction (ACII'2007), Lisbon, Portugal.

- Schröder, M., Pirker, H. and Lamolle, M. (2006). First suggestions for an Emotion Annotation and Representation Language, LREC 2006 workshop on Emotion: Corpora for Research on Emotion and Affect, Genoa, Italy.

Technical reports

- Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., and Zovato, E. (2011). Emotion Markup Language (EmotionML) 1.0. W3C Last Call Working Draft, World Wide Web Consortium. http://www.w3.org/TR/2011/WD-emotionml-20110407/

- Schröder, M., Pelachaud, C., Ashimura, K., Baggia, P., Burkhardt, F., Oltramari, A., Peter, C., et al. (2011). Vocabularies for EmotionML. W3C Working Draft, World Wide Web Consortium. http://www.w3.org/TR/2011/WD-emotion-voc-20110407/

- Schröder, M., Baggia, P., Burkhardt, F., Oltramari, A., Pelachaud, C., Peter, C., and Zovato, E. (2010). Emotion Markup Language (EmotionML) 1.0. W3C Working Draft, World Wide Web Consortium. http://www.w3.org/TR/2010/WD-emotionml-20100729/

- Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., and Zovato, E. (2009). Emotion Markup Language (EmotionML) 1.0. W3C First Public Working Draft, World Wide Web Consortium. http://www.w3.org/TR/2009/WD-emotionml-20091029/

- Schröder, M., Baggia, P., Burkhardt, F., Martin, J., Pelachaud, C., Peter, C., Schuller, B., Wilson, I. and Zovato, E. (2008). Elements of an EmotionML 1.0. W3C Final Incubator Group Report, World Wide Web Consortium. http://www.w3.org/2005/Incubator/emotion/XGR-emotionml-20081120/

- Burkhardt, F. and Schröder, M. (2008). Emotion Markup Language: Requirements with Priorities. W3C Incubator Group Report, World Wide Web Consortium. http://www.w3.org/2005/Incubator/emotion/XGR-requirements-20080513/

- Schröder, M., Zovato, E., Pirker, H., Peter, C. and Burkhardt, F. (2007). W3C Emotion Incubator Group Final Report. World Wide Web Consortium. http://www.w3.org/2005/Incubator/emotion/XGR-emotion-20070710

Public project deliverable reports

- Schröder, M., Bevacqua, E., Eyben, F., Gunes, H., ter Maat, M., Pammi, S., de Sevin, E., Valstar, M. and Wöllmer, M. (2010). Final SAL system (SEMAINE Project Deliverable No. D1d).
- Schröder, M. (2009). First full-scale SAL system (SEMAINE Project Deliverable No. D1c).

1.5 Collaborations

The work presented in this thesis would not have been achieved without the close interaction and collaborative work with colleagues in several contexts.

In the SEMAINE project, I have worked with project partners from TU München, ParisTech/CNRS, Universiteit Twente, Imperial College London, Queen's University Belfast, and DFKI. In particular, the design decisions of the SEMAINE API have been influenced by discussions and brainstorming meetings with Björn Schuller, Florian Eyben, Martin Wöllmer (München), Catherine Pelachaud, Elisabetta Bevacqua, Etienne de Sevin (Paris), Dirk Heylen, Mark ter Maat (Twente), Maja Pantic, Michel Valstar, Hatice Gunes (London), Roddy Cowie, Gary McKeown (Belfast), and Sathish Pammi (DFKI).

Similarly, work on the Emotion Markup Language is very much a collaborative endeavour. Work on the predecessor of EmotionML, the HUMAINE EARL, was carried out in collaboration with Hannes Pirker (OfAI Vienna) and Myriam Lamolle (IUT Paris), in the context of the HUMAINE project. Subsequent work on EmotionML in the context of the World Wide Web consortium (W3C) was carried out in collaboration with many people over the years, but most importantly Paolo Baggia and Enrico Zovato (Loquendo, Turin), Felix Burkhardt (Deutsche Telekom Laboratories, Berlin), Christian Peter (Fraunhofer IGD-R, Rostock), and Catherine Pelachaud (ParisTech).

1.6 Outline

The thesis is structured as follows.

This **Chapter 1 Introduction** identifies the topic area and the key contribution of the thesis. **Chapter 2 Motivation** describes from a conceptual point of view why it may be desirable to endow machines with emotional and social intelligence, and thus provides the reasons for starting an endeavour as the one described here.

The main body of the thesis consists of four parts.

The **Background** part reports on related work relevant to the thesis. **Chapter 3** summarises the state of the art in Embodied Conversational Agent research. Existing component integration frameworks are reviewed in **Chapter 4**, and their key properties are compared to one another and to the requirements for a component integration framework that arise from the SEMAINE project. We conclude that none of the existing frameworks is a suitable match for the project. **Chapter 5** therefore reviews the available choices of middleware and describes their key properties, thus preparing an informed choice of a suitable middleware technology in our newly created component integration framework.

In the **Infrastructure** part, we describe the key aspects of system infrastructure we have provided. **Chapter 6** presents the SEMAINE API, including the reasons for choosing a messaging middleware, and the properties of the SEMAINE API as a component integration framework. The system architecture of the SEMAINE system, built on top of the SEMAINE API, is described in **Chapter 7**. We briefly summarise the components involved in the system, and present two alternative system architectures organising the data flow: a naive pipeline architecture, and a more sophisticated prepare-and-trigger architecture.

The **Communication** part describes how the present work enables communication between system components by means of standard or pre-standard representation formats. **Chapter 8** provides an overview of the representation languages supported in the SEMAINE API, and describes their respective use within the context of a naturally interacting ECA system. **Chapter 9** reports on the process of defining the Emotion Markup Language as a new web standard.

The properties of the SEMAINE API are investigated in the **Assessment** part. We measure the performance of key aspects of the system in **Chapter 10**. In **Chapter 11**, we illustrate how the SEMAINE API makes it simple to build new

emotion-oriented systems from existing and new parts, using the SEMAINE API as a component integration framework.

In the final **Summary and Outlook**, we review the contribution of the thesis, and discuss its potential impact in the context of research on emotionally competent and naturally interacting machines.

Chapter 2

Motivation

In this chapter we motivate why we consider the topic area of emotion-oriented systems to be worthwhile: it can potentially increase the ease of use of complex technology for non-expert users. Furthermore we point out why a framework providing standards-based interfaces, encouraging re-use of system components, can be of benefit to the research community.

2.1 Emotional machine intelligence

Systems with some emotional competence, so-called 'affective computing' systems, are a promising and growing trend in human-machine interaction (HMI) technology. They promise to register a user's emotions and moods, for example to identify angry customers in interactive voice response (IVR) systems; to generate situationally appropriate emotional expression, such as the apologetic sound of a synthetic voice when a customer request cannot be fulfilled; in certain conditions they even aim to identify reasons for emotional reactions, using so-called 'affective reasoning' technology.

In fact, an increasing number of interactive systems deal with emotions and related states in one way or another. Common tasks include the analysis of the user's affective state (Zeng et al., 2007) from the face (Pantic and Rothkrantz, 2000; Ioannou et al., 2005), the voice (Batliner et al., 2006; Schuller et al., 2007), or physiological measures (Peter and Herbon, 2006); the evaluation of events in the world according to affective criteria (Gebhard, 2005); and the generation of emotion-

related system behaviour, such as facial expression (Tsapatsoulis et al., 2002; Bevacqua et al., 2007) and voice (Burkhardt and Sendlmeier, 2000; Schröder, 2008), but also other media such as music and colour (Castellano et al., 2007).

We discuss the relevance of such emotion-oriented systems in the broader context of machine intelligence.

In his seminal paper, Alan Turing (Turing, 1950) proposed to operationalise the question "Can machines think?" in terms of an "imitation game", a written dialogue between an interrogator and an entity which could be either a human or a machine imitating a human's behaviour. The idea in this "Turing Test" is that the machine can be said to be "intelligent" if the interrogator cannot reliably distinguish the machine from the human based on the written interchange. Turing uses poetry, maths, and chess as examples of intelligent behaviours that could be assessed via the written dialogue, and claims that "the question and answer method seems to be suitable for introducing almost any one of the fields of human endeavour that we wish to include" (p. 435). He proposed to focus on a written interchange so as not to "penalise the machine for its inability to shine in beauty competitions, nor to penalise a man for losing in a race against an aeroplane" (p. 435). In other words, the aim was to allow the human-machine interaction to focus on the relevant aspects of intelligence by leveling the playing field in other respects that are not so relevant for intelligence.

Here we argue in favour of a different perspective on intelligence in general and of machine intelligence in particular. We briefly describe a relevant aspect of human intelligence that is not covered by Turing's method, namely social and emotional intelligence. We discuss the metaphors used by humans to conceptualise machines, and try to corroborate the claim that the conceptualisation of machines as "social entities" is becoming increasingly important in the lives of people, thus motivating the need for social and emotional intelligence in machines.

2.1.1 Social and emotional intelligence

The concept of intelligence has evolved since 1950. Wechsler (1958) states that "intelligence is the aggregate or global capacity of the individual to act purposefully, to think rationally, and to deal effectively with his environment" (Wechsler, 1958, p. 7). Out of this generic notion of a *general intelligence*, Turing appears to

have focused on the capability to think rationally, the intellectual aspect of intelligence. In the human sciences, concepts have since been developed which are explicitly complementary to this purely rational notion of intelligence; among them are the notions of social intelligence and emotional intelligence. Indeed, dual-processing accounts from diverse literatures in cognitive and social psychology posit that rational thought is an architecturally and evolutionarily distinct mechanism sited amongst a much broader system or collection of systems that deal with most of the implicit, nonverbal and emotional aspects of human life and interaction (Evans, 2008). Although Turing sought to level the playing field by ignoring these aspects, there are likely to be important interactions between rational thought and this broad base of complementary mechanisms that could, if removed, tilt the playing field against a machine. A rationality test without social and emotional context would result in reduced tolerance and opportunity for repair. These are normally offered to humans due to cordiality, etiquette and a desire not to upset other individuals which normally incurs some form of social sanction.

Social intelligence is considered to encompass "our abilities to interpret others' behaviour in terms of mental states (thoughts, intentions, desires and beliefs), to interact both in complex social groups and in close relationships, to empathize with others' states of mind, and to predict how others will feel, think and behave" (Baron-Cohen et al., 1999, p. 1891). Humans can lack social intelligence while having a very high general intelligence, as can be the case for people with autism (Baron-Cohen et al., 1999).

The "social brain hypothesis" suggests that the rational intelligence Turing sought evolved as an extension and an ability to manipulate and manoeuvre through the intricacies of social dominance relations and social hierarchies (Dunbar, 2003). Almost synonymous with social cognitive abilities is the recognition of a Theory of Mind (Premack and Woodruff, 1978), the ability to read another's mental state and understand another individual as an intentional agent like oneself.

One relevant aspect of social intelligence is the ability to have a conversation with other people. This requires the respect of social conventions – hello and goodbye rituals, appropriate turn-taking, speaking at the right time, with appropriate loudness, choice of words, and gaze behaviour, depending on the situational context, the relation with the interlocutors and many other factors (Brunet et al., 2009).

Emotional intelligence was proposed as "the subset of social intelligence that involves the ability to monitor one's own and others' feelings and emotions, to discriminate among them and to use this information to guide one's thinking and actions" (Salovey and Mayer, 1990, p. 189). It includes the capability to become aware of, identify and label one's own or another person's affective state; the capability to reason in terms of the appraisals that lead to affective responses, and to predict possible future actions from the affective state; and finally, a set of capabilities related to the regulation of the affective states, be it the capability to hold back a socially inappropriate own emotion, or to act in a certain way so as to influence the emotions of another person. There is evidence that the capability of feeling one's own emotion is an essential element of a range of seemingly unrelated capabilities. For example, lesions in emotion-related brain regions leave people unable to feel emotions, engage in simple decision-making or make socially appropriate choices (Damasio, 1994). Capgras syndrome disconnects emotional from rational areas leading people to conclude impostors have replaced friends or family (Hirstein and Ramachandran, 1997).

What role should the concepts of social and emotional intelligence play in machine intelligence? The answer to this question naturally depends on the concept of machine intelligence one chooses to adopt, which is related to the perspective on machines in general. From a philosophical point of view, one could ask basic questions such as whether a machine can potentially be conscious. We will approach the topic from a more pragmatic perspective here, and view machines from a utilitarian point of view. From this viewpoint, we can say that a machine is "intelligent" if it is useful, if it is good at its job. Given the fact that humans have created machines to do work for them, one can say that a machine's job in general is to, in one way or another, make the life of humans easier.

The following section discusses metaphors that people seem to use in defining their relation with machines, and how this relates to the utilitarian view on machine intelligence.

2.1.2 Machines as social entities

Humans can conceptualise machines in terms of a range of metaphors, including the *tool* and the *social entity* (cf. Reeves and Nass (1996)). Simple machines such

as staplers, loudspeakers or vacuum cleaners are likely to be conceptualised as tools, not much different from a hammer. The machine, electronic or not, is perceived as an extended hand or arm (Lockman, 2000), as a thing with predictable properties that one can fully control if one has learned how to use it. In the hands of an expert, more complex machines such as mechanical clocks or desktop computers may also be approached from the "tools" metaphor: in operating the device, the human feels in control, and if something doesn't work as expected, the understanding is that there must be something wrong in the mechanism (be it the clock's cog wheels or the computer's programming) which could potentially be fixed.

As the mechanisms of machines become more complex and the causal chains of functioning become opaque, humans need to replace simple cause and effect models with more complex mental models of functioning. A natural second metaphor for machines is that of a social entity. Even though people often explicitly deny anthropomorphism regarding machines and computers, they still interact with them as social entities in states of "mindlessness" (Nass and Moon, 2000) – states similar to the implicit/automatic systems in dual-processing theories (Evans, 2008). In fact, Reeves and Nass (1996) have shown that people tend to interact with computers, new media and the likes as they do with people: they are polite, behave differently with computers that speak with a male vs. a female voice, they use proximity-regulating behaviour with faces on the screen, and much more. It seems that people apply rules similar to those governing social behaviour when interacting with new technology beyond a certain threshold of complexity.

What determines, for a given machine, whether a given person will view it as a tool or a social entity, whether he or she will "use" or "interact with" the machine? We can only begin to discuss this point. Biocca et al. (2001) provide elements of a definition of "social presence", of the sense of "being with" another social entity. According to them, a sense of social presence requires, firstly, a co-presence, where one perceives the other and is perceived by the other; some "psychological involvement" in the sense that the user forms a mental model of the machine as having "some minimal intelligence in its reactions to the environment and the user" (Biocca et al., 2001, p. 7); and an element of behavioural engagement, including interaction and synchronisation between user and machine. From that point of view, then, the more reactive, interactive, and "intelligent" a machine becomes, the more likely users would perceive it as a social entity.

It seems that users generally prefer to feel "in control" when interacting with machines (Mick and Fournier, 1998; Dautenhahn, 1999). For simple and predictable machine behaviour, a sense of control may best be covered by the tool metaphor. Where the machine's complexity exceeds a certain threshold, it is common for users to feel incompetent and to use associated coping strategies (Mick and Fournier, 1998). One way to avoid this reaction is to *design* a user interface according to a social entity metaphor that mitigates the information overload (Maes, 1994); this then raises the expectation that behaviour should be predictable based on social conventions (Dautenhahn, 1999). Whether or not the same mechanism is unconsciously applied by people when dealing with complex machines in general, even machines not designed to behave as a social entity, remains to be investigated.

Society at large is clearly moving towards ever-more complex technology, with ever-fewer people in the position to really understand and control that technology. Technology is becoming more autonomous, from self-updating software programs on the desktop to autonomous vacuum cleaner robots. Often they take over roles previously filled by human service staff, as is the case for train ticket vending machines or airline self check-in terminals. These machines show some awareness of the user, they interact and exhibit some limited intelligence, so they seem to fulfill the minimal criteria for a feeling of social presence. If unpredictable behaviour is added to that, some users may indeed feel that they are interacting with a social entity.

At the same time, the functions filled by some of these machines have a high relevance for their users. If I need my train ticket or boarding card on time, the utilitarian "intelligence" of the machine has a high impact on my well-being. If it doesn't do its "job" right, the machine becomes an obstacle between me and my goal, and anger or a feeling of helplessness will result (Scherer, 1999).

This discussion is intended to show that there are reasons to believe people will increasingly use the "social entity" metaphor when interacting with machines. Whether or not they are designed explicitly for that, a machine's actions may be interpreted increasingly as social actions. Human users might, for example, attribute personality traits based on appearance, reaction speed, ease of use, clarity etc. They may apply their own social intelligence in an attempt to infer the machine's state of mind – is it thinking, grumbling, unfriendly, or why is it not

responding? The machine may be perceived as arrogant if the user's situation of distress is peacefully ignored by a pre-recorded friendly voice. The machine may be perceived as hectic or distressed if it moves too fast, or as sluggish or bored if it moves too slow, etc. Basically, the expectation is that human users will automatically compare the machine's behaviour with a somehow adapted version of their mental model of other social entities that they know, such as other humans, pets, cartoon characters or similar.

A valuable challenge for computer science, from this point of view, would therefore be to endow machines with the kinds of intelligence they need to be perceived as useful, helpful and intelligent in this utilitarian way, by providing them with the capabilities to perceive, predict and generate socially and emotionally relevant signals.

2.2 The potential benefits of standards for research on emotion-oriented systems

Different emotion-oriented systems have a number of elements in common. All of them need to represent emotional states in order to process them; and many of the systems are built from components, such as recognition, reasoning, or generation components, which need to communicate with one another to provide the system's capabilities. In the past, systems used custom solutions to these challenges, usually in clearly delimited ways that were tailor-made for their respective application areas (see Chapter 3 for related work). However, existing emotion-oriented systems seem to be neither explicitly geared towards the use of standard representations, nor are they available as open source.

Standards enable interoperability and reuse. Nowadays, standards are taken for granted in such things as the voltage of electricity in a given country, fuel grade, or the dimensions of screw threads (ISO, 1998). More recently, standards for document formats (ISO, 2006) have entered the public debate, under the perspective of long-term accessibility of documents. Web standards such as the Hyper-Text Markup Language HTML (Raggett et al., 1999) enable access to information in the world wide web through a broad variety of software products supporting the standard format.

Proprietary formats, on the other hand, can be used to safeguard a company's competitive advantage. By patenting, or even by simply not documenting a representation format, a company can make sure not to open up the market to its competitors.

The same considerations seem to apply in the emerging area of emotion-oriented systems. Agreeing on standard formats and interfaces would enable interoperability and reuse. An initial investment of effort in defining suitably broad but sufficiently delimited standard formats can be expected to pay off in the long run by removing the need to start from scratch with every new system built. Where formats, software frameworks and components are made generally available, for example as well-defined specifications or as open source software, these can be used as starting points and building blocks for new systems, speeding up development and research. Thus, to the extent that a framework for building emotion-oriented systems is easy to use, it may be able to promote progress in research in emotion-oriented technology.

Part I

Background

Chapter 3

Embodied Conversational Agent systems: elements of natural interaction and emotional competence

In the mid 1990's, spoken dialogue systems were investigated under the perspective of human language interfaces to information (Goddeau et al., 1994; Zue and Glass, 2002). Major concerns at the time were speech recognition and language understanding accuracy, and the challenge of extending the speech recognition vocabulary for open domains such as yellow pages lookup (Goddeau et al., 1994). Dialogue interaction was modelled as a ping-pong scenario where user and system would speak in turn. Naturalness of the interaction was not an aim as such; instead, criteria related to task performance were investigated, for example in the influential PARADISE framework for the evaluation of dialogue systems (Walker et al., 1997). PARADISE quantitatively assesses the quality of task-oriented dialogues in terms of usability measures. The objective of maximising user satisfaction is operationalised in terms of task success, and dialogue costs which quantify the efficiency of the dialogue at carrying out the task.

Conversational dialogue systems (Allen et al., 2001a) improved on the 'ping-pong' dialogue capabilities by adding the notions of system goals and dialogue obligations, leading to richer mixed-initiative dialogues. The notion of incremen-

tal analysis of user behaviour is introduced and identified as a precondition for giving listener feedback in order to ground the degree of mutual understanding in the dialogue. Similarly, incremental generation is introduced as a means to deal with issues of turn-taking and interruptions.

Instead of using hand-crafted dialogue designs, dialogue strategies can be optimised by statistically learning from simulated dialogues (Lemon and Pietquin, 2007). Statistical machine learning techniques explore in simulation the large space of possible system behaviours in the presence of a multitude of parameters, including user behaviour related to cooperativeness and expertise. The aspects of dialogue strategy that can be learned include selection of high-level plans such as utterance selection as well as low-level aspects such as turn-taking behaviour.

Despite intense efforts, none of the language processing technologies involved in a spoken dialogue system, as identified by Zue and Glass (2002), seem to have reached a state where they "just work". In particular, the single most important source of problems for spoken dialogue systems is still considered to be the limited accuracy of speech recognition technology (Pardo et al., 2010).

As explained in the Introduction to this thesis, the emphasis in the present work is not on improving these individual technologies, nor is the criterion for success of a system as we intend to build it appropriately defined in terms of task accuracy. Instead, important criteria are the naturalness with which a conversation flows, as well as the dialogue system's ability to deal with emotions in a competent way. The following sections review the state of the art of ECA technology with an emphasis on these factors.

3.1 Types of Embodied Conversational Agent systems

Embodied conversational agent (ECA) interfaces (Cassell et al., 2000) explicitly cast the interaction with a "system" in terms of the social metaphor of interacting with an "agent". The agent often takes the appearance of a human-like face or body displayed on a computer screen. ECA interfaces make it natural to reason in terms of human-human multimodal interaction capabilities.

André and Pelachaud (2010) have provided a thorough review of the history and current state of ECA technology. In the following subsections, we will follow André and Pelachaud in distinguishing four types of interactive setting into which the different ECA systems can be grouped. We will then cast a closer look at the specific aspects of expression and perception of emotion by ECAs, and existing work on the responsiveness of ECA systems.

3.1.1 TV-style presenter systems

The simplest, non-interactive version of an ECA system produces a monological presentation, where a single ECA presents information through spoken or textual verbal output accompanied by facial expression and/or gesture.

The virtual human presenter (Noma et al., 2002) takes as input text enriched with commands related to the ECA's body language. The generated speech and 3D animation are rendered in synchrony to perform a presentation in a virtual reality environment or on the web.

Similarly, the DFKI PPP persona (André et al., 1999) can present information by speaking, gesturing, and pointing to items displayed on the screen.

In the same general setting, AutoBriefer (André et al., 2005) can generate multimedia presentations supported by an animated agent, automatically generated from a high-level outline. Users can create or edit domain knowledge for use in the presentation using a graphical user interface.

For the present work, presenter systems are of limited interest since they do not address the issues of natural interaction and emotional competence.

3.1.2 Presentation teams

André et al. (2000) proposed the idea of replacing a single presenter with a team of characters presenting information in a role-play fashion. This change is inspired by the evolution of TV commercials, which evolved from experts presenting the properties of a product towards interactive stories. Similar to TV commercials, presentation teams can share the task of presenting different viewpoints, discussing the pros and cons of a topic, etc. In addition, a presentation team can be designed such that it can be parameterised in different ways, so that a user can watch different variants of the same scene, for example by changing the personality of the

"customer" character from interested to sceptical, or by changing the topics to be discussed. André et al. (2000) illustrated the concept using two demonstrators, a sales scenario and a pair of characters jointly commenting a soccer match.

The sales scenario was carried further in the NECA project's eShowroom demonstrator (van Deemter et al., 2008), where scripted dialogues between ECAs were generated to present and discuss the properties of a car. The dialogue scripts contained annotations on how to emotionally influence the facial expression and tone of voice. Users could indicate a number of topics they wanted to see discussed, such as environmental friendliness, horse power, or family friendliness, as well as the personality of the salesperson and customer character. The characters' emotional expressivity was influenced by the personality as well as the preferences, both of which could be determined by the user.

As for the single presenter case, presentation teams are of limited interest for the present work due to the lack of a real-time interactive conversational setting.

3.1.3 One-to-one human-ECA interaction

The ECA systems most similar to the present work are those that represent the communicative situation of a single ECA conversing with a single human user. A number of systems have been built according to this paradigm.

One of the first ECA systems is Gandalf (Thórisson, 1997), a multimodal dialogue system which interacts with the human user via speech, facial expression and hand gesture. The user's behaviour is observed using speech recognition, an eye tracker and a body suit. Gandalf models turn-taking and backchannel feedback (Thórisson, 2002). It runs as a multilayered architecture distributed across eight computers.

The ECA system August (Gustafson et al., 1999) communicates with users through synthetic speech, facial expression, and head movements. For some of the synthetic utterances, the prosody has been manually tuned to sound very natural. The system was placed in a public location (a cultural centre) for a period of six months to collect data on naive users' interaction patterns. The dialogue system supported a variety of topic domains as well as behaviour of different complexity in order to investigate the effect of these factors on human speech input. For the human interlocutor, only speech input was analysed. Importantly, given the noisy

environment in a public space, users had to press a push-to-talk button in order to speak to the system. Consequently, natural turn-taking was not an issue in the August system.

The Real-Estate Agent REA (Cassell et al., 1999, 2001) has been designed specifically with conversational naturalness in mind. The authors argue that a human-like appearance is a precondition for bringing the richness and naturalness of human-human dialogue to human-machine dialogue. User behaviour is observed in terms of face and hand positions, speech detection, and automatic speech recognition. REA uses gesture, facial expression, posture shifts, and synthetic voice to fulfil a range of conversational functions. For example, she generates listener feedback such as head nods or 'mmh' vocalisations when the human speaker makes a short pause, and uses gaze and gestures to reflect the agent's perception of turn management. Both task-oriented and social dialogue are supported.

The medical advisor agent Greta (Pelachaud et al., 2002) is an ECA system with an explicit model of personality, emotion and synchronised verbal and non-verbal behaviours. An important emphasis is on the generation of high-quality facial expressions reflecting the agent's mental state and dialogue message. The utterances produced by the agent are taken from a script of sentences that were manually annotated with the Affective Presentation Markup Language APML (de Carolis et al., 2004). Keyword spotting on user input is used to select sentences to be generated by the agent; no listener behaviour is supported.

The emphasis in the SmartKom system (Wahlster, 2003) is on symmetric multimodality. SmartKom is a multimodal dialogue system in the information kiosk scenario, with sophisticated support for multiple input and output modalities. The user interacts with a small i-shaped information agent named "Smartakus". Non-verbal behaviour by the user, such as deictic gestures, is interpreted in conjunction with speech input in order to resolve references. A typical scenario involves a user pointing to an area on a display and saying "I want to reserve this seat". In addition, the system has initial support for emotion recognition from speech and from facial expressions, and shows some limited expressivity to inform the user about the system's internal state while a system response is being generated.

The anthropomorphic agent Max (Kopp et al., 2003) inhabits a CAVE-like virtual reality environment and is visualised in human size. Max employs synthetic speech, gesture, facial expression and gaze to communicate with the user. Since

the user is wearing a data glove, position markers and a microphone, Max can observe the user's speech and gesture. The interaction is based on an assembly task, with Max explaining to the user how to assemble parts into an object. The agent's actions are controlled by a combination of reactive and deliberative control processes.

It is interesting to note that all of these systems have been presented in the years 1997-2003. This is not to say that research on this type of ECA-human dialogue has stopped there, though. As will be seen in Sections 3.2 to 3.4, the research focus has rather shifted towards more specific research questions.

3.1.4 Multiparty conversations

A number of projects investigated scenarios involving multiple ECAs and/or users.

Agneta and Frida (Höök et al., 1999) are two ECAs, representing mother and daughter, that watch the user browsing the web. They make humorous and ironic comments on the content of the web page, the user's browsing behaviour, and any server problems that may occur.

In the context of the Mission Rehearsal Exercise, Traum and Rickel (2002) have realised a multi-party dialogue system in an immersive virtual environment. ECAs communicate with a user and with each other. The system addresses issues relevant to situated communication, including proximity and attention, multimodal messages including speech and non-verbal signals, and the ability to maintain conversations involving multiple parties.

CrossTalk (Gebhard et al., 2003) is an interactive installation designed for display in public spaces such as trade fairs. It consists of two ECAs conversing with one another and optionally including a user in the scenario. The user can either watch passively, or communicate actively. The CrossTalk setup is based on the "theatre" metaphor, according to which the ECAs represent actors. While no user is deemed to be present, they switch into an "off-mode" and chat among themselves. When a user is detected, they switch into "on-mode" and start performing their drama. The user can comment via a GUI interface and thus influence the development of the dialogue. The dialogues in CrossTalk are designed using hierarchical finite state machines. A design tool called "SceneMaker" supports the developer in creating dialogues, thus reducing the level of expertise required to

create new dialogues. By separating the dialogue structure from the textual script, it is rather straightforward to translate a SceneMaker-based dialogue into a different language.

The VirtualHuman project (Kempe et al., 2005) supports dialogues involving multiple humans and multiple ECAs. Every ECA has its own "Conversational Dialogue Engine" (CDE); in addition, every user is represented by a CDE. The CDEs communicate with one another using an ontological representation, which means that an ECA perceives the actions of humans and of other ECAs in the same format. A user's CDE encapsulates the steps of analysing and interpreting the user's behaviour. The ECAs' behaviour is generated by a multimodal generation component, which takes into account an estimate of the time it will take to generate the utterance and, if the estimate exceeds a threshold, will insert hesitation vocalisations such as "well…" or "uhm…" to gain time without losing the floor. Emotions to be expressed by the ECAs are computed by rules operating on the domain knowledge, and are realised through facial expression, skin texture, tears, and breathing patterns.

The previous works presented in this section have focused on involving users in a dialogue situation with multiple ECAs. Isbister's Helper Agent (Isbister et al., 2000) does the opposite: it attempts to support human-human conversation in a video chat environment. Helper Agent is a dog-faced ECA which listens to the audio of the two persons interacting. Modelled on the role of a party host, it intervenes if it detects longer silences, and attempts to suggest "safe" topics of conversation.

A number of multiparty dialogue systems have been created around the idea of mixed reality games.

The game agent created by Rehm et al. (2005) fills the role of a co-player. It is projected onto the wall by the side of a table in order to convey the impression that the agent is sitting at the table with its human co-players, playing a dice game. An instrumented dice cup serves as a tangible interface which allows the agent to see the dice. Interaction with the human co-players is driven by the game logic. Users and ECA communicate via speech.

The project IDEAS4Games (Gebhard et al., 2008) has realised an emotion-aware poker game, in which two ECAs and a user play against each others with physical cards equipped with Radio Frequency Identification (RFID) tags. The

user interacts with the system by placing his own and the ECAs' poker cards onto a poker table equipped with an RFID tag reader, and by selecting options from a graphical user interface (GUI) menu on the screen. The ECAs' utterances are determined by game events. The emotions of the ECAs are computed from game events using an affective reasoner (Gebhard, 2005), and realised through the synthetic voice and through body movements.

3.2 Expressive behaviour

The main justification for the existence of ECA systems is for them to be able to represent, through their expressive behaviour, the computer's side of the human-computer interaction in a way that is natural for humans to interpret (Cassell et al., 2000). Expressive behaviour includes facial expression, head movements, gestures, body posture, and voice; sometimes other modalities are included as well, such as skin texture or proxemics. A lot of effort has been invested in improving the technologies available for generating natural expressive behaviour in these different modalities. We will briefly review a number of important aspects.

The simulation of facial expressions in animated agent systems has a long history. Most models are based on Ekman's work on the six basic emotion-related facial expressions (Ekman, 2003). MPEG-4 based facial animation systems, such as the Greta player (Hartmann et al., 2002), can be used to render these expressions; Raouzaiou et al. (2006) have generated mixed expressions as weighted combinations of these expressions. Similarly, Albrecht et al. (2005) have interpolated between emotion-specific facial expressions to generate expressions for intermediate emotions.

Kipp et al. (2007) addressed the generation of hand and arm gestures. They trained a machine learning algorithm on an annotated database of gestures as produced by one specific person. The algorithm is able to predict from text, annotated with word and utterance boundaries as well as topic/focus structure, a sequence of gestures as the training person might have produced it.

The EMOTE system (Chi et al., 2000) makes available the parameters *effort* and *shape* as properties of gestures that can be controlled in a three-dimensional animated agent system. Similarly, the spatial and dynamic properties of gestures

– the speed of movement, the size of a gesture, etc. – are controlled by Hartmann et al. (2006) in terms of a small number of *expressivity parameter* dimensions.

The emotionally expressive sound of an agent's voice is addressed in the topic area of expressive speech synthesis. A number of technologies can be applied to achieve the intended expressive speaking style, such as domain-oriented unit selection, signal modification, voice conversion or statistical parametric synthesis trained on expressive speech material. For a detailed review, see Schröder (2009a).

Bodily behaviour is also modelled sometimes, for example in the context of expressing emotions (Hyniewska et al., 2010), or when using proxemics, i.e. distance behaviour, as a means to influence the social behaviour of human users (Rehm et al., 2005).

As additional modalities, the VirtualHuman project (Reithinger et al., 2006) added breathing patterns to reflect arousal, skin texture to simulate a blush, and tears to express sadness.

Whereas much certainly remains to be done in order to optimise the technical properties of these expressive behaviours, it is important to investigate the question whether and how expressive behaviours are actually perceived by human users as improving the interaction.

One relevant question in this context is the degree of human-likeness that behaviours should show. Experience from the Disney animation studios seems to suggest that for cartoon-like characters, it is important not to simply mimick behaviours as humans would perform them, but to substantially exaggerate them (Bates, 1994). Other research strives for human-likeness, both in optical realism (Albrecht et al., 2005) and in gesture dynamics (Kipp et al., 2007).

Regarding the role of expressivity in dialogue, it appears that the expressive behaviour that is generated can in fact have a positive effect on the dialogue. For example, Pardo et al. (2010) found that the presence of an ECA showing contextually appropriate gestural behaviour had a positive effect on objective measures of dialogue success related to turn management and error recovery. At the same time, the perception of expressive behaviour appears to depend on user characteristics: Krämer et al. (2010) found systematic effects of the users' gender, age, and computer literacy on their assessment of an ECA's non-verbal behaviour. For example, female users gave better ratings than male ones when the agent showed

more smiles and self-touching gestures; male users preferred less non-verbal behaviour.

3.3 Perceiving emotions

While simulating an emotional expression is relatively easy for an ECA, dealing competently with human emotion is more difficult. A first major hurdle is the actual perception of that emotion, from any and all available modalities, which depending on the specific context may include face, head movement, voice, words spoken, and physiological measures.

Zeng et al. (2007) present an overview of affect recognition research from face and voice modalities. It becomes clear from their overview that, despite substantial progress in the recent past, the issues around emotion recognition are far from resolved. Research has moved from recognising simulated emotion displays towards more naturally occurring emotion; the benefits of multimodal input are starting to be exploited. However, the quality reached still remains limited. For example, the CEICES initiative (Batliner et al., 2011) addressed a four-class problem in a speech corpus of children interacting with the robot dog AIBO. Despite advanced methods for feature selection from a very large space of acoustic and linguistic features, the recognition rates stayed below 70% accuracy.

Whereas the majority of emotion recognition works approach the problem as a discrete classification task, there is also a growing body of literature in which emotion recognition is addressed as a dimensional, continuous problem (Gunes and Pantic, 2010a). Rather than predicting the "right category" of an observed emotion, the recognition task consists in predicting a location in a uni- or multi-dimensional space. For example, Eyben et al. (2009) predicted positions on the emotion dimensions *arousal* and *valence* from acoustic and linguistic cues, with correlations of around 0.5 between manually labelled and automatically predicted values.

While face and voice are the most frequently studied modalities for predicting emotions, other modalities are also used. For example, Gunes and Pantic (2010b) report on the prediction of five emotion dimensions from head gestures. Poh et al. (2010) present a wearable sensor for electrodermal activity and show its ability to distinguish emotionally aroused from calm states.

Whereas these base technologies are the subject of intense research, the systems that make use of them to take the human user's emotion into account are still relatively few.

The SmartKom system (Wahlster, 2003) uses both speech and face analysis to determine the user's emotional state. When fusing the information from different modalities, dynamic confidence measures are used, since for example, the mouth region does not provide reliable information about the emotional state while the user is speaking.

Burkhardt et al. (2005) have presented a research prototype of an emotion-aware voice portal. The idea is to detect a customer's anger in time to redirect the customer to a human agent. A crucial aspect in this context is the challenge of keeping the false-positive rate low: treating a customer as angry who is not actually angry may have a very negative impact on the customer relationship.

Other systems also make use of emotion recognition without actually implementing a dialogue with the human.

Clavel et al. (2008) trained a fear-type emotion detector in view of public safety applications: a surveillance microphone would detect fear-type vocalisations in public spaces such as underground stations. The detector was trained on panic and fear scenes vs. non-fear scenes from movies. The error rate is around 30%.

Inanoglu and Caneel (2005) presented an emotionally aware voicemail system: by analysing the emotion expressed in voice mails, the system could present the user with a pre-selection of most relevant messages. The percentage of correct classifications on a number of binary choices, such as urgent vs. not urgent, or happy vs. sad, is around 60-70% each.

To summarise, it can be suspected that the high error rates found in existing emotion recognition methods still make it difficult to use these in interactive systems. On the other hand, also human judges have a rather limited classification accuracy (e.g., the human subjects in Banse and Scherer (1996) achieved around 60% correct classification in a forced choice setup with 14 categories of exaggerated acted emotional speech). Despite the limited accuracy, however, humans seem quite able to exploit emotional information in interaction. It may be that interactive systems need to develop more refined methods for dealing with emotional information under high uncertainty.

3.4 Responsiveness

Attention mechanisms are a first important aspect of a responsive system. Nakano et al. (2003) implemented a mechanism to observe a human user's head movements and gaze direction in the context of the MACK agent (Stocky and Cassell, 2002). By observing the user's head with a stereo camera, the system could identify head nods, and distinguish three gaze types: "looking at agent", "locking at map" (in a direction-giving scenario), and "looking elsewhere". This information is used by the "grounding module", together with verbal evidence, to determine whether a shared understanding between the agent and the user has been achieved. The rules used by the grounding module have been derived from a study of human-human direction-giving dialogues. The most important evidence for grounding was found to be lack of negative feedback.

Similarly, the conversational robot Mel (Sidner et al., 2005) tracks the human interlocutor's face to determine whether the user is looking at the robot, at an object of shared attention, or elsewhere. Mel uses this information to determine the user's engagement. This information is used for determining an appropriate dialogue strategy. Furthermore, the robot can identify head nods and interprets them as backchannels or agreement depending on the current state of the dialogue.

Observing behaviours relevant for attention is one thing; determining their meaning is another. Peters (2005) makes this step explicit in terms of a Theory of Mind: his ECA interprets the behaviours of other inhabitants of a virtual world, specifically the direction of eyes, head and body, in terms of the other's attention. The information is then used to model the other's presumed goals, and to take these into account alongside the ECA's own goals when taking a decision whether or not to initiate a conversation with the other.

A second important aspect of a responsive dialogue system is its turn-taking behaviour. A baseline approach consists in the system waiting for the user to be silent for longer than some threshold before taking the turn.

Edlund et al. (2005) proposed an end-of-utterance detection algorithm that takes into account not only the duration of silences, but also intonation. Level boundary tones in the mid pitch range are used as indicators for hesitation pauses, inhibiting the recognition as end-of-utterance even if a long pause follows. Experiments showed that the algorithm correlates with human judgments of utterance

segmentation; the algorithm triggered less false alarms than the baseline algorithm using only pause duration.

Raux (2008) has realised an adaptive threshold detection mechanism, thus reducing the threshold without increasing the error rate. He improved this further by representing turn-taking state by a finite-state machine, with turn-taking actions having different cost depending on the current state. In addition to further reducing the threshold for end-of-turn detection, this approach is also useful for handling interruptions.

Jonsdottir et al. (2008) implemented a machine learning approach to adapting the turn-taking behaviour of a dialogue system to the individual interlocutor in real time. By observing the interlocutor's prosody, their talking agent tunes its turn-taking behaviour on-the-fly to the current situation using reinforcement learning.

ter Maat and Heylen (2009) investigated the effect of varying timing of turn-taking in simulated dialogues on people's attribution of personality, emotion and interpersonal stance. They showed that if an agent allows for long pauses before taking the turn, that agent is perceived differently than if the pauses are shorter or the agent starts before the interlocutor has finished speaking. Differences were found in attributions of friendliness, rudeness, arousal and several other dimensions.

In a conversational setting, a third essential aspect of responsiveness is listener behaviour, i.e. the behaviour of the ECA while it is listening rather than speaking. This includes visual and vocal *backchannel* behaviour, such as head nods or "mmh" vocalisations, to signal to the speaker that one is still paying attention. A crucial decision to make is *when* to emit such behaviours.

Both turn-taking and listener feedback are addressed in the Ymir Turn-Taking Model YTTM (Thórisson, 2002), implemented in the Gandalf system (Thórisson, 1997). The YTTM is a rule-based system for generating turn-taking decisions, associated behaviour, and backchannel feedback behaviour. It is based on a multi-layered architecture of perception-action feedback loops running at different update frequencies. The rules are based on hypotheses derived from the human science literature.

Edlund and colleagues built Hummer, a system that automatically generates synthetic backchannel feedback (Wallers et al., 2006) at suitable moments as judged from the human speaker's prosody (Edlund and Heldner, 2006).

Morency et al. (2010) addressed the question of backchannel timing from a machine-learning perspective. They trained sequential probabilistic models (Hidden Markov Models and Conditional Random Fields) on a human-human interaction corpus, to predict the locations of listener backchannels based on the multimodal output features of the speaker (including prosody, spoken words and eye gaze). For the prediction of head nods, the model significantly outperformed a previous model based on hand-crafted rules.

Kopp et al. (2008) use listener behaviour as a means to communicate to the human user the agent's mental state. Their agent Max shows a range of multimodal backchannels, including head nods or shakes, tilts or protrusions varying in repetitions and movement quality. The choice of behaviour is based on the agent's incremental reasoning and deliberative processing while the user is typing his or her input into a keyboard. The timing of backchannels is based on end-of-utterance detection, which is simulated using the "enter" key.

Bevacqua et al. (2008) predict both the timing and the meaning to be expressed in listener behaviour, based on the user's non-verbal behaviour as well as the agent's characteristics such as its personality and emotional state. The meaning of a listener feedback is represented in terms of "communicative functions", such as agreement, interest, or liking. Prediction is based on hand-crafted rules.

The Rapport agent (Gratch et al., 2007) observes the head movements and voice prosody of a user telling a story, and generates contingent visual listener behaviour including nods and posture shifts. In a carefully controlled study, the automatically generated listener behaviour was rated approximately as well as natural human listener behaviour in a face-to-face condition, and significantly better than a non-contingent version of the system (effectively playing back the contingent behaviour generated from the *previous* subject). The rapport agent produces no vocal feedback, and it never speaks.

In a study with a variant of the Rapport agent, von der Pütten et al. (2009) investigated the role of perceived agency and of behavioural realism on the feeling of social presence. They varied the instructions given to subjects such that some believed they were interacting with a human through an ECA display, whereas the others believed they were interacting with a computer character. In both groups, behavioural realism was varied. No effect of perceived agency was found, but ratings of mutual awareness clearly showed a main effect of behavioural realism.

This seems to indicate that, given suitable expressive behaviour, it should be possible for an autonomous ECA to induce a sense of social presence.

The effectiveness of contingent emotional adaptation to a user's emotion in a dialogue system was investigated by Acosta (2009). In a speech-based dialogue system aiming to persuade students of the values of graduate school, the emotion-related prosody of the system's utterances was modified as a function of the emotion recognised from the preceding user utterance. System utterances were generic phrases that followed a pre-defined script; only their prosody was adapted. Users rated the system as significantly better on a number of rapport-related scales, compared to a neutral baseline as well as a non-contingent version where the expressivity matched the previous rather than the current subject.

Adaptation also appears to occur in the other direction. Porzel and Baudis (2004) investigated effects of a dialogue system on the user's non-verbal behaviour, and found a tendency for the human users to adapt to the non-verbal behaviour of their interlocutor. Human users produced significantly fewer vocal feedback signals, used less overlapping speech, and made longer pauses when interacting with a spoken dialogue system than when interacting with a human.

3.5 Conclusion

The present chapter has reviewed current research in Embodied Conversational Agent systems, specifically in view of natural interaction and emotional competence.

We presented four types of ECA-based scenarios: monologues, simulated dialogues, human-ECA dialogues, and multiparty interactions. For the human-ECA dialogue scenario, which is the most relevant for the present work, we looked at existing work on expressivity, emotion awareness, and responsiveness.

We found research on expressive behaviour of ECAs to be well advanced, with sophisticated and in part highly natural expressive behaviour being simulated through the face, the voice, body gestures, and sometimes other modalities such as proxemics or skin texture. There also is some evidence that adding expressive behaviour to a dialogue system can indeed improve the interaction.

In contrast, despite intense research, the capabilities of ECAs to detect human emotion are still quite preliminary. A main reason is the high error rate that state

of the art research still cannot surmount with naturally occurring, non-acted expressive data. Recent work attempts to complement the classification task with a dimensional paradigm, predicting positions on emotion dimensions rather than individual categories. Again, however, correlations with human judgements remain moderate. It may be that for interactive systems to make effective use of emotion detection capabilities, dialogue strategies need to be developed that can deal with high uncertainty.

In the area of responsiveness, we looked at the observation of human attention, turn-taking mechanisms, the simulation of listener behaviour, and adaptation to the user's expressive behaviour. Individual advances in each of these areas exist, partly using rule-based, partly using data-driven approaches. Evaluations showed that responsive systems often outperformed non-responsive baseline systems with respect to grounding, perceived naturalness, and rapport, and that the choice of behaviour has an effect on attributions of personality and emotion.

Despite the many efforts that have been reported in the literature, to the best of our knowledge, no full-scale dialogue system has been built before that takes into account the user's emotion and non-verbal behaviour from visual and vocal cues, and interacts in real time both as a speaker and a listener in a multimodal conversational setting. It is also significant that none of the systems mentioned above is publicly available as open source, which makes it difficult to improve existing work incrementally.

Chapter 4

Component integration frameworks for multimodal interactive systems

The topic of the present thesis is a component integration framework for an ECA system, suitable for natural interaction and for emotionally competent behaviour. In this chapter, we will point out a number of requirements as they arise from the SEMAINE system scenario, and review a number of related endeavours from this perspective.

4.1 Requirements for a component integration framework in SEMAINE

In order to be able to discuss and compare, in the following, a number of existing component integration frameworks for multimodal interactive systems, we start by making explicit a number of requirements that arise from the targeted type of system in the SEMAINE project: an ECA system with natural interaction and emotional competence, based on standard representations to the extent possible.

Given the expected complexity of the system, it is a requirement to keep the overall system state under control. The framework should provide a dedicated system manager functionality to keep an overview of the state of all system components.

Regarding component interfaces, the framework should provide support for both standard and custom XML formats, but also for text and binary information.

A number of practical requirements are also important: the framework must run on Windows, Linux and Mac OS X; it must be available for research, ideally as open source, but it must also be usable with closed source components since not all SEMAINE components will be made open source.

Finally, if possible the framework should be actively maintained so that in case of problems, there is a chance of getting help to resolve the issues.

4.2 Existing component integration frameworks

4.2.1 Mirage

Thórisson et al. (2004) propose and test-run a design methodology for the creation of interactive ECA-based systems. Following the basic idea of LEGO bricks, they conceive of the design as *constructionist*, i.e., modular in the sense of allowing re-use of existing building blocks. A key element of their design methodology appears to be a conceptual modularisation hierarchy, starting with the distinction of perception, decision/planning, and action/animation, and then distinguishing further sub-categories. Communication between components is conceptualised in terms of well-defined message types passing via blackboards with publish-subscribe functionality.

The design methodology is illustrated with the example of the Mirage system, an ECA in an augmented-reality setting which can perceive virtual and physical objects, and communicate with a human user through speech and gesture. The system uses Psyclone (CMLabs, 2007) as the middleware layer (see Chapter 5) to integrate components in C++ and Java for Linux, Mac OS X and Windows. All communication between components passes via a single blackboard. Components communicate via text messages in an explicitly documented ad-hoc format. The system runs as a distributed system, with different components running on different computers in order to achieve an acceptable run-time performance. Thórisson et al. (2004) point out that integration across multiple computers brings about a number of challenges, related to difficult-to-predict network delays and performance of individual computers. One means to address this issue is the use of universal time stamps in messages in order to coordinate events; further steps to

increase the fault-tolerance of the system were considered necessary but not yet implemented.

Mirage is not actually a component integration framework, but rather represents a design methodology. The Psyclone middleware used in Mirage is available for research purposes (see also Section 5.3).

4.2.2 GECA

The Generic Embodied Conversational Agent (GECA) framework (Huang et al., 2008) aims to facilitate the process of building integrated ECA systems from individual components. Similar to other ECA systems, the authors distinguish the capabilities of (i) acquiring user behaviour from sensor data, (ii) interpreting the meaning of the behaviour and deliberative planning; and (iii) generation of ECA behaviour. Given the fact that several of these functionalities are not domain-specific and thus potentially reusable, Huang et al. (2008) suggest to provide a consistent interface and integration platform for such components, across programming languages and operating systems.

The GECA platform supports C++, C# and Java components, which communicate via the OpenAIR protocol that is implemented by Psyclone (CMLabs, 2007); messages are sent using an asynchronous messaging paradigm, using blackboards with a publish-subscribe mechanism (see also Section 5.1). It is unclear which operating systems are supported. Messages are represented as XML structures, in three custom formats: a format for representing user input, partially inspired by W3C's Extensible MultiModal Annotation markup language EMMA (Johnston et al., 2009); a format for representing system behaviour to be generated, with easy interfaces to Microsoft's text-to-speech API (MS SAPI) as well as MPEG-4 Facial and Body Animation Parameters (FAP/BAP); and a scenario markup language for scripting the dialogue.

There is no mention of a publicly available version of GECA.

4.2.3 VHMsg

The Virtual Human Messaging Library (VHMsg) was created by the University of Southern California's Institute for Creative Technologies as part of their Virtual Human Toolkit (see `http://vhtoolkit.ict.usc.edu/index.php/VHMSG`) and is

available as open source under a GNU Lesser General Public License (LGPL) from `http://vhmsg.sourceforge.net`. It is based on the message-oriented middleware ActiveMQ, and provides an API wrapper to send and receive text messages between components in a simple way. Client wrappers are available for C++, Java, C#, and TCL. VHMsg has no built-in support for XML or binary messages; furthermore, it does not provide any central system manager support for verifying the overall system state, nor does it have any built-in centralised logging functionality.

4.2.4 MULTIPLATFORM

The MULTIPLATFORM (Multiple Language / Target Integration Platform for Modules) testbed (Herzog et al., 2004) served as the component integration layer in the projects Verbmobil (Wahlster, 2000) and Smartkom (Wahlster, 2006). Verbmobil was a very large joint research project in Germany working on a speech-to-speech translation system. Smartkom was a large joint project in Germany working on a multimodal interactive system. In both systems, MULTIPLATFORM in increasing levels of maturity was used to integrate the components in a distributed architecture.

Herzog et al. (2004) propose to distinguish a number of different types of components: *device*, *recognizer*, *analyzer*, *generator*, and *synthesizer* represent various stages of the multimodal analysis and synthesis processes; *modeller* and *service* represent knowledge sources and application-specific functionality, respectively. MULTIPLATFORM supports the programming languages C, C++, Java, Perl, Prolog, and Lisp, and the operating systems Windows, Linux and Solaris. It uses the parallel virtual machine PVM (Geist et al., 1994) for communicating information between components. A custom layer called "pool communication architecture" was added on top of PVM to support the publish-subscribe model of message-oriented middleware (see Section 5.1).

In addition to the functional components, MULTIPLATFORM provides a *testbed manager* which organises and keeps track of the overall system state, starts or resets components, and provides a graphical user interface presenting the full system. A logging component saves all messages to a log file for later inspection.

Messages are represented as XML structures. In Smartkom, MULTIPLATFORM used a unified XML language for multimodal dialogue systems called Multimodal Markup Language (M3L).

At the end of the SmartKom project, the software was released as open source (`http://multiplatform.sourceforge.net`), but has not been maintained after 2003. As of February 2011, the source code is not accessible anymore from the site.

4.2.5 CHILix

In the area of ubiquitous computing, the project Computers in the Human Loop (CHIL) investigated a broad range of smart space technologies for smart meeting room applications. Its system integration middleware CHILix (Dimakis et al., 2008) uses XML messages for integrating the components in a smart space application. CHILix supports synchronous (request-response) and asynchronous interaction, and publish-subscribe as well as point-to-point connections between components. The wire format appears to be a custom XML-over-TCP format. The platform is available on Windows, Linux and Mac OS X, with language bindings for Java, C/C++, Perl, and TCL. The freely available NIST DataFlow System II (NIST, 2008) is used for streaming distributed sensor data. CHILix supports the idea of using well-specified XML representations at component interfaces, in order to facilitate the exchange of components. However, XML message formats used in the CHIL project seem to be domain-specific, custom formats; documentation does not seem to be freely available.

The CHILix framework appears not to be publicly available.

4.2.6 CAST

In the domain of interactive robots research, the project CognitiveSystems (CoSy) has developed a system integration and communication layer called CoSy Architecture Schema Toolkit (CAST) (Hawes et al., 2009). The components of a robot's architecture are structured into subarchitectures in which components work on a jointly accessible working memory. Access to data structures is through predefined types, similar to objects. Communication passes through the object-oriented remote method invocation middleware ICE (Henning, 2009).

CAST is available on Linux and Mac OS X. It supports language bindings for Java, C++ and Python. Definitions of data representations at component interfaces are written in Slice, ICE's interface specification format, and compiled into Java, C++ and Python source code.

The CAST framework is available as open source under a GNU General Public License (GPL), and is currently being maintained in follow-up projects to CoSy.

4.2.7 Others

A number of component integration frameworks are of general interest but are not really applicable to the target domain of the present work. We mention them only briefly.

The Open Agent Architecture OAA (Martin et al., 1999) is a framework for integrating heterogeneous software agents in a distributed setup. All agents communicate with a *Facilitator* agent which dispatches information to other agents as needed. All messages between agents are represented in the Interagent Communication Language ICL, which is capable of representing natural language expressions. The focus of OAA appears to be on coordinating multiple agents for collaborative problem solving. Language bindings exist for Java, C++, and Prolog, for Windows, Linux and Solaris. The software is available free of charge under a research license.

The OpenInterface Platform (Lawson et al., 2009) aims to facilitate the creation and reuse of multimodal interactive systems. Components provide interfaces to the rest of the system, defined in the Component Interface Description Language CIDL. Communication between components passes via the OI Kernel, which runs on Windows and Linux. Components in Java, C++, C# and Matlab can be connected. Remote connection of components via TCP or RPC appears to be possible. The software is available as open source under a BSD license from `https://forge.openinterface.org/projects/oikernel/`; as of version 0.4.0, released in December 2009, the software still declares its maturity as "alpha software".

Sonntag et al. (2010) describe an Ontology-based Dialogue Platform (ODP) which uses an ontological representation for dialogue management and can flexibly interface with multiple devices and services. The platform itself does not pro-

vide support for multiple programming languages; interaction with devices and services is via a range of industry-standard protocols.

There seem to be no component integration frameworks with specific support for emotion-oriented systems. Existing commercial programming environments for character-based expressivity or for user perception technology provide relevant component technologies, but they do not allow the user to integrate components across programming languages and operating system platforms. For example, the EMotion FX SDK (Mystic Game Development, 2010) is a character animation engine that supports animation designers to streamline the process of designing the graphical properties of games characters and include them into a game environment. It includes facial animation such as emotional facial expressions and lip synchronisation. Luxand FaceSDK (Luxand, 2010) is a facial feature point detection software, which can be used for face detection, the generation of 3D face models, and the automatic creation of animated avatars. Both are relevant component technologies for an emotion-oriented system, but they do not solve the issue of how to integrate heterogeneous components across platforms.

4.2.8 Summary

Table 4.1 summarises key properties of the component integration frameworks presented in this chapter, and compares them with the requirements as defined in the SEMAINE project proposal.

First of all, it is interesting to note that most of the component integration frameworks discussed in this chapter have opted for an asynchronous publish-subscribe mechanism for connecting senders to receivers of information, either via blackboards or via messaging.

Only the MULTIPLATFORM framework appears to support system management explicitly, in the form of its testbed manager which checks that all components are in a running state.

Some of the frameworks use XML-based message formats for communication, others use ad-hoc text formats, and only CAST uses an object interface approach. All frameworks appear to rely on custom or ad hoc interface formats; none of the frameworks appears to provide support for any standard representation formats at component interfaces.

	SEMAINE requirements	Mirage	GECA	VHMsg	MULTI-PLATFORM	CHILix	CAST
Application domain	ECA system	ECA system	ECA system	ECA system	Multimodal dialogue	Smart Space	Interactive Robot
Integration approach		blackboard pub/sub	blackboard pub/sub	messaging pub/sub	messaging pub/sub	multiple	blackboard pub/sub
System mgt. support	yes	no	no	no	yes	?	?
Interface type	standard and custom XML, text, binary	ad hoc text	custom XML	ad hoc text	custom XML	custom XML	custom Object
XML support	yes	no	yes	no	yes	?	no
Using standard formats	yes	no	no	no	no	no	no
Operating systems	Windows, Linux, Mac	Windows, Linux, Mac	?	Windows, Linux, Mac	Windows, Linux, Solaris	Windows, Linux, Mac	Linux, Mac
Middleware used		Psyclone	OpenAIR	ActiveMQ	PVM	NDFS II	ICE
Available for research	yes	no	no	yes	(yes)	no	yes
Available as open source	yes	no	no	yes, LGPL	(yes, LGPL)	no	yes, GPL
Usable with open source components	yes	no	no	yes	yes	no	yes
Usable with closed source components	yes	no	no	yes	yes	no	no
Actively maintained	yes	no?	yes?	no?	no	no	yes

Table 4.1: Key properties of several component integration frameworks for multimodal interactive systems

Most frameworks support all three operating systems that are to be supported in SEMAINE: Windows, Linux and Mac OS X. Exceptions are CAST, which does not support Windows, and MULTIPLATFORM, which does not mention Mac OS X support, presumably because support for the framework was discontinued before Mac OS X was released.

An interesting variety of middlewares are used for supporting the communication; no clear pattern can be identified at this level.

Of the frameworks presented here, only few are publicly available: VHMsg, MULTIPLATFORM (assuming that despite the current technical challenges the code could be obtained), and CAST. Furthermore, the Psyclone middleware used in Mirage is available, but only under a closed-source research license, not as open source. The aggressive "viral" GPL license used by CAST, which requires all software distributed with CAST to also be released under the GPL, makes it difficult to use CAST with closed-source components.

Most of the frameworks appear not to be actively maintained; the exception appears to be CAST, where new versions continue to be released.

This comparison has shown that none of the existing frameworks covers all of SEMAINE's requirements. Only three frameworks are actually available, of which only one – CAST – appears to be actively maintained; however, it lacks support for Windows and follows an integration paradigm different from the XML-based message paradigm targeted in SEMAINE. Furthermore, it is encumbered by the GPL license, which makes its use with closed-source SEMAINE components difficult. MULTIPLATFORM has the most interesting feature set; however, support for it was discontinued in 2003, and it does not appear to support Mac OS X. The remaining available framework, VHMsg, lacks important properties such as XML support and a system manager.

4.3 Conclusion

This chapter has outlined the requirements that the SEMAINE system poses towards its component integration framework.

A number of existing component integration frameworks were reviewed and compared to the requirements. Since none of them fully – or even approximately

– addresses all requirements, we conclude that it is necessary to build a new component integration framework for SEMAINE.

One particular point for which no best practice pattern could be observed in existing work is the middleware used for the communication layer in the component integration framework. For this reason we review existing middlewares in the following chapter, before presenting the new component integration framework we built in Chapter 6.

Chapter 5

Middleware

Middleware is “a general-purpose service that sits between platforms and applications” (Bernstein, 1996, p. 89); being neither a platform (i.e., a processor architecture and operating system) nor an application, middleware services “are generic across applications and industries, they run on multiple platforms, they are distributed, and they support standard interfaces and protocols” (Bernstein, 1996, p. 90). The concept is very generic, and encompasses examples as diverse as, e.g., printing managers, directory servers, relational database systems, e-mail, resource brokers, authentication services, etc.

Here, we look at *communication middlewares* only, i.e. methods for making processes and components working potentially on different machines and in different processes, implemented using different programming languages, to work together. In the following, the term *middleware* is used in the sense of *communication middleware*.

As described in the previous chapter, we call a *component integration framework* the software that is used for connecting the components into a system; in order to implement a distributed system, this framework will use a *middleware* as its communication layer. In addition, the component integration framework may provide functionality such as a system manager to check the state of the components, debug tools such as a centralised logging functionality, or utilities such as XML handling tools.

The present chapter provides a broad review of existing middlewares. This is motivated by the conclusion of Chapter 4 that a new component integration

framework is needed for SEMAINE. The review of existing component integration frameworks has revealed a preference for messaging or blackboard middlewares, but no clear pattern was found regarding the specific middleware software used. For this reason, the present chapter lays the basis for an informed choice for the middleware used in the SEMAINE API component integration framework.

5.1 Message-oriented middlewares

A *message oriented middleware (MOM)* (Banavar et al., 1999) is specifically designed to integrate different applications or processes through messages being routed from message producers to message consumers. The aim is to "glue together applications both within and across organizations, without having to reengineer individual components" (Banavar et al., 1999). One method for describing how messages should be sent from sources to destinations is a *message flow graph*, in which the nodes represent components and the arcs represent message flows (Banavar et al., 1999).

MOM provides a fundamentally *asynchronous* communication protocol. The act of sending a message on the sender's side is decoupled from the act of receiving the message on the receiver's side, both temporally and programmatically. Whereas in synchronous middlewares, the sender is waiting for a return value or at least a confirmation from the receiver that the message has been properly received and handled, in the asynchronous case the sender is free immediately after handing over the message to the MOM. The sender does not need to consider whether or not the receiver is currently online and able to receive the message. The reliability of message delivery can be configured in the MOM in various ways – for example, messages can be stored persistently so that, even if the middleware itself needs to be restarted, no messages are lost; messages can be delivered to clients when they connect as if they had been connected at the time when the message was sent; the communication between middleware and receiver can include an acknowledgement that the message has been received. Furthermore, a message can have a life span, so that it is discarded if that time has passed before the message could be delivered. Naturally, more checks and safety measures add overhead and thereby reduce the throughput of the middleware.

In asynchronous communication, any feedback information from a message receiver's side back to the sender needs to be modelled explicitly. Where it is important to inform the sender about the reception, and possibly the result of some processing, the receiver needs to send another message back to the sender containing the relevant information.

Two main types of message communication can be distinguished: *queuing* and *publish-subscribe*. Message queues are essentially one-to-one communication channels, with a single sender and, in the simplest case, a single receiver. As the name suggests, a message queue will store messages that are sent until they can be delivered. In particular, this means that a fast sender does not need to wait for a slow receiver to process the message before it can send the next one. Queues can be configured to guarantee that a message has been delivered exactly once. For applications such as load balancing, where several receivers are supposed to share the work of processing messages, it is possible to register more than one receiver to a queue. The receiver that picks up a message will be the only recipient of that message. This is to guarantee that, for example, a shopping order is processed exactly once.

A more flexible communication setup is the *publish-subscribe* model. Here, communication passes via so-called *Topics* to which one or more senders can register as *publishers* and one or more receivers can register as *subscribers*. Whenever one of the publishers sends a message to the Topic, all subscribers receive the message. This model allows for the organisation of communication architectures by type of information: by convention, a Topic can represent a certain type of information, independently of the source of that information. All processing units that want to use that information can simply subscribe to the respective Topic to receive the respective messages.

Communication between the sender and the receiver of a message passes necessarily via the middleware, which can be a single stand-alone server. For improved performance, it is possible to interconnect several instances of a MOM server on different computers in order to share the task of message routing.

Messages are unconstrained with respect to the types of data they can transport. Plain text or binary messages are the most generic types of messages possible. Message-oriented middlewares usually do not provide mechanisms for verifying

the adherence of messages to a certain interface definition; any such verifications need to be carried out by the user of the middleware.

The market of message-oriented middlewares includes commercial as well as open source players. Well-established commercial solutions include IBM's MQ-Series, now called WebSphere MQ; Tibco Rendezvous; and Microsoft's MSMQ. In the Java world, the Java Message Service (JMS) is a standardised API for interfacing with a MOM from Java code. JMS interfaces exist for all major messaging solutions, either from the solution vendor itself of from third-party providers. Various open source implementations of JMS also exist, including Apache ActiveMQ (Apache Software Foundation, 2008).

As of today, MOMs from different vendors cannot easily be combined, because there is no agreed over-the-wire protocol for messaging middlewares. With such a protocol, it would be possible to combine senders, servers and receivers from different vendors. Recently, a working group of major companies from the technology and finance sectors have proposed the "Advanced Message Queuing Protocol" (AMQP) as a standard over-the-wire protocol. The standard has recently reached version 1.0 (AMQP Working Group, 2010). A number of implementations exist, including OpenAMQ, RabbitMQ, Qpid, and ZeroMQ. Many of these use a dual licensing model, providing the system for free download under an open source license and offering commercial licenses including support.

5.2 Remote invocation middlewares

Remote method invocation (RMI) middleware is centered around the idea of controlling a remote process through a local API as if it was a local process. To this end, RMI middlewares provide façade programming interfaces which represent the functionality of a remote process. Every time a method of the façade interface is called, a sequence of steps is carried out behind the scenes:

- the parameter data provided in the method call are serialised ("marshalled"), and passed over the wire to the remote process;
- the remote process deserialises ("unmarshals") the data and re-creates an internal representation of the data, e.g. as class objects;

- the remote method processes the data, and generates an internal representation of a return value;
- the return value is marshalled and passed over the wire back to the calling process;
- the calling process unmarshals the return data and re-creates an internal representation which is handed over to the calling code.

This sequence of steps is hidden from the user of the remote interface; the code looks like a local method invocation.

One important difference to MOM is that in RMI, communication is *synchronous*: while the above-mentioned sequence of steps unfolds, the calling code is waiting for the method to return. This is different from MOM where sending and receiving of messages is handled asynchronously. There, if a component at the other end of a message queue provides a return value, it sends it through a separate message queue back to the original sender, which receives it independently of the original request. It is up to the original sender to match the reply to the query. In RMI, this matching step is trivial due to the synchronous nature of the remote method call.

A second difference is the definition of interfaces. Where MOM can use unconstrained messages for communication, the creation of a façade interface in RMI requires a detailed formal definition of the functionalities provided by the remote method. In object-oriented RMI, this includes the definition of the structure of the data that is used as parameters and return values.

One simple RMI middleware is the XML Remote Procedure Call (XML-RPC) specification. It uses HTTP as the over-the-wire protocol, and defines simple data structures as combinations of primitives, which can be used for inter-process communication across many different programming languages. The method calls and return values are encoded in XML structures. XML-RPC does not provide full object-oriented remoting, but requires the client code to create structures that can be marshalled in the XML-RPC format. Also, XML-RPC does not perform client-side parameter checking: if the parameters sent to the remote method do not match the expectations, this will be noticed only at the remote end, and an error message is returned. Many vendor-specific non-standard extensions to XML-RPC exist,

adding different functionalities such as Java object serialisation, however at the cost of reduced interoperability.

A genuinely object-oriented RMI middleware is the famous Common Object Request Broker Architecture (CORBA). Its first version was defined in 1991 and therefore predates XML which was first defined in 1998. Interfaces to method calls are formally specified by means of an Interface Definition Language (IDL) which is converted both in the implementation and the façade into actual source code in the respective programming language. This allows for compile-time verification that the method invocation is using the correct format and type of data. Communication passes via Object Request Brokers (ORBs) which communicate with each other using one of a number of over-the-wire protocols which all instantiate the General Inter-ORB Protocol (GIOP). They transport data by means of the Common Data Representation (CDR), a standardised binary representation of the data. The most commonly used protocol is the Internet Inter-Orb Protocol (IIOP) that sends messages over TCP/IP. A wide range of programming languages are supported. While CORBA has been widely used, it has also been criticised (Henning, 2006) for being an overly complex standard. Its use seems to be declining.

More recently, a company called ZeroC created the Internet Communications Engine (ICE), a re-implementation of the main ideas behind CORBA while trying to avoid the mistakes and the complexity that, according to the creators of ICE, encumbered the CORBA standard (Henning, 2006). Similar to CORBA's IDL, ICE uses an interface definition (called SLICE, the Specification Language for ICE) from which client and server code stubs can be automatically generated in various programming languages. Over-the-wire communication uses a custom binary format. Method invocation can be done either synchronously or asynchronously, using a notification mechanism when the result of a remote invocation becomes available. ICE is optimised with respect to speed as well as memory and CPU consumption (Henning, 2009). It uses a dual licensing model: it is available under the GNU Public License (GPL) for open source projects, and under a commercial license for closed-source projects.

5.3 Blackboard architectures

Middleware centered around the Blackboard metaphor simulates the concept of a group of experts standing together in front of a blackboard, jointly trying to solve a problem. Each "expert" can contribute pieces of the solution, but it is through their collaboration that the solution is ultimately found. Each "expert" is watching for information that he/she can process further. The results of that processing steps can then be processed by other "experts" until some result is obtained.

In Blackboard architectures, a "Blackboard" or a "Space" is a distributed memory that different components can access jointly. Each of the components that is registered to a blackboard can place information items onto it and read information items from it. Specific types of items can be found by phrasing a request to the blackboard in terms of a template that matches some items but not others. Notification mechanisms can alert components of new items matching the template as they become available. Components can either take exclusive control of an item, by removing it from the blackboard, or merely read it and leave it visible for others.

If used with the Master-Worker pattern, a blackboard can be used for load balancing in a way that scales easily. The pattern consists of a Master process handing out work items by placing them on the blackboard; a flexible number of Workers are registered to the blackboard, each taking exclusive control of one work item at a time, processing it, and writing the result back to the blackboard. The master collects the results of processing. It is obvious that this approach can scale systems merely by adding more workers.

Another possibility is the use of several blackboards in a publish-subscribe pattern, similar to a MOM. When each blackboard represents a given type of information, some components can place information onto the blackboard and others can read it. Through the appropriate definition of blackboards and the corresponding publishers and subscribers, complex system architectures can be defined in a way that corresponds to a message flow graph in MOM. An important difference is that the information items stay on the blackboard until they are either removed or their lifetime expires, whereas in a MOM a message is delivered once to each subscriber and then discarded.

One moderately influential specification of a Blackboard architecture is the JavaSpaces specification by Sun Microsystems (Sun Microsystems, 2005). It de-

fines a joint object space that can be shared between distributed components. Its most prominent implementation is the commercial software GigaSpaces, which extends the pure java approach from Sun towards interoperability with C++ and .NET. While the software is rather expensive for full-scale commercial use, free licenses are offered to academia and to startup companies (GigaSpaces, 2009).

Psyclone (CMLabs, 2007) is an implementation of a blackboard architecture with publish-subscribe capabilities. It is used by some parts of the Embodied Conversational Agents (ECA) and Artificial Intelligence (AI) communities. Psyclone itself runs as a server process on Windows, Linux or Mac OS X; clients connect to it via the so-called OpenAIR protocol. Client implementations are available for Java, C++ and other programming languages. While the software is not open source, it is freely available for research purposes.

5.4 Web services

One specific segment of middleware-type applications has seen substantial growth in recent years: *web services* are pieces of reusable technology that are integrated over the world wide web, and can be combined to provide complex functionalities across the boundaries of individual service providers. For example, one service can provide stock quotes that a second service aggregates and analyses, and a third service presents these analyses to a customer. Another example could be web-based flight booking portals which provide a single homogeneous interface to their customers but interact with a broad range of airlines to determine the availability of flights according to the customer's wishes.

In other words, the idea behind web services is to provide a web that can be used by machines – while the human-readable world wide web provides a lot of information, the presentation of the information is too unstructured for machines to make reliable use of it. Instead, web services address the main concerns for software-level integration over the web: mechanisms to reach reliable interoperability over well-defined interfaces, and scalability to work in environments with, potentially, thousands or millions of simultaneous users.

In the context of the present thesis it is worth mentioning that real-time responsiveness is *not* a requirement for web services. Given that the typical latencies of web-based message transport are in the order of several tens or hundreds of mil-

liseconds (Gummadi et al., 2002), sub-millisecond responsiveness is not a priority for web services.

In the following, we present two approaches to realising web services which are in widespread use: Service-oriented Architecture and RESTful HTTP.

5.4.1 Service-Oriented Architecture

The Service-Oriented Architecture (SOA) approach to realising web services is centered around the concept of a self-contained and reusable *service* realised by a component over the web. The service, as an atomic process, represents a real-world task such as looking up a piece of information, booking a ticket, analysing or transforming data, etc. Each service component appears as a black box to its customers (Harding, 2006) even though it may be composed of other services.

The programming model for SOA resembles Remote invocation middlewares: a service is invoked remotely and synchronously, and generates some output from the given input. However, a number of specific aspects distinguish SOA from generic RMI architectures.

SOA includes a mechanism for interface definition. In view of interoperability, a reliable and full description of a service and how to use it is provided by the *Web Service Description Language* (WSDL) (Chinnici et al., 2007). The WSDL description of a service defines the interface between the service and its users. Everything the users need to know about the web service is defined in the WSDL description; in particular, users do not need to know how the service is implemented. This reflects a SOA design principle, the concept of *loose coupling* (Erl, 2005): interdependencies between services should be minimal and be limited to well-defined interfaces.

The communication format used between SOA services is the XML-based Simple Object Access Protocol (SOAP) (Gudgin et al., 2007). All information communicated in the input and output of a SOA web service is represented in a SOAP structure. While this is somewhat inefficient, due to the time needed to marshal, transmit and unmarshal large XML structures, it is a well-documented and parsable format which supports interoperability. While SOAP as such is independent of the transport layer, it is predominantly used via HTTP.

Applications are built from services by means of a workflow definition ("orchestration") language, the Business Process Execution Language BPEL (Jordan and Evdemon, 2007). The language provides a syntax for representing workflows over SOA services, including sequences, conditions, loops, dependencies, etc.

A large number of additional specifications exist to standardise a broad range of functionalities in the SOA universe. However, these are beyond the scope of this short overview.

Given the commercial success of the SOA approach to web services, there are many SOA frameworks available. Open source options include Glassfish, Apache Geronimo and JBoss; commercial implementations include, for example, Oracle SOA Suite, TIBCO ActiveMatrix, and IBM Websphere ESB.

Regarding SOA performance, the additional abstraction layers of SOAP-based encoding of object-type information make the process of sending and receiving messages more costly than in simpler middlewares such as MOMs. This is not too problematic for the large-chunk structure of loosely coupled SOA components, which exchange comparatively few messages; for a real-time interactive system exchanging a large number of fine-grained messages per second, however, the overhead in SOA makes it suboptimal for the task.

5.4.2 RESTful HTTP

Roy Fielding, one of the main developers of the standard Hypertext Transfer Protocol (HTTP) specification (Fielding et al., 1999), defined the concept of Representational State Transfer (REST) in his PhD thesis (Fielding, 2000). REST describes "an architectural style for distributed hypermedia systems" (Fielding, 2000, p. 76): a set of design principles for building distributed systems based on the same principles as the world wide web. According to Fielding, these principles are the following.

- **Client-server communication.** REST architectures are fundamentally based on the concept of a client interacting with a server.

- **Stateless communication.** In REST architectures, the communication between client and server is stateless; all information needed for the server to

process the client's request must be contained in the request. The state of the interaction is fully represented in the client.

- **Caching.** For efficiency, any resources communicated from the server to the client make it explicit whether or not they can be cached.

- **Uniform interface.** Independently of their implementation, all REST servers expose the same kind of interface to the client. This is a thoroughly reductionist constraint which forces server implementations to shield off their implementation specificities for the sake of interoperability. The specific interface constraints are (1) the unique identification of resources, (2) the manipulation of resources by the client through representations of these resources, (3) self-descriptive messages, and (4) the use of hypermedia as the "engine" of application state.

- **Single-layer visibility.** Obviously, in any architecture a server can itself act as a client to additional servers, potentially yielding complex architectures. In REST, however, a client only sees the single layer that it interacts with. This enables, for example, improved scalability through the use of caching and load-balancing front-ends, as well as the implementation of security policies such as access control.

- **Code-on-demand.** Optionally, REST allows a server to send the client code to be executed on the client side. This limits the requirements on pre-defined functionality provided by the client.

This view of a distributed system is fundamentally different from the previously mentioned SOA approach. Whereas SOA focuses around the notion of processes and workflows, REST is phrased in terms of resources and their representations (Richardson and Ruby, 2007).

While the REST principles are independent of any specific implementation, by far the most important real-world implementation is what is typically referred to as "RESTful HTTP": using the HTTP transfer protocol for the communication and the interface, and Universal Resource Identifiers (URIs) to identify resources. This means that all client-server interaction functionality must be phrased in terms of the four HTTP actions: GET, POST, PUT and DELETE. When a client requests

a representation of the resource identified by a given URI, this representation can contain further URIs; the client's state changes by using these URIs in subsequent HTTP requests.

REST is agnostic about the content of the resource representations that a RESTful server provides to a client; it is merely suggested that standard data types should be used, and that the MIME type as provided in both requests and responses of the HTTP protocol be used for identifying the type of data being transferred. For machine-to-machine interaction, the use of XML formats supports general interoperability; however, how to interpret any concrete representations is considered an application-level concern independent of the REST approach.

Fielding emphasizes the fact that REST principles are intended for the specific case of web-based systems, and that they may not be optimal for other kinds of system integration: "The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction." (Fielding, 2000, p. 82)

In particular, this approach seems inappropriate for real-time interactive multimodal systems in which many fine-grained messages need to be transmitted.

5.5 Linguistic middlewares

A number of middlewares and component integration frameworks have been created specifically in order to facilitate the combination of linguistic analysis components.

Unstructured Information Management Architecture (UIMA) version 1.0 (Ferrucci et al., 2009) is an OASIS standard defining platform-independent data representations and interfaces that allow the analysis of text and multimodal data, with the aim of enabling interoperability among analysis components. Unstructured data is represented by means of a Common Analysis Structure (CAS). The CAS is an object graph consisting of objects representing the Subject of analysis ("Sofa") and objects representing the Annotation which points to a section of the Sofa as a stand-off annotation. All representations use standard object-oriented representations including Unified Modelling Language (UML) and XML Metadata Interchange (XMI). They are grounded in a user-extensible type system. There is not

yet an agreed body of types, but the specification expresses the intention to set up domain-specific type systems in the future, to facilitate the interoperability of components within a given domain area.

Apache UIMA (Apache Software Foundation, 2010) is an implementation of the UIMA specification from the Apache Foundation. It provides a Java Framework for creating analysis components communicating via the CAS. A C++ framework adaptor is provided to allow for the integration of C++ components via JNI. For scalability, the UIMA Asynchronous Scaleout (UIMA-AS) manager enables the definition of distributed systems; the message-oriented middleware ActiveMQ (see Section 5.1) is used for the communication between components. Apache UIMA comes pre-packaged with a number of linguistic Annotator components including a tokenizer, stemmer, part-of-speech tagger, named entity extraction, and more. For simple integration into other web services, a simple server is provided that presents a REST interface (see Section 5.4.2) for sending queries and receiving the analysis results in reply.

SeMantic Information Logistics Architecture (SMILA), an open-source component integration framework initiated by the research project Theseus-ORDO, is aimed towards the creation of "search solutions for unstructured information in the enterprise" (Schütz, 2008, p. 6). SMILA is founded on the principles of the SOA approach (see 5.4.1), thus naturally allowing for orchestration via BPEL and distributed execution of components as web services. It leverages OSGi (The OSGi Alliance, 2009), the Java component integration technology underlying, for example, Eclipse, a widely used Integrated Development Environment (IDE). Integration of non-Java components via suitable adaptors is intended for the future; currently, SMILA is an Eclipse Incubator project working towards its 1.0 release.

5.6 Multimedia middlewares

Several dedicated middlewares have been created for providing connectivity and interoperability among multimedia components.

5.6.1 Network-integrated multimedia middleware

Network-integrated multimedia middleware (NMM) is a middleware specialised in the distributed handling of multimedia data (Lohse, 2007; Lohse et al., 2008). It can combine recording, processing, and output of multimedia data across multiple computers and in multiple formats in a synchronised way.

Technologically, NMM combines elements of message-oriented (see Section 5.1) and remote-invocation (see Section 5.2) middlewares.

The flow of data between components follows message-oriented middleware (MOM) principles. Processing components in NMM are called *nodes*. They are interconnected using *jacks*, as a metaphor for the analog method of connecting multimedia equipment using jacks and cables. Flow graphs, similar to the *message flow graphs* in MOM, are created by connecting one node's output jack to another node's input jack. Messages of two types are transported along these connections: *buffers* transporting actual multimedia data, and *events* transporting control information such as requests to change the volume of an output device. The creation of flow graphs works transparently for the developer, in the sense that the same approach is used to interconnect two nodes running on the same machine and two nodes running on remote devices. NMM automatically selects the most efficient means of transporting the data.

In addition, NMM also contains a remote method invocation (RMI) communication layer, providing object-oriented interfaces to nodes and jacks which are defined using an Interface Definition Language (IDL) similar to Corba. This layer is particularly suited for a high-level, type-safe control of components from the system application.

NMM provides a temporal synchronisation mechanism to ensure time-aligned presentation across different devices. A registry service which is part of the middleware allows for dynamic registration and lookup of nodes. Multiple plugins exist for handling many different codecs, input and output devices.

The NMM software was developed at the Computer Graphics lab at Saarland University, and is now supported by the Motama GmbH, a spin-off company of Saarland University. Motama provides the software through a dual-licensing scheme, as open source software under the GPL and under a commercial license.

NMM runs in C++ on Windows, Linux, Mac OS X and PS3. It is not yet possible to implement NMM nodes in Java.

5.6.2 Open Sound Control

Open Sound Control (OSC) is a protocol for communication among software and hardware audio components (Wright, 2002), developed as a more versatile successor to the Musical Instruments Digital Interface MIDI (MIDI Manufacturers Association, 1982). It defines message formats in terms of primitive data types, independently of the transport layer. Data are indexed by hierarchically-organised OSC Address Patterns, allowing for flexible transport of arbitrary data. The meaning of fields is not standardised; it is up to the application to define a suitable set of fields for communicating the necessary information.

The downside of this flexibility, paired with the lack of agreed types of information, is the problem of interoperability of different implementations of the OSC protocol. The OSC website (Wright, 2002) lists several dozen implementations of the OSC specification in both hardware and software; due to the lack of standardised meanings of message fields, these cannot be guaranteed to interoperate.

One important software supporting the OSC protocol is PureData (Puckette, 2009), a graphical programming toolkit for sound and multimedia. It uses OSC for distributed implementation of complex signal processing setups.

5.7 How to choose a middleware

The choice of the right middleware for a given integration task should, firstly, be guided by an analysis of requirements.

One important aspect of requirements is the expected main type of interaction between components. If information mainly travels from one component to the next, and little synchronisation is required between the individual components' processing, an asychronous message-oriented (MOM) paradigm may be suited. If components essentially need to submit data to other components and require a synchronised response, then a remote-invocation (RMI) paradigm may be more suited. Where several components repeatedly need to access the same data as it evolves over time, a blackboard approach may be most suitable. This choice,

abstract as it may seem, is the most basic decision to be taken when choosing a middleware, and it will determine much of the integration experience that will result.

Another important aspect is the expected type of information flowing between components. To what extent is it considered desirable to use automatic verification of representation formats at component interfaces? Is there a need for object-oriented interfaces? The Interface Definition Language (IDL) formalisms used in Object Request Broker (ORB) type RMI technology provides this functionality. It supports the developer in verifying that only syntactically correct requests are sent, in which objects are marshalled at the sender end into an over-the-wire format and unmarshalled at the receiver end. The alternative to object-oriented interfaces are simpler interfaces provided in terms of primitive data types. These can be used to encode information in XML-based standard formats, or in custom string-based or binary representations. As a matter of principle, and independently of the efficiency of any given implementation, there is additional effort required for the marshalling and unmarshalling steps inherent in the ORB type of object-oriented communication, which is not necessarily the case in simpler interfaces. Therefore, object-oriented communication between components must generally be expected to be slower than communication using basic data types.

A third relevant aspect is the question whether the communication is required within a local distributed system or across the world wide web. For web communication, web services should be used whereas for local distributed systems, simpler and potentially much faster approaches can be used. Among the web services, there is the basic choice between Service-Oriented Architectures (SOA), implementing a web-ready version of the RMI paradigm, or the simpler REST approach. Whereas the latter does not make any commitments about the representation formats used at component interfaces, it makes very explicit use of optimisation mechanisms on the web today, such as caching.

Less principled but equally important questions concern practical issues. Which operating systems and programming languages must be supported? How easy is it to reconfigure a system architecture, to add or remove a system component? How fast is a particular implementation of a middleware technology? How well is it tested, supported, and documented? And, of particular relevance

for open source projects: is an open source version of the software available, and is its license compatible with the intended licensing scheme of the project?

Finally, there are considerations of interoperability and reuse. Among the suitable middlewares, is one more widely used by relevant colleagues in the field? Are there components available for reuse in some middleware framework? Are there standardised and well-documented interfaces allowing a user to combine existing with new components?

5.8 Conclusion

This chapter has provided an overview of different types of communication middlewares. The principles underlying messaging, remote invocation and blackboard middlewares were explained. A number of more specialised middlewares were also presented, including web services, linguistic, and multimedia middlewares. Finally, the chapter has formulated a number of criteria for selecting a suitable middleware for a given task.

In the next chapter, we will make use of the overview of middlewares presented here, and motivate the choice of the middleware used for the communication layer of the SEMAINE API component integration framework.

Part II

Infrastructure

Chapter 6

The SEMAINE API

This chapter will present the core technology developed in the present thesis: the component integration framework "SEMAINE API" used as the system integration backbone in the SEMAINE system. We start by motivating the choice of the underlying communication middleware from the requirements of the SEMAINE system, and then go on to describing the component integration framework itself.

6.1 Choice of a middleware for a naturally interacting and emotionally competent ECA

The most efficient method of passing data from one system component to another would be to maintain data in the run-time memory of one integrated piece of software. In such a monolithic system, only a reference to the data would need to be passed on from one component to the next. However, given the complexity of the system envisaged in the SEMAINE project, it is unrealistic to expect components provided by different research partners to be integrated into one piece of software. Not least, different components existing at different partner sites may be written in different programming languages; furthermore, they may be running on different operating systems. Clearly, thus, a middleware as defined in the previous chapter is required for integrating such a system. In the present section, we analyse the requirements arising from the target application, and select a suitable middleware.

6.1.1 Requirements for the ECA middleware

Using the criteria for choosing a middleware as elaborated in Section 5.7, we now define the requirements for choosing the middleware for the SEMAINE API.

Inter-component communication pattern

A reactive ECA system requires functionality for perceiving the user's behaviour, for reasoning about that, and for generating system behaviour in a timely manner. These functionalities can be clearly distinguished and delimited, so that they can be implemented by separate components. For example, a component for analysing the user's speech from the microphone signal is different from the component that interprets the user's state on the basis of that analysis; the decision for the system to act in a certain way, based on the current understanding of the user's state, can be clearly distinguished from the audio-visual behaviour generation and rendering components. The following types of inter-component information flow can be foreseen:

- periodic or event-based updates of user behaviour assessment;
- the system's current best guess of the user's state and of other relevant state information;
- system behaviour information at various levels of concreteness, from the proposed or selected action to the concrete data to be rendered.

Out of these, the user analysis data as well as the system output data are naturally described as volatile, momentary representations that need to pass through a number of components. A given piece of information may need to be analysed by one or more components; a given component may require one or more types of information. This is suitably described, in the general case, as a many-to-many message passing network. There is no obvious requirement to synchronise analysis components; they merely update the internally held information state whenever the respective analysis results become available. As for system output, there is an obvious requirement for synchronisation in rendering video and audio data, notably for lip syncing. As for the inter-component communication, however, the processing steps leading to the player input are loosely coupled in a way similar to

the input components; every component can simply process its input when it becomes available. The player must make sure all expected parts of an audio-visual output are present before it starts to render the output.

The system's representation of the user state is of a different nature. It constitutes a centrally held piece of information, frequently accessed or updated by several components.

Comparing these requirements to the basic types of communication middleware introduced in Chapter 5, we see that input and output messages naturally fit the message-oriented middleware (MOM) paradigm, whereas the state information would best fit the Blackboard paradigm. There does not seem to be an obvious need for a remote-invocation (RMI) functionality.

Representation of information flowing between components

The components closest to the physical world have clear representational constraints. On the input side, these are the feature extractors of audio and video signals. They produce feature vectors at periodic intervals, corresponding to a video frame for visual feature extraction and to an audio frame for speech feature extraction. Feature vectors are basically structured collections of values for which a convention must exist between producer and consumer regarding the meaning of the various values. Similarly, the input data to the player component consists of representations that can directly be rendered by the player, such as binary audio data and visual output parameters. For these representations, an object-oriented representation seems of little use; important is the possibility to transport the raw information as quickly and efficiently as possible.

For intermediate processing steps, the application scenario does not impose any strong constraints. Whether or not an object-oriented representation is chosen is a design choice.

Local vs. web integration

The real-time ECA system is intended as a local application, which runs as a distributed system merely for reasons of ease of integration and of distributing computationally intensive components onto several computers. Web capabilities are not required.

Operating systems

System components that pre-exist at different partners' sites are developed on Linux, Mac OS X and Windows.

Programming languages

Some pre-existing components are written in C++, others in Java.

Re-configurability

For research and testing, it should be as easy as possible to re-configure the system. The system should be robust to the addition or removal of a component, obviously within the limits of the application logic. This makes it possible to experiment with additional functionality. An example is a user presence detector, which makes use of the audio and video features to decide whether a user is present. If the component is absent, the system should continue to work as before, except that there is no automatic way of switching the context state regarding user presence.

Speed

Reaction speed is crucial for a real-time ECA system capable of rendering backchannels. Delays of the order of 350 to 700 ms can be crucial (Ward and Tsukahara, 2000). Given the fact that this number includes processing time, it is obvious that as little time as possible should be spent on communication between components. This argument favours simple middlewares based on raw data interfaces rather than object-oriented middlewares.

Quality of documentation and support

The middleware should be properly documented, and support should be available.

Licensing status

The reactive ECA system should be mainly open source, and a purely open source version should exist; however, individual components will be available only as freeware in binary form. Therefore, the system's core component must allow for

redistribution as open source as well as the combination with closed source components. This requirement excludes, on the one hand, commercial middlewares, since they are not redistributable. On the other hand, it also excludes open-source middlewares released under the GNU General Public License GPL, since that license disallows redistribution with components which are not themselves open source. This means that a suitable middleware should be available under a permissive open source license, such as the Apache License or the GNU Lesser General Public License LGPL.

Interoperability and reuse

The project should try to avoid the reduplication of effort where relevant components are already integrated into a suitable middleware. The components for planning and realising the behaviour of the 3d ECA Greta (Niewiadomski et al., 2009) are integrated via the Blackboard middleware Psyclone (CMLabs, 2007). Since the Greta components are an essential part of the targeted ECA system, Psyclone should be seriously considered as a candidate technology.

It is an aim of the present endeavour to establish a framework for creating new emotion-oriented interactive systems. To achieve this, the design choice has been made to use standards-based XML representations at component interfaces wherever that is possible and reasonable. This view on interoperability and reuse favours a middleware that makes it easy to transport XML representations between components.

6.1.2 Selection of a middleware software

The requirements analysis makes it clear that a suitable middleware should:

- be a message-oriented or Blackboard middleware;
- preferably provide fast primitive or XML-based rather than object-oriented interfaces;
- run on Windows, Mac OS X and Linux, with support for both C++ and Java;
- be well tested and supported;

- be publicly available for free, ideally under a permissive open source license.

These criteria exclude many of the options presented in Chapter 5. The requirement to support both Java and C++ excludes otherwise interesting technologies such as NMM, JavaSpaces or SMILA, and casts doubts over predominantly Java-oriented middlewares such as Apache UIMA. For speed reasons, the entire web service family must be excluded, including SOA and REST approaches. The preference for MOM or Blackboard middlewares, as well as the preference for primitive over object-oriented interfaces, disfavour CORBA. The preference for well tested and supported code casts doubts over XML-RPC and OSC. The exclusion of commercial middlewares rules out the commercial MOMs from IBM, Tibco and Microsoft, as well as GigaSpaces.

A small number of contenders remain after this filtering process: open source MOMs such as ActiveMQ; the research-only Blackboard middleware Psyclone; and, because of its asynchronous communication possibilities and fast implementation, the RMI middleware ICE. Their properties are now discussed and compared in some detail.

Psyclone

Psyclone provides blackboard-based communication with publish-subscribe capabilities, allowing for the flexible n-to-m type of information flow required in the project. It offers text-only message content, which is suitable for XML-type messages but suboptimal for binary audio data or feature vectors. The Psyclone server is available on Windows, Linux and Mac OS X, and clients for its OpenAIR protocol exist for C++ and Java. The code is not available as open source; binary releases are made available under a free research-only license. The degree of support is uncertain, and given the relatively narrow user base of Psyclone, it must be expected to contain some bugs. Documentation exists but is rudimentary. Most importantly though, we have found Psyclone to be very slow compared to ActiveMQ (see Chapter 10 for details).

ICE

ICE is essentially an object-oriented RMI middleware, providing both asynchronous and synchronous method invocation. It is thoroughly optimised to be

very fast and to require as little CPU and memory resources as possible. It supports C++ and Java, on Windows, Linux, and Mac OS X. It seems to be well supported and broadly used. Source code is available under the GPL. In fact, the use of the GPL is the only reason for not preferring ICE: since in our project we need to combine open source with closed source components, the GPL would disallow us to hand the full system to third parties.

ActiveMQ

ActiveMQ is an implementation of the Java Message Service (JMS) specification, i.e. a MOM implemented in Java. It provides primitive messages in a range of formats, including text and binary message types. This makes it optimally suited for the range of messages envisaged in our ECA. The Java server and client code run on any platform supporting Java; a C++ API runs on Windows, Linux, and Mac OS X. As an Apache project, ActiveMQ is broadly used, supported and actively developed. It is available as open source under the permissive Apache license.

In conclusion, out of these options, ActiveMQ and ICE are best from the points of view of speed and quality of support. The more permissive license speaks in favour of ActiveMQ. Psyclone remains a serious contender because of the previous integration of Greta via Psyclone; however, we have found Psyclone to be at least one order of magnitude slower than ActiveMQ (see Chapter 10).

For these reasons, we have chosen ActiveMQ as the communication middleware for our work. The next section describes the SEMAINE API, our component integration framework implemented on top of ActiveMQ. It provides additional functionality which simplifies the task of building a reactive ECA, and to lower the barrier to reuse existing components when creating new emotion-oriented systems.

6.2 Component Integration Framework

We have created the SEMAINE API as the component integration framework for the SEMAINE project (see Section 1.1).

Given the project's focus on standards-based integration of emotion-oriented components for real-time interaction, the SEMAINE API has the following main aims:

- to integrate the software components needed by the SEMAINE project in a robust, real-time system capable of multi-modal analysis and synthesis;
- to enable others to re-use the SEMAINE components, individually or in combination, as well as to add their own components, in order to build new emotion-oriented systems.

The present section describes how the SEMAINE API supports these goals on a technical level. First, we present the SEMAINE API's approach to system integration, including use of the message-oriented middleware ActiveMQ for communication between components, as well as the software support provided for building components that integrate neatly into the system and for producing and analysing the representation formats used in the system.

6.2.1 System integration

The essential building blocks in the SEMAINE API are *components*, which communicate with one another asynchronously by sending and receiving messages. All communication passes via so-called *Topics* (see Section 6.2.2); by convention, all data messages sent between components pass via Topics whose names are prefixed with '`semaine.data.`'.

In addition to data messages, components can also send and receive *callback* messages, which are intended for communicating processing state. For example, the player component can use callback messages to inform other components of the fact that a certain piece of system behaviour has started playing, has been completed, etc. Callback messages by convention are sent via Topics whose names are prefixed with '`semaine.callback.`'.

Up to here, the component integration framework does not really do more than providing conventions for using the middleware.

In order to actually provide component integration into a well-defined system, the SEMAINE API provides a mechanism for verifying and maintaining a global

system state. A *system manager* component maintains contact with a *meta messenger* embedded in every component. When a component is started, its meta messenger registers with the system manager over a special meta communication channel, the Topic 'semaine.meta'. At registration time, the meta messenger describes the component in terms of the data and callback Topics that it sends data to and that it receives data from; if the component is an input or output component (in the sense of the user interface), that status is communicated as well. The system manager is keeping track of the components that have been registered, and checks at regular intervals whether all components are still alive by sending a 'ping'. In reply to such a ping, each meta messenger confirms the respective component's status and sends debug information such as the average time spent processing incoming requests. The system manager keeps track of the information about registered components, and sends global meta messages informing all components that the overall system is ready or, if a component has an error or is stalled, that the system is not ready. Also, the system manager broadcasts a global system time. All components use this global time via their meta messenger, and thus can meaningfully communicate about timing of user and system events even across different computers with potentially unsynchronised hardware clocks.

A centralised logging functionality uses the Topics prefixed with 'semaine.log.'. By convention, messages are sent to 'semaine.log.<component>.<severity>', e.g. the Topic 'semaine.log.UtteranceInterpreter.debug' would be used for debug messages of component UtteranceInterpreter. The severities used are 'debug', 'info', 'warn' and 'error'. Through this design, it is possible for a log reader to subscribe, e.g., to all types of messages from one component, or to all messages from all components that have at least severity 'info', etc. Furthermore, a configurable *message logger* can optionally be used to log certain messages in order to follow and trace them. Notably, it is possible to read log messages in one central place, independently of the computer, operating system or programming language used by any given component.

Figure 6.1 illustrates this system architecture. Components communicate with each other via Topics in the 'semaine.data' and 'semaine.callback' hierarchies (indicated by black arrows). Meta information is passed between each component's meta messenger and the system manager via the 'semaine.meta' Topic

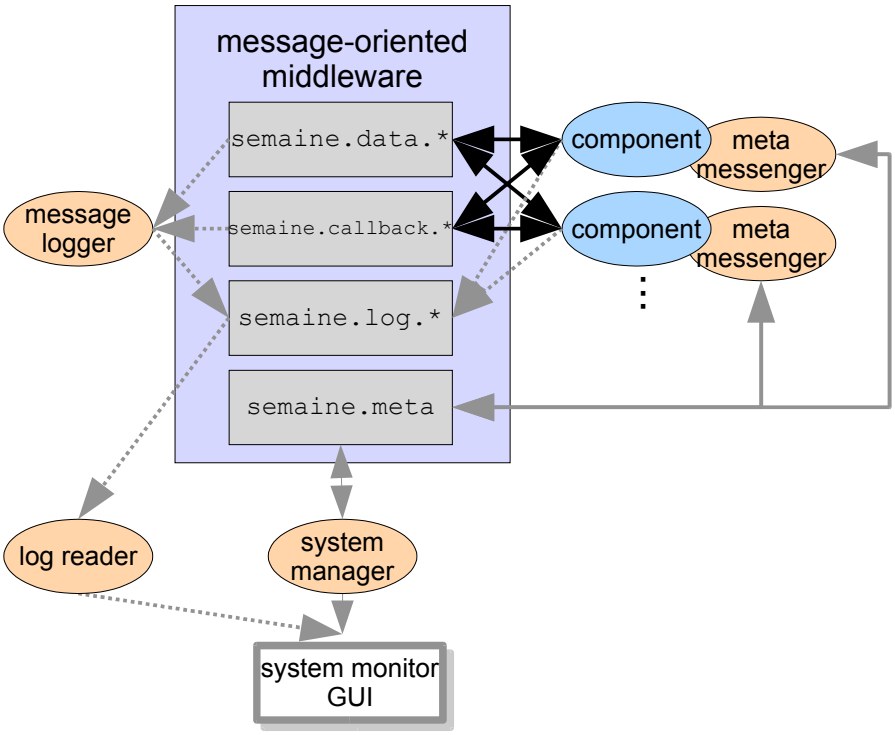

Figure 6.1: SEMAINE API system architecture

(grey arrows). Optionally, components can write log messages, and a message logger can log the content messages being sent; a configurable log reader can receive and display a configurable subset of the log messages (dashed grey arrows).

Optionally, a system monitor GUI visualises the information collected by the system manager as a message flow graph. Input components are placed at the bottom left, output components at the bottom right, and the other components are sorted to the extent possible based on the data producer/consumer relationships, along a half-circle from left to right. Where there are contradictory partial ordering constraints, such as in circular structures, a least-square solution is computed to minimise the global distance between producer/consumer neighbours. This criterion is overly simplistic for complex architectures, especially with circular message flows, but is sufficient for simple quasi-linear message flow graphs. If a new component is added, the organisation of the flow graph is recomputed. This way, it is possible to visualise message flows without having to pre-specify the layout.

Figure 6.2 shows the system monitor GUI for the SEMAINE-2.0 system described in Chapter 7. Components are represented as ovals, whereas Topics are represented as rectangles. Data Topics are shown between components; callback Topics are located in the middle of the image. Callback topics are blue when inactive whereas data topics are grey. Both turn yellow when a message is sent via the respective Topic, and slowly go back to their inactive colour. In Figure 6.2, it can be seen that there have been recent callback messages in the Topics '`semaine.callback.output.BAP`' and '`semaine.callback.output.FAP`', but not in '`semaine.callback.output.audio`'. In other words, the player has recently started or stopped playing an animation without vocal output, such as a head nod. When the user clicks on a Topic rectangle, the GUI shows a history of the messages transported by that Topic; debug information about a component is shown when the user clicks on the component oval. A log message reader is shown on the right-hand side. It can be configured with respect to the components and the severity of messages to show.

The remainder of this section describes the various aspects involved in the system in some more detail.

6.2.2 Topics

In its 'publish-subscribe' model, JMS routes messages via so-called *Topics* which can be identified by name. The SEMAINE API adopts this terminology. Names of Topics can be arbitrarily chosen. In order to establish a communication link from component A to component B, it is sufficient for component A to register as a 'publisher' to a named Topic, and for component B to register as a 'subscriber' to the same Topic. Whenever A sends a message to the Topic, B will receive the message. Topics allow for an arbitrary number of publishers and subscribers, so that it is trivial to set up n-to-m communication structures.

For a given system, it is reasonable to choose Topics such that they represent data of a specific type, i.e. with a well-defined meaning and representation format. This type of data may be produced by several system components, such as a range of modality-specific emotion classifiers. If there are no compelling reasons why their outputs need to be treated differently, it is possible to use a single Topic for their joint output, by registering all the components producing this data

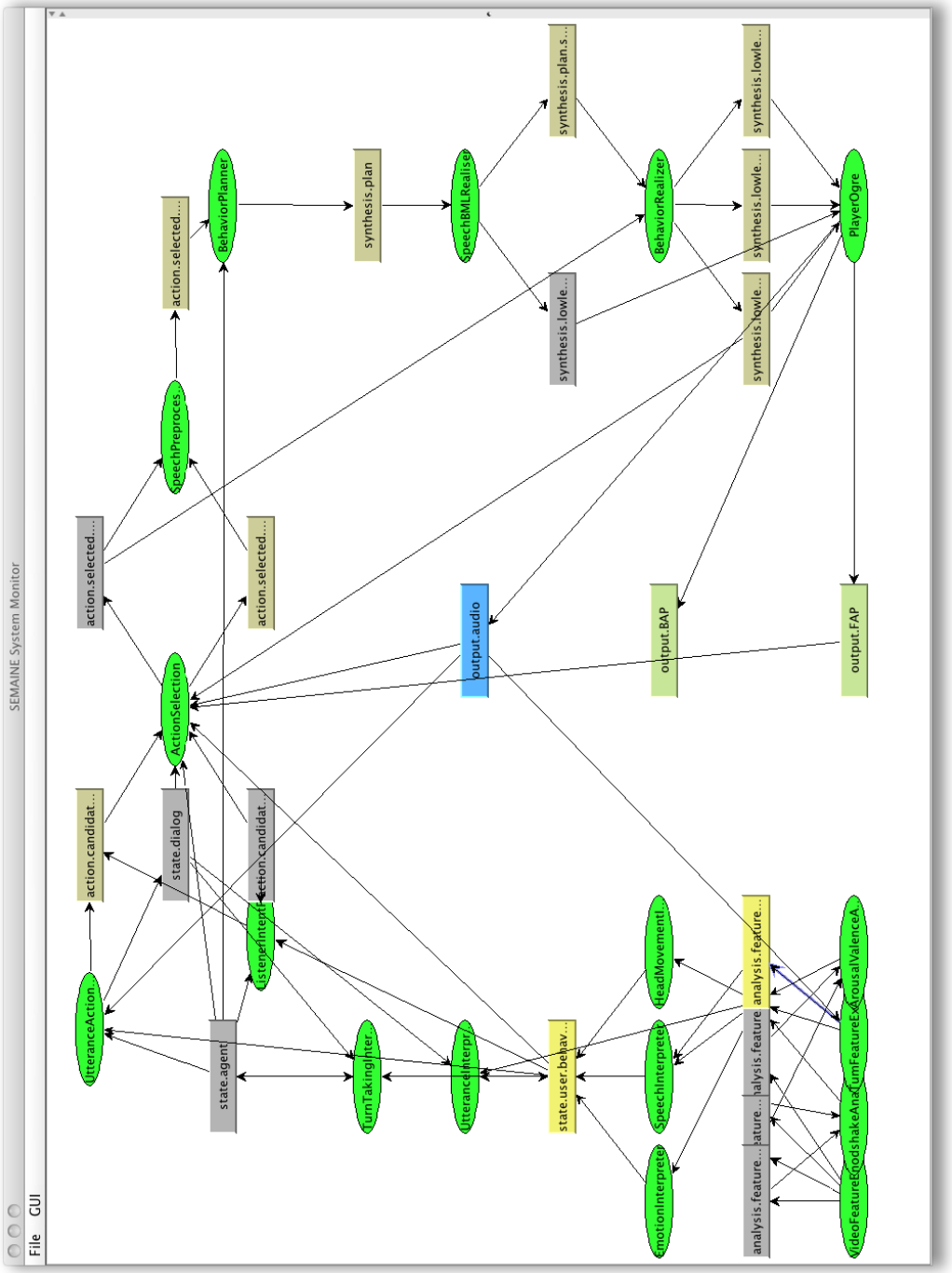

Figure 6.2: Screenshot of the System Monitor GUI showing the message flow graph for the SEMAINE system 2.0.

type as publishers to the Topic. Similarly, several components may reasonably take a given type of data as input, in which case all of them should register as subscribers to the respective Topic. Using Topics as 'information hubs' in this way immensely simplifies the clarity of information flow, and consequently the message flow graph (see Figure 6.2 for an example).

6.2.3 Components

The creation of custom components is made simple by the base class '`Component`' in the SEMAINE API, which provides the basic functionality required for the component to interact with the rest of the system. The '`Component`' will register as a subscriber and/or as a publisher to a configurable list of Topics using suitable, possibly type-specific message receivers and message senders. Whenever a message is received, the subclass's '`react()`' method is called, allowing the component to process the data and perform some action, including the emission of any output messages. In addition, the method '`act()`' is called at configurable intervals (every $100ms$ by default), allowing for actions to be triggered by component-internal mechanisms such as timeouts or custom process logic.

The '`Component`' base implementation also instantiates the meta messenger (see Figure 6.1) which handles all meta communication with the system manager, without requiring customisation by user code in subclasses.

Examples of simple component classes are provided in Chapter 11.

6.2.4 API support for relevant representation types

The SEMAINE API aims to be as easy to use as possible, while allowing for state of the art processing of data. This principle is reflected in an extensive set of support classes and methods for parsing, interpreting and creating XML documents in general and the representations specially supported (see Section 8.1) in particular. XML processing is performed by the standards-compliant parser Xerces (Apache Software Foundation, 2009) which converts between a textual representation contained in messages and a user-friendly Document Object Model (DOM) representation (Le Hors et al., 2004). Parsing is done in a namespace-aware manner in order to maintain a clear distinction between the elements used in mixed representations. Examples of mixed representations are the use of the Extensible Mul-

timodal Annotation language EMMA to transport a recognition result expressed in EmotionML, or the use of the Speech Synthesis Markup Language SSML to encode the speech aspect of ECA behaviour in the Behavior Markup Language BML. These combinations make perfect sense; namespaces are a suitable method for clearly identifying the markup type of any given element when interpreting a mixed representation.

Support classes exist for the representation formats listed in Section 8.1, notably as dedicated *receiver*, *sender* and *message* objects. For example, when a component registers an 'EmmaReceiver' to listen to a given Topic, it will receive messages directly as 'SEMAINEEmmaMessage' objects with methods appropriate for interpreting the content of the EMMA data; a 'FeatureSender' will take an array of `float` values and send it as a textual or binary feature message; 'BinarySender' and 'BinaryReceiver' classes can be used to transport, e.g., audio data between components. A generic 'XMLSender' and 'XMLReceiver' can be used to send and receive arbitrary XML structures.

In sum, these support classes and methods simplify the task of passing messages via the middleware, and help avoid errors in the process of message-passing by implementing standard encoding and decoding procedures. Where representations beyond those previewed by the API are required, the user always has the option to use lower-level methods such as plain XML or even text messages and implement a custom encoding and decoding mechanism.

Chapter 8 provides more details on the representations available for representing callback and data messages, including volatile input and output data as well as centrally held state information.

6.2.5 Supported platforms

The SEMAINE API is currently available in Java and as a shared library in C++, for Linux, Mac OS X and Windows. State of the art build tools (Eclipse and ant for Java, Visual Studio for C++ on Windows, GNU automake/ autoconf for C++ on Linux and Mac) are provided to make the use of the API as simple and portable as possible.

6.2.6 Status

As of version 3.1, the SEMAINE API is functionally complete and can be considered to be robust and stable enough for real-world use. There is detailed documentation available on the SEMAINE wiki page (`http://semaine.opendfki.de`) including full API documentation for Java (Javadoc) and C++ (Doxygen), a description of the possibilities for configuring the API and the SEMAINE system, a description of the representation formats supported (see also Chapter 8) and a tutorial for building new emotion-oriented systems with the SEMAINE API (see also Chapter 11).

The SEMAINE API starts to be used also outside the SEMAINE project, for example by Kipp et al. (2010) and by Dibeklioglu et al. (2010).

6.3 Conclusion

The present chapter has presented the technical properties of the SEMAINE API, the component integration framework developed in this thesis.

We have analysed the requirements for the underlying middleware that arise from the target system, motivating the choice of the message-oriented middleware ActiveMQ as the basis for inter-component communication in the SEMAINE API.

We have then presented the architecture of the component integration framework itself. In addition to actual communication between the components, the SEMAINE API provides a system manager functionality to keep track of the system's state, as well as a centralised logging functionality. The API provides substantial support to simplify the task of writing components, including a Component base class and utilities for sending and receiving the supported representation formats and for XML handling.

The framework is available for Java and C++, for Linux, Mac OS X and Windows. Substantial technical documentation is available.

In the next chapter, we will discuss the architecture of the actual SEMAINE system built on top of the API, before we take a closer look at inter-component communication in Part III of this thesis.

Chapter 7

System architecture of the SEMAINE system

This chapter will present the SEMAINE system with a special emphasis on system architecture and its effect on the system's reactivity. We start by presenting briefly the components making up the SEMAINE system. We then present two versions of the system. In SEMAINE-2.0, the information flow is organised as a simple pipeline architecture, where the data flows from one component's output into the next component's input in an essentially linear fashion from user input via dialogue planning to output generation. We then present SEMAINE-3.0, which implements an alternative architecture where the output part of the system is organised as a prepare-and-trigger fashion in order to speed up the time from the decision to act until the output starts playing.

7.1 Conceptual framework

Conceptually, the architecture that orients the implementation of the SEMAINE system is very similar to the architectures of other multimodal interactive systems as described in Chapter 3. Fig. 7.1 shows the main items. First, user behaviour is observed through a camera and a microphone, and low-level features are computed using a battery of *feature extractor* components. Features are low-level data computed from the raw signals, and are typically computed at a constant frame rate, e.g. every 10 ms for audio data, and every video frame for video data. These

features are used by *analyser* components to compute an estimate of the current user state. We call *analysers* such components that try to make some sense of user behaviour without using context information, such as classifiers. The raw features and the preliminary estimate of the user state are further interpreted in the light of all available information by a set of *interpreter* components. These take decisions about the system's 'current best guess' about the state of the user, the dialogue and the agent. Interpreters do such diverse things as conclude when the user is speaking or not, make a final estimate of the user's current emotion, and update the agent state such as the agent's desire to speak.

In parallel to this analysis and interpretation of the user's input, a group of *action proposers* continuously take decisions on whether to propose an action given the current state information. These include the action to speak, i.e. to produce a verbal utterance, as well as the action to exhibit some listener behaviour, such as a feedback signal or a mimicry backchannel. An *action selection* component makes sure only one action is being realised at a time. The selected action is then rendered in terms of concrete vocal, facial and gestural behaviour, and finally rendered by a *player* component.

The following section provides a description of how this conceptual framework has been instantiated in the SEMAINE-2.0 system.

7.2 SEMAINE-2.0: A pipeline architecture

The software integration in the SEMAINE project proceeded in stages. SEMAINE-1.0 represented a first integrated system with preliminary functionality; the main system infrastructure and preliminary versions of most components were in place. This "early integration" approach was designed to identify problems with the conceptual setup at an early stage, and thus simplify the integration of the full system.

SEMAINE-2.0 was then a first full instantiation of SEMAINE's Sensitive Artificial Listener system. All key functionality was implemented, and it was possible to interact with the system and have a multimodal interaction with the system. The system components were trained on preliminary data, and not optimised for performance. The system architecture was a simple pipeline architecture as depicted in Figure 7.1.

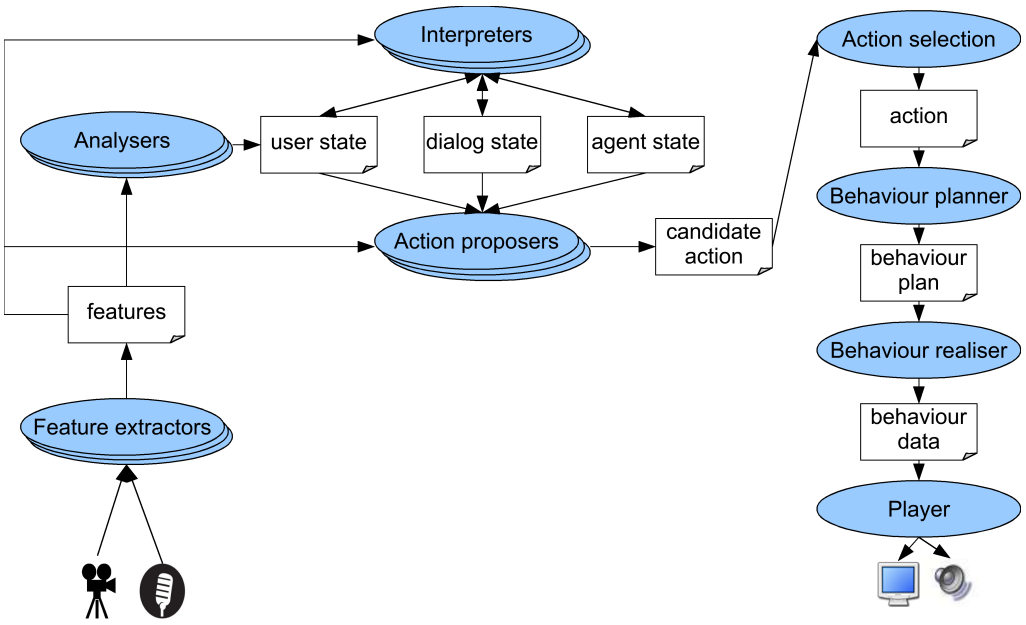

Figure 7.1: Message flow in the SEMAINE system using a pipeline architecture.

We will now describe the concrete instantiations of the components that made up the SEMAINE-2.0 system. For more technical detail on the system architecture of SEMAINE-2.0, see Schröder (2009b).

7.2.1 Feature extraction

All understanding of the user's behaviour that the SAL system may achieve starts with the extraction of low-level features characterising the user's voice, head movements, and facial expression. User behaviour is analysed from a camera and a microphone signal. The components TumFeatureExtractor and VideoFeature-Extractor compute features of the audio and video signals, respectively.

The audio feature extraction component performs, first of all, voice activity detection, in order to determine whether the user is currently speaking or not. Furthermore, the component detects keywords incrementally, and a speaker adaptation on the feature level.

The face feature extraction component starts with a face detection – the decision whether a face is present or not. This component also carries out facial

point detection and tracking, and a global head motion estimation. For details and references for further reading, see Schuller et al. (2009).

7.2.2 Understanding human behaviour

Analysers attempt to find different types of information in the low level features.

Audio analysers identify pitch variations such as rising and falling pitch, which may be useful for determining when the agent should show backchannel behaviour. The system can attempt to identify the user's gender. Non-linguistic vocalisations such as laughter and sigh are detected. And, most importantly, emotions are recognised, including interest, valence, arousal, power, anticipation, and intensity.

Face analysers detect head nods and shakes from head movements; emotions are detected in terms of arousal and valence based on facial points.

For more details, see Pantic et al. (2009a) and Pantic et al. (2009b).

7.2.3 Dialogue management

A number of interpreter, action proposer and action selection components jointly make up the dialogue management part of the system.

Interpreter components produce internal state information, which corresponds conceptually to the system's "current best guess" regarding the state of the world. We distinguish user state, agent state, dialogue state, and context state; the latter covers the relevant aspects of the current setup, such as whether a user is present, and which of the four SAL characters the user is currently interacting with. State information is communicated in the system as described in Section 8.3.

One interpreter observes head movements. If for example a head shake is detected, this may lead to an interpretation of a user state of "disagreement", which can then be taken into account during utterance selection. Other interpreters specialise in deciding whether the low-level voice activity detection is to be interpreted as the user being actually currently silent or speaking; at this level, different thresholds may be applied than for the low-level detection. An emotion interpreter consolidates the evidence of detected emotions into the system's current best guess of the user's emotion. An agent mental state interpreter takes into account the current user state and the agent's pre-defined personality, in order to

update the propensity of the agent to express different communicative functions towards the user while listening.

Maybe the most important interpreter is the turn-taking interpreter. It takes into account a number of variables to compute the agent's willingness to take the turn. Relevant factors include not only the time the user has been silent, but also recent emotions, the length of the user's turn, the agent's personality, and other factors. The resulting "propensity to take the turn" will have an impact on the likelihood of the agent selecting an utterance action.

Two main action proposers are present in the SEMAINE system: the utterance action proposer and the listener intent planner, representing the agent's actions while speaking and while listening, respectively. The utterance action proposer uses the available information to select an utterance from the existing script. The criteria for the selection differ from usual dialogue systems due to the nature of the Sensitive Artificial Listener scenario which allows for open-domain dialogues with no semantic modelling. Criteria for selecting an utterance include, for example, the user's emotion, introduction of new topics, continuation questions, or specific questions after long silences.

The listener intent planner, on the other hand, implements rules for triggering visual and/or vocal backchannel and feedback behaviour, and for determining the communicative functions to express in these behaviours.

An listener action selection component determines when to allow and when to block listener behaviour. In particular, the agent must not produce listener behaviour while it is speaking. Depending on the user's interest, the number of listener behaviours that are realised are varied.

For more details, see Heylen et al. (2009).

7.2.4 Generating SAL behaviour

Once the dialogue components have determined whether the agent is in a speaking or a listening role, and how it should act given that role, its behaviour must be realised. We use the same components for generating both speaking and listening behaviour.

In both cases, the multimodal behaviour planner needs to instantiate abstract communicative functions in its input in terms of character-specific behaviours.

For example, one character might realise "agreement" as a head nod while another character might smile, and a third character might say "yeah". A character-specific behaviour lexicon is used for this mapping.

The behaviour realiser renders the behaviour plan in terms of synthetic speech and synchronised lip movements, facial expressions, and head gestures.

The low-level behaviour specification is then realised by a 3-d audiovisual ECA player.

For details, see Pelachaud et al. (2009).

7.3 SEMAINE-3.0: Introducing the prepare-and-trigger architecture

Version 3 of the SEMAINE system represents the functionally final version of SEMAINE's SAL implementation.

The architecture of the SEMAINE-3.0 system has been extended and cleaned up compared to the previous version SEMAINE-2.0. New components were added; the structure of the architecture was homogenised on the input side; and the output side was extended with a prepare-and-trigger mechanism to speed up system responses. Full technical detail is available in Schröder et al. (2010b).

Figure 7.2 shows a schematic Message Flow Graph of the full system. Comparing it with the previous version (Figure 7.1), two main changes should be noticed:

- the input side of the architecture now includes fusion components between the analysers and the interpreters;
- the output side of the architecture distinguishes between two output branches, a direct one and a prepare-and-trigger branch.

7.3.1 Overview of changes in SEMAINE-3.0

Analysis of user behaviour

The feature extractors and analysers have been improved in SEMAINE-3.0, in terms of quality and performance. For example, a more accurate algorithm for keyword spotting was integrated into the real-time system. Facial analysis now

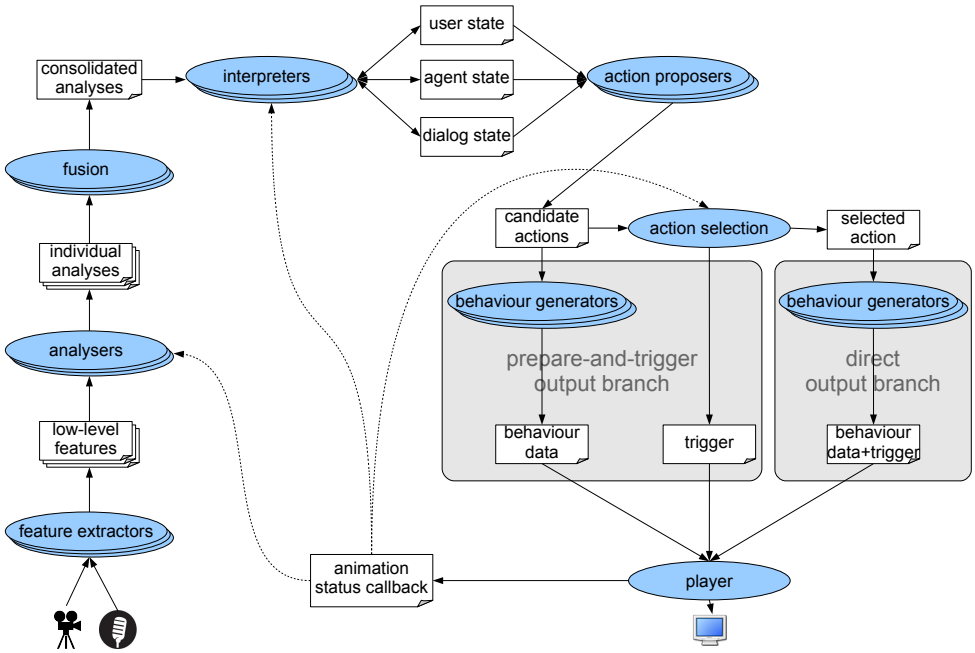

Figure 7.2: Message flow in the SEMAINE system using a prepare-and-trigger architecture.

includes an action unit detector. Emotion is now predicted from voice, face, and head movements.

New fusion components have been added to make explicit the step of consolidating evidence from different modalities regarding non-verbal behaviour and emotion detection. For example, given the fact that arousal is usually more reliable conveyed through the voice and valence is easier to detect from the face, the fusion can, in case of conflict, give preference to the voice-based prediction of arousal but use the face-based prediction of valence.

Details can be found in Schuller et al. (2010) and Pantic et al. (2010).

Dialogue management

The dialogue management components have been improved with respect to the reusability of the components: a template system is now used to define the dialogue scripts, which means that new dialogues can be scripted in XML without having to modify any program code. The utterance scripts have been enriched with non-verbal behaviours, so that the SAL characters are now expressive also while speaking, not only while listening. A new data-driven criterion for selecting utterances has been added, based on machine learning from annotated dialogue scripts.

The utterance action proposer has been adapted to support the new prepare-and-trigger architecture (see below).

For details on dialogue management in SEMAINE-3.0, see Heylen and ter Maat (2010).

Behaviour generation

For the behaviour generation, there have been a number of improvements in detail, such as a better coverage of audio-visual listener behaviour (such as nodding and saying "uh-huh" at the same time). The most important change, however, concerns the architecture which complements direct generation with a prepare-and-trigger branch.

The basic idea is to separate the preparation of an action from its output in the player. This allows us to prepare *likely* future actions ahead of time, and to *trigger* their immediate start when they are indeed selected. We call this the "prepare-

and-trigger" branch in the SEMAINE-3.0 architecture. When actions are selected which have not been prepared ahead of time, they are realised using the traditional "direct" branch.

A protocol for the player component was defined to ensure that both the direct branch and the prepare-and-trigger branch can feed the player without causing conflicting states. The definition of the player protocol is provided in Appendix A.

Details on the ECA behaviour generation in SEMAINE-3.0 are available in Pelachaud et al. (2010).

7.3.2 Motivation for the prepare-and-trigger architecture: Competitive Queuing

The idea for the prepare-and-trigger architecture comes from cognitive neuroscience, more precisely the notion of Competitive Queuing (CQ) in fluent series of actions (Bullock, 2004). Traditional action sequence models would consider a transition from states where in each state, a single action is activated. In CQ, on the other hand, multiple actions which form a sequence are active in parallel; the action with the highest activation level is executed next, and then deleted from the set of active actions. These models are used to explain, for example, anticipatory behaviour and also re-sequencing errors in well-learned sequences, such as typing familiar words on a computer keyboard. The anticipation of actions makes it possible to execute sequences of actions fluently.

Bullock and Rhodes (2003) point out the key properties of architectures based on CQ as follows.

> "competitive queuing models [...] follow naturally from two assumptions: (1) More than one plan representation can be simultaneously active in a planning layer; and (2) The most active plan representation is chosen, in a second neural layer, by a competition run to decide which plan to enact next." (Bullock and Rhodes, 2003, p. 241)

In order to benefit from the ideas of CQ in the context of our ECA system, we would therefore have to modify our architecture as follows.

- "Activate" predictable future actions ahead of time, without triggering them.

- Activate multiple potential actions.
- Establish a fast mechanism for triggering activated actions.

In our case, the "activation" of an action consists in rendering it through the sequence of steps needed to get from the action, as produced by the utterance action proposer, to the player input. In other words, rather than first selecting an action and then starting to render it, the CQ-based method consists in first *preparing* a number of likely actions, and upon selection of one of these actions, *triggering* its immediate realisation.

It is important to note that the CQ approach is designed to make more fluent the generation of *predictable* actions. This means that even an architecture based on CQ principles must retain the possibility to generate actions which were not predicted beforehand, such as reactions to unexpected events. In SEMAINE, we have realised this by *complementing* the direct pipeline architecture with a "prepare-and-trigger" branch. The idea is to generate as many actions as possible via the prepare-and-trigger branch, and to use the direct branch as a fallback.

The following sections show in detail how we have realised the approach. We start with a closer look at the pipeline approach, so that it is easier to see the difference to the new prepare-and-trigger approach.

7.3.3 Information flow for output generation in the pipeline architecture

In the pipeline approach to output generation, information flows in a simple sequence from the action selection to the player. Conceptually this sequence is the same in the SEMAINE-2.0 system (Figure 7.1) and in the "direct branch" in SEMAINE-3.0 (Figure 7.2). The following detailed description follows the SEMAINE-3.0 system in order to be consistent with the description of the prepare-and-trigger approach.

Starting with an utterance selected by the dialogue components, the following steps are performed in sequence before the player can start executing the animation.

- The component SpeechPreprocessor computes the accented syllables and any phrase boundaries as anchors to which any gestural behaviour can

be attached. It reads from Topics 'semaine.data.action.selected.function' and 'semaine.data.action.selected.behaviour', and writes its results to 'semaine.data.action.selected.speechpreprocessed'. Whereas conceptually this processing step could work with purely symbolic specification of prosody-based anchor points, the current implementation of the behaviour planner component requires absolute timing information. For this reason the output of the SpeechPreprocessor already contains the detailed timing information.

- BehaviourPlanner determines suitable behaviour elements based on the intended function/meaning of the action. It uses character-specific behaviour lexicons to map FML to BML (see Chapter 8 for details on these representation formats). It reads from 'semaine.data.action.selected.speechpreprocessed' and writes to 'semaine.data.synthesis.plan'.

- SpeechBMLRealiser carries out the actual speech synthesis, i.e. the generation of audio data. It reads from 'semaine.data.synthesis.plan'; it writes a BML message including the speech timings to 'semaine.data.synthesis.plan.speechtimings', and the binary audio data including a file header to 'semaine.data.synthesis.lowlevel.audio'.

- BehaviorRealizer reads from 'semaine.data.synthesis.plan.speechtimings' and 'semaine.data.action.selected.behaviour', and produces the low-level video features in Topics 'semaine.data.synthesis.lowlevel.video.FAP' and 'semaine.data.synthesis.lowlevel.video.BAP'. In addition, it sends two types of information to Topic 'semaine.data.synthesis.lowlevel.command': (1) the information which modalities form part of a given animation as identified by a unique content ID; and (2) the trigger commands needed to start playing back the animation.

- PlayerOgre is the audiovisual player component. It reads the low-level player data from topics 'semaine.data.synthesis.lowlevel.audio', 'semaine.data.synthesis.lowlevel.video.FAP' and 'semaine.data.synthesis.lowlevel.video.BAP'. A unique content ID is used to match the various parts of a multimodal animation to be rendered. Two types

of information are received via the Topic 'semaine.data.synthesis.lowlevel.command': the information which modalities are expected to be part of a given animation / content ID; and the trigger to start playing the animation. The Player sends callback messages (see Section 8.2) to the topic 'semaine.callback.output.Animation', to inform about the preparation or playback state of the various animations it receives.

7.3.4 Information flow for output generation in the prepare-and-trigger architecture

The prepare-and-trigger branch in the SEMAINE-3.0 system (Figure 7.2) replicates the middle part of the processing pipeline of the direct branch, but using different Topics.

- QueuingSpeechPreprocessor reads from 'semaine.data.action.prepare.function' and 'semaine.data.action.prepare.behaviour', and writes to 'semaine.data.action.prepare.speechpreprocessed';
- BehaviorPlannerPrep reads from 'semaine.data.action.prepare.speechpreprocessed' and writes to 'semaine.data.synthesis.prepare';
- QueuingSpeechBMLRealiser reads from 'semaine.data.synthesis.prepare' and writes to 'semaine.data.synthesis.prepare.speechtimings' and 'semaine.data.synthesis.lowlevel.audio';
- BehaviorRealizerPrep reads from 'semaine.data.synthesis.prepare.speechtimings' and 'semaine.data.action.prepare.behaviour' and writes to 'semaine.data.synthesis.lowlevel.video.FAP', 'semaine.data.synthesis.lowlevel.video.BAP' and 'semaine.data.synthesis.lowlevel.command'.

The difference to the direct branch is at the two ends of the processing pipeline. At the input end, the UtteranceActionProposer feeds into 'semaine.data.prepare.function' candidate utterances that the current character may perform in the near future. At the output end, the BehaviorRealizerPrep sends the data

and the content-level description to the player, but it does not send the trigger commands. Instead, when the player has received all necessary parts of a given animation, it sends a "ready" callback message which is then registered by the `UtteranceActionProposer`. When its utterance selection algorithm determines that the selected utterance already exists in prepared form in the player, all it needs to do is send a trigger command directly from `UtteranceActionPoposer` to '`semaine.data.synthesis.lowlevel.command`', which then starts the playback of the prepared animation without any further delay. If no prepared version of the selected utterance is available, e.g. because it was not expected that this utterance would be selected, or because the preparation has not yet been completed, the utterance is generated using the direct branch.

The prepare-and-trigger branch is used only for full utterances. Listener actions are so short and fast to generate that they always use the direct branch. Since both branches are technically completely independent, this architecture scales well to multiple computers: it is easy to run the direct branch on one computer and the prepare-and-trigger branch on a different computer if they jointly would overstretch the CPU resources of a single PC.

7.4 Conclusion

This chapter has described the components that make up the SEMAINE system implementing a set of Sensitive Artificial Listeners. We have only touched upon the individual components very briefly, pointing to relevant references for the interested reader. We have described two versions of the system, the first full system SEMAINE-2.0 organised according to the traditional "pipeline" architecture, and the latest version SEMAINE-3.0 adding the prepare-and-trigger branch inspired by competitive queuing in order to speed up the reaction time for predictable actions. We will assess the effectiveness of this approach in Chapter 10.

In the next chapter, we will provide information on the representations that are used in the communication between the components.

Part III

Communication

Chapter 8

Representation formats

In this chapter, we present the representation formats used in the SEMAINE system and supported in the SEMAINE API. We present their status with respect to standardisation, and the extent to which domain-specific representations appear to be needed.

8.1 Representation formats supported in the SEMAINE API

In view of future interoperability and reuse of components, the SEMAINE API aims to use standard representation formats where that seems possible and reasonable. For example, results of analysis components can be represented using EMMA (Extensible Multi-Modal Annotation), a Wold Wide Web Consortium (W3C) Recommendation (Johnston et al., 2009). Input to a speech synthesiser can be represented using SSML (Speech Synthesis Markup Language), also a W3C Recommendation (Burnett et al., 2004).

Several other relevant representation formats are not yet standardised, but are in the process of being specified. This includes the Emotion Markup Language EmotionML (see Chapter 9), used for representing emotions and related states in a broad range of contexts, and the Behaviour Markup Language BML (Kopp et al., 2006), which describes the behaviour to be shown by an Embodied Conversational Agent (ECA). Furthermore, a Functional Markup Language FML (Heylen et al., 2008) is under discussion, in order to represent the planned actions of an ECA on

the level of functions and meanings. By implementing draft versions of these specifications, the SEMAINE API can provide hands-on input to the standardisation process, which may contribute to better standard formats.

On the other hand, it seems difficult to define a standard format for representing the concepts inherent in a given application's logic. To be generic, such an endeavour would ultimately require an ontology of the world. In the SEMAINE system, which does not aim at any sophisticated reasoning over domain knowledge, a simple custom format named SemaineML is used to represent those pieces of information that are required in the system but which cannot be adequately represented in an existing or emerging standard format. It is conceivable that other applications built on top of the SEMAINE API may want to use a more sophisticated representation such as the Rich Description Format RDF (Becket and McBride, 2004) to represent domain knowledge, in which case the API could be extended accordingly.

Whereas all of the aforementioned representation formats are based on the Extensible Markup Language XML (Bray et al., 1998), there are a number of data types that are naturally represented in different formats. This is particularly the case for the representations of data close to input and output components. At the input end, low-level analyses of human behaviour are often represented as feature vectors. At the output end, the input to a player component is likely to include binary audio data or player-specific rendering directives.

Table 8.1 gives an overview of the representation formats currently supported in the SEMAINE API. The following subsections briefly describe the individual representation formats.

8.1.1 Feature vectors

Feature vectors can be represented in an ad hoc format. In text form (see Figure 8.1), the feature vectors consist of straightforward key-value pairs – one feature per line, values preceding features.

As feature vectors may be sent very frequently (e.g., every 10 ms in the SEMAINE system), compact representation is a relevant issue. For this reason, a binary representation of feature vectors is also available. In binary form, the feature names are omitted, and only feature values are being communicated. The first

Type of data	Representation format	Standardisation status
Low-level input features	string or binary feature vectors	ad hoc
Analysis results	EMMA	W3C Recommendation
Emotions and related states	EmotionML	W3C Working Draft
Domain knowledge	SemaineML	ad hoc
Speech synthesis input	SSML	W3C Recommendation
Functional action plan	FML	very preliminary
Behavioural action plan	BML	draft specification
Low-level output data	binary audio, player commands	player-dependent

Table 8.1: Representation formats supported by the SEMAINE API.

```
0.000860535 rmsEnergy
12.6699 logEnergy
-2.59005e-05 rmsEnergy-De
-0.0809427 logEnergy-De
...
```

Figure 8.1: Textual representation of a feature vector.

four bytes represent an integer containing the number of features in the vector; the remaining bytes contain the float values one after the other.

8.1.2 EMMA

The Extensible Multimodal Annotation Language EMMA, a W3C Recommendation, is "an XML markup language for containing and annotating the interpretation of user input" (Johnston et al., 2009). As such, it is a wrapper language that can carry various kinds of payload representing the interpretation of user input. The EMMA language itself provides, as its core, the `<emma:interpretation>` element, containing all information about a single interpretation of user behaviour. Several such elements can be enclosed within an `<emma:one-of>` element in cases where more than one interpretation is present. An interpretation can have an `emma:confidence` attribute, indicating how confident the source of the annotation is that the interpretation is correct; time-related information such as `emma:start`, `emma:end`, and `emma:duration`, indicating the time span for which the interpretation is provided; information about the modality upon which the interpretation is based, through the `emma:medium` and `emma:mode` attributes; and many more.

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma"
        xmlns="http://www.w3.org/2009/10/emotionml">
  <emma:interpretation emma:start="12457990" emma:end="12457995"
                       emma:mode="voice" emma:verbal="false">

    <emotion dimension-set="http://www.w3.org/TR/emotion-voc/xml#fsre-dimensions">
      <dimension name="arousal" value="0.3"/>
      <dimension name="valence" value="0.7"/>
    </emotion>

  </emma:interpretation>
</emma:emma>
```

Figure 8.2: An example EMMA document carrying EmotionML markup as interpretation payload.

Figure 8.2 shows an example EMMA document carrying an interpretation of user behaviour represented using EmotionML (see Chapter 9). The interpretation refers to a start time. It can be seen that the EMMA wrapper elements and the EmotionML content are in different XML namespaces, so that it is unambiguously determined which element belongs to which part of the annotation.

EMMA can also be used to represent Automatic Speech Recognition (ASR) output, either as the single most probable word chain or as a word lattice, using the `<emma:lattice>` element.

8.1.3 EmotionML

The Emotion Markup Language EmotionML is specified, at the time of this writing, by a W3C Last Call working draft (see Chapter 9 for details).

The SEMAINE API is one of the first pieces of software to implement EmotionML. It is our intention to provide an implementation report as input to the W3C standardisation process in due course, highlighting any problems encountered with the current draft specification in the implementation.

EmotionML aims to make concepts from major emotion theories available in a broad range of technological contexts. Being informed by the affective sciences, EmotionML recognises the fact that there is no single agreed representation of affective states, nor of vocabularies to use. Therefore, an emotional state `<emotion>`

```
<semaine:dialog-state xmlns:semaine="http://www.semaine-project.eu/semaineml">
        <semaine:agent believesHasTurn="false"/>
</semaine:dialog-state>
```

Figure 8.3: An example SemaineML document representing dialogue state.

can be characterised using four types of descriptions: <category>, <dimension>, <appraisal> and <action-tendency>. Furthermore, the vocabulary used can be identified. The EmotionML markup in Figure 8.2 uses a dimensional representation of emotions, annotating the two dimensions arousal and valence.

EmotionML is aimed at three use cases: 1. Human annotation of emotion-related data; 2. automatic emotion recognition; and 3. generation of emotional system behaviour. In order to be suitable for all three domains, EmotionML is conceived as a "plug-in" language that can be used in different contexts. In the SEMAINE API, this plug-in nature is applied with respect to recognition, centrally held information, and generation, where EmotionML is used in conjunction with different markups. EmotionML can be used for representing the user emotion currently estimated from user behaviour, as payload to an EMMA message. It is also suitable for representing the centrally held information about the user state, the system's "current best guess" of the user state independently of the analysis of current behaviour. Furthermore, the emotion to be expressed by the system can also be represented by EmotionML. In this case, it is necessary to combine EmotionML with the output languages FML, BML and SSML.

8.1.4 SemaineML

A number of custom representations are needed to represent the kinds of information that play a role in the SEMAINE demonstrator systems. Currently, this includes the centrally held beliefs about the user state, the agent state, the dialogue state, and the context state. Most of the information represented here is domain-specific and does not lend itself to easy generalisation or reuse. Figure 8.3 shows an example of a dialogue state representation, focused on the specific situation of an agent-user dialogue as targeted in the SEMAINE system (see Chapter 7).

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
    xml:lang="en-US">
  <voice gender="female">
    And then <break/> I <emphasis>wanted</emphasis> to go.
  </voice>
</speak>
```

Figure 8.4: An example standalone SSML document.

The SEMAINE API comes with a generic mechanism for defining custom XML structures and for making the information available to system components as "state information" via a unique keyword. Section 8.3 describes this mechanism.

8.1.5 SSML

The Speech Synthesis Markup Language SSML (Burnett et al., 2004) is a well-established W3C Recommendation supported by a range of commercial text-to-speech (TTS) systems. It is the most established of the representation formats described in this chapter.

The main purpose of SSML is to provide information to a TTS system on how to speak a given text. This includes the possibility to add <emphasis> on certain words, to provide pronunciation hints via a <say-as> tag, to select a <voice> which is to be used for speaking the text, or to request a <break> at a certain point in the text. Furthermore, SSML provides the possibility to set markers via the SSML <mark> tag. Figure 8.4 shows an example SSML document that could be used as input to a TTS engine. It requests a female US English voice; the word "wanted" should be emphasised, and there should be a pause after "then".

8.1.6 FML

The functional markup language FML is still under discussion (Heylen et al., 2008). Its functionality being needed nevertheless, a working language FML-APML was created (Mancini and Pelachaud, 2008) as a combination of the ideas of FML with the former Affective Presentation Markup Language APML (de Carolis et al., 2004).

```
<fml-apml version="0.1">
  <bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
    <speech id="s1" language="en-US" text="Hi, I'm Poppy."
        ssml:xmlns="http://www.w3.org/2001/10/synthesis">
         <ssml:mark name="s1:tm1"/>
           Hi,
           <ssml:mark name="s1:tm2"/>
           I'm
           <ssml:mark name="s1:tm3"/>
           Poppy.
           <ssml:mark name="s1:tm4"/>
    </speech>
  </bml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml1">
    <performative id="p2" type="announce" start="s1:tm1" end="s1:tm4"/>
    <world id="w1" ref_type="person" ref_id="self"
           start="s1:tm2" end="s1:tm4"/>
  </fml>
</fml-apml>
```

Figure 8.5: An example FML-APML document.

Figure 8.5 shows an example FML-APML document which contains the key elements. An <fml-apml> document contains a <bml> section in which the <speech> content contains <ssml:mark> markers identifying points in time in a symbolic way. An <fml> section then refers to those points in time to represent the fact, in this case, that an announcement is made and that the speaker herself is being referred to between marks s1:tm2 and s1:tm4. This information can be used, for example, to generate relevant gestures when producing behaviour from the functional descriptions.

The representations in the <fml> section are provisional and are likely to change as consensus is formed in the community.

For the conversion from FML to BML, information about pitch accents and boundaries is useful for the prediction of plausible behaviour time-aligned with the macro-structure of speech. In our current implementation, a speech preprocessor computes this information using TTS technology (see Chapter 7). The information is added to the end of the <speech> section as shown in Figure 8.6. This is an ad hoc solution which should be reconsidered in the process of specifying FML.

```
<fml-apml version="0.1">
  <bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
    <speech id="s1" language="en_US" text="Hi, I'm Poppy."
        ssml:xmlns="http://www.w3.org/2001/10/synthesis">
        <ssml:mark name="s1:tm1"/>
        Hi,
        <ssml:mark name="s1:tm2"/>
        I'm
        <ssml:mark name="s1:tm3"/>
        Poppy.
        <ssml:mark name="s1:tm4"/>
        <pitchaccent id="xpa1" start="s1:tm1" end="s1:tm2"/>
        <pitchaccent id="xpa2" start="s1:tm3" end="s1:tm4"/>
        <boundary id="b1" time="s1:tm4"/>
    </speech>
  </bml>
  <fml xmlns="http://www.mindmakers.org/fml" id="fml1">
    <performative id="p2" type="announce" start="s1:tm1" end="s1:tm4"/>
    <world id="w1" ref_type="person" ref_id="self"
           start="s1:tm2" end="s1:tm4"/>
  </fml>
</fml-apml>
```

Figure 8.6: Pitch accent and boundary information added to the FML-APML document of Figure 8.5.

```
<bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
  <speech id="s1" language="en_US" text="Hi, I'm Poppy."
      ssml:xmlns="http://www.w3.org/2001/10/synthesis">
    <ssml:mark name="s1:tm1"/>
    Hi,
    <ssml:mark name="s1:tm2"/>
    I'm
    <ssml:mark name="s1:tm3"/>
     Poppy.
    <ssml:mark name="s1:tm4"/>
    <pitchaccent id="xpa1" start="s1:tm1" end="s1:tm2"/>
    <pitchaccent id="xpa2" start="s1:tm3" end="s1:tm4"/>
    <boundary id="b1" time="s1:tm4"/>
  </speech>
  <gaze id="g1" start="s1:tm1" end="s1:tm4">
    ...
  </gaze>
  <head id="h1" start="s1:tm3" end="s1:tm4" type="NOD">
    ...
  </head>
</bml>
```

Figure 8.7: An example BML document containing SSML and gestural markup.

8.1.7 BML

The aim of the Behaviour Markup Language BML (Kopp et al., 2006) is to represent the behaviour to be realised by an Embodied Conversational Agent. BML is at a relatively concrete level of specification, but is still in draft status (Mindmakers, 2008).

A standalone BML document is partly similar to the <bml> section of an FML-APML document (see Figure 8.5); however, whereas the <bml> section of FML-APML contains only a <speech> tag, a BML document can contain elements representing expressive behaviour for the ECA at a broad range of levels, including <head>, <face>, <gaze>, <body>, <speech> and others. Figure 8.7 shows an example of gaze and head nod behaviour added to the example of Figure 8.6.

While creating an audio-visual rendition of the BML document, we use TTS to produce the audio and the timing information needed for lip synchronisation. Whereas BML in principle previews a <lip> element for representing this information, we are uncertain how to represent exact timing information with it in a

```
<bml xmlns="http://www.mindmakers.org/projects/BML" id="bml1">
  <speech id="s1" language="en_US" text="Hi, I'm Poppy."
      ssml:xmlns="http://www.w3.org/2001/10/synthesis"
      mary:xmlns="http://mary.dfki.de/2002/MaryXML">
    ...
    <ssml:mark name="s1:tm3"/>
    Poppy.
    <mary:syllable stress="1">
      <mary:ph d="0.092" end="1.011" p="p"/>
      <mary:ph d="0.112" end="1.123" p="A"/>
      <mary:ph d="0.093" end="1.216" p="p"/>
    </mary:syllable>
    <mary:syllable>
      <mary:ph d="0.141" end="1.357" p="i"/>
    </mary:syllable>
    <ssml:mark name="s1:tm4"/>
   ...
</bml>
```

Figure 8.8: An excerpt of a BML document enriched with TTS timing information for lip synchronisation.

way that preserves the information about syllable structure and stressed syllables. For this reason, we currently use a custom representation based on the MaryXML format from the MARY TTS system (Schröder et al., 2009b) to represent the exact timing of speech sounds. Figure 8.8 shows the timing information for the word "Poppy", which is a two-syllable word of which the first one is the stressed syllable.

The custom format we use for representing timing information for lip synchronisation clearly deserves to be revised towards a general BML syntax, as BML evolves.

8.1.8 Player data

Player data is currently treated as unparsed data. Audio data is binary, whereas player directives are considered to be plain text. This works well with the MPEG-4 player we use (see Chapter 7) but may need to generalised as other players are integrated into the system.

```
<callback xmlns="http://www.semaine-project.eu/semaineml">
  <event data="Animation" id="fml_lip_70" time="1116220"
         type="start" contentType="utterance"/>
</callback>
```

Figure 8.9: A callback message sent by the Player component to indicate the start of an animation representing an utterance.

8.2 Callback messages

Callbacks are required to inform other components about the processing state of a certain piece of information. Specifically, the player needs to send callback messages indicating when it is starting and ending the playback of an animation.

This information is used in several places in the system, in particular:

- the speech analysis components use the messages regarding start and end of audio playback to prevent voice activity detection while the agent is speaking. This is useful so that the system does not reply to itself;
- the dialogue components assign a unique identifier to an utterance that they request to be played. The player's callback messages confirm that a given utterance has been played, and thus allow for a smooth follow-on. In particular, the ActionSelection component can avoid sending additional playback requests while the system is still talking, which would interrupt the current system output.

Callback messages are sent via their own Topic hierarchy. Whereas data messages pass via Topics below '`semaine.data`', callback messages are sent to Topics in the hierarchy '`semaine.callback`'. This way, it is possible to structurally distinguish the two types of messages very easily. This is also reflected in the layout of the System Monitor GUI, as explained in Chapter 6.

Figure 8.9 shows an example of a callback message sent to the Topic '`semaine.callback.outout.Animation`'. It contains information about the time and type of event, the unique identifier of the data concerned, and the content type of that data. Possible event types are "ready", "start", "end", as well as "stopped" and "deleted". Possible content types are "utterance", "listener-vocalisation" and "visual-only".

8.3 A mechanism for defining state information

The SEMAINE system needs to maintain various kinds of state information: the context, such as the current character and the information whether or not there is a user present; the agent's mental state; the agent's interpretation of the user's behavioural and emotional state; and the state of the dialogue.

In a Message-Oriented Middleware (MOM) it is not straightforward to preserve a centrally held state in such a way that several components can access it, because a MOM does not provide a shared memory. One option for implementing distributed access to state information would have been a specialised information repository component; however, this would have required each component to send and receive a message every time that it wants to access a certain information.

The solution implemented in the SEMAINE API is of a different kind. A specialised class State-Receiver is keeping track of state-related messages; it parses incoming messages and saves the information in a local information repository. That means, every component has local access to its own copy of the latest state information. Since the local copies are updated through the same state-related messages, they are updated in synchrony. Looking up a certain piece of information is thus as easy as accessing a local variable.

An important challenge in this kind of setup is to make the encoding-decoding link between the representation of information in XML-based messages, and a unique "short name" by which components can access the information. For example, the dialogue state information that the agent does not have the turn at the moment is encoded as shown in Figure 8.3; if components had to parse an XML structure for every information item, this would be complicated and error-prone.

Instead, components should be able to access the information independently of the representation format, through a unique short name such as '`agentTurnState`'.

If the link between message format and short name were hard-coded in the program code, it would not allow users to flexibly reuse the state information mechanism in novel domains and applications, which would be incompatible with the SEMAINE API's ambition to be a reusable framework. Therefore, we have developed and implemented a mechanism that allows developers to represent this relationship in a configuration file, using namespace-aware XPath expressions (Berglund et al., 2007). XPath is a formalism for navigating through

XML documents to access information. In the example in Figure 8.3, the information whether or not the agent believes it has the turn can be accessed by going to the element <dialog-state> in the namespace http://www.semaine-project.eu/semaineml, then to the child element <agent> in the same namespace, and finally by accessing the value of the attribute believesHasTurn. In XPath, this can be encoded as:

```
/semaine:dialog-state/semaine:user/@believesHasTurn
```

where the namespace prefix 'semaine' is bound to the namespace 'http://www.semaine-project.eu/semaineml'. By associating this XPath expression with a short name 'agentTurnState', it is possible to provide the relation in a configuration file.

XPath in its general form is a very powerful framework which is intended only for accessing information in existing XML documents. It is not originally intended to be used for the generation of documents. However, by using only the subset of navigating to elements and accessing attribute values or textual content, we can reuse the XPath expressions also for the generation of XML documents and thus for encoding the information.

As a result, we have a fully configurable mechanism for encoding and decoding state information. Both the relation between short names and XPath expressions and the namespace prefixes can be given in the configuration file, providing full flexibility for future extensions. Figure 8.10 shows an excerpt of the respective configuration file. The latest version of the file itself can be found at http://semaine.opendfki.de/browser/trunk/java/config/stateinfo.config; documentation describing the meaning of attributes and legal values is available at http://semaine.opendfki.de/wiki/StateInfo.

8.4 Conclusion

The present chapter has presented three types of representations that are needed in a system such as the SEMAINE system and are therefore supported by the SEMAINE API. First, there are the representations of data communicated within the system, such as intermediate processing results of analysing user input or of generating system behaviour. We have argued that research can benefit from using

```
[DialogState]
[namespace prefixes]
semaine http://www.semaine-project.eu/semaineml

[short names]
userTurnState    /semaine:dialog-state/semaine:user/@believesHasTurn
agentTurnState  /semaine:dialog-state/semaine:agent/@believesHasTurn
convState       /semaine:dialog-state/semaine:agent/@convState
```

Figure 8.10: The DialogState section of the state information configuration file used in the SEMAINE-3.0 system.

standard representations to ease re-use of components to build new systems. In part, suitable standard representations exist, such as EMMA and SSML; other representations are in the process of standardisation, such as EmotionML and BML.

A second type of message, conceptually distinct from the data flow in the system, are the callback messages, by means of which one component can inform other components about the current processing state of an identified item.

The third type of representation is highly domain-specific: the state of the user, the agent, the dialogue and the context. Here we have presented a generic mechanism for reconciling the need for components to have simple access to information via a short identifier with the volatile nature of message-oriented middleware. A configuration file defines the available information and how it is to be encoded in messages. It does so by providing a mapping between namespace-aware XPath-structures, representing the message format, and short names under which the information is accessed by components.

In the next chapter, we will provide a detailed and in-depth presentation of EmotionML, the Emotion Markup Language, which is of central importance for creating emotion-oriented systems in general and for the present real-time ECA system in particular.

Chapter 9

Emotion Markup Language

One of the representation formats described in Chapter 8 is EmotionML, the Emotion Markup Language – a format for representing emotions for use in technological systems. It is obvious that computerised systems, to the extent that they can recognise, simulate or otherwise process emotion-related information, need a representation format. If several components are to work collaboratively on the information, the format must be well-defined. In order to reach the best possible interoperability, a standard representation format should be used.

This chapter describes the work, in a long-running collaborative effort spearheaded by the author, on defining and standardising EmotionML. We start with an outline of the process involved in standardising a language such as this one. We then build up the logic of the language, starting from use cases and resulting requirements, and comparing that with scientific representations of emotion. We describe the syntax of EmotionML in the light of the requirements and discuss its relation to the aim of scientific validity.

The chapter ends with two comments, one on an alternative solution to the syntax which was discarded for reasons of common practice, but which still seems worth reporting; the other making explicit a number of relevant aspects of emotion that are not covered by the current specification.

9.1 The process of defining a standard Emotion Markup Language

The definition of a standard representation format such as EmotionML is a long-term effort with a large number of intermediate steps. In this section we briefly identify the main milestones in order to give the reader a sense of the process.

Work on EmotionML started informally in 2006, as a joint effort in the HUMAINE Network of Excellence on emotion-oriented technology. The initial thoughts are reflected in a publication on an "Emotion Annotation and Representation Language (EARL)" (Schröder et al., 2006). At that time, with a small group of people we tried to cover the full ground in a short time, spanning the range from scientific descriptions of emotions, via use cases and requirements of technological applications, to the definition of an XML syntax.

This initial spark of interest, and positive feedback from the community, led us to initiate in 2007 a so-called "Incubator" group on the topic at the World Wide Web Consortium (W3C). Incubator groups can be set up with fewer formal thresholds than full Working Groups. They allow for "experimental" work on topics for which it is not yet clear whether the work should lead to a standard; their lifetime is limited to one year.

In the Emotion Incubator group, thus, interested parties from all over the world came together to discuss the use cases for an emotion markup language and the resulting requirements. The final report of the Emotion Incubator group (Schröder et al., 2007b) listed 39 individual use cases grouped into three types (see below), and identified a structured set of requirements resulting from these use cases. The report also assessed pre-existing markup languages and concluded that none is available that fully addresses the requirements. The joint work was also reflected by a joint publication at the Affective Computing conference (Schröder et al., 2007a).

A second Incubator group was set up in 2008 to work more specifically on the possible syntax of an Emotion Markup Language. The group started by prioritising the requirements (Burkhardt and Schröder, 2008; Schröder et al., 2008b), and only then started working on a syntax for the language itself. By the end of the one-year lifetime of this second Incubator group, key ideas for the syntax of EmotionML were in place, and open issues were identified as such (Schröder et al., 2008a). The

group at that point also decided that it seemed worthwhile to work on producing a formal specification of EmotionML in the so-called "Recommendation Track" at the W3C.

This formal work started in 2009, as part of the W3C's existing Multimodal Interaction (MMI) Working Group. A First Public Working Draft (FPWD) of EmotionML 1.0 was published in 2009, followed by a second Working Draft in 2010 (Schröder et al., 2010a). The specification process consisted mainly in resolving the open issues in the second Incubator report, in making the syntactic choices compatible with other works in W3C and in the MMI group, and in ensuring that the syntax was sufficiently simple to be usable in real-world settings.

A W3C workshop on EmotionML was organised in October 2010 (`http://www.w3.org/2010/10/emotionml/cfp.html`) to invite feedback on the draft specification from scientific experts as well as from potential users. The workshop provided highly relevant feedback and clarification, and played an important role in the definition of the full specification published as a Last Call Working Draft (LCWD) in spring 2011 (Schröder et al., 2011a). The definition of a number of vocabularies for EmotionML was published as a separate W3C Working Draft (Schröder et al., 2011b) at the same time as the LCWD.

After the publication of the Last Call Working Draft, a number of steps remain to be done before the final Recommendation status is reached. At LCWD stage, the specification is thoroughly reviewed by all interested W3C members. All feedback must be formally addressed. If changes to the specification are needed, a new LCWD must be published. Otherwise, the process continues towards the Candidate Recommendation, which must include a number of implementation reports of the specification. The purpose of these reports is to verify the implementability of the specification. If no major problems occur at this stage, the specification then moves via the stage of Proposed Recommendation to the final stage of Recommendation.

9.2 Previous work

The representation of emotions and related states has been part of several activities.

In the area of labelling schemes, maybe the most thorough attempt to propose an encompassing labelling scheme for emotion-related phenomena has been

the work on the HUMAINE database (Douglas-Cowie et al., 2007). The relevant concepts were identified, and made available as a set of configuration files for the video annotation tool Anvil (Kipp, 2001). A formal representation format was not proposed in this work. However, members of the team working on the HUMAINE database were active in the first Emotion Incubator group and made sure that the concepts identified as relevant were present in the discussion on use cases and requirements.

Markup languages including emotion-related information were defined mainly in the context of research systems generating emotion-related behaviour of ECAs. The expressive richness is usually limited to a small set of emotion categories, possibly an intensity dimension, and in some cases a three-dimensional continuous representation of activation-evaluation-power space (cf. Schröder et al., 2011c).

For example, the Virtual Human Markup Language VHML (Gustavsson et al., 2001) was created in order to control the behaviour of animated characters (virtual agents); in addition to markup for facial animation, speech synthesis, dialogue management etc., the specification also contains a section for representing emotions. The actual representations are very simple: a set of nine emotions is encoded directly as XML elements, e.g.:

```
<afraid intensity=”40”>
  Do I have to go to the dentist?
</afraid>
```

The Affective Presentation Markup Language APML (de Carolis et al., 2004) provides an attribute “affect” to encode an emotion category for an utterance (a “performative”) or for a part of it:

```
<performative affect=”afraid”>
  Do I have to go to the dentist?
</performative>
```

The Rich Representation Language RRL (van Deemter et al., 2008) uses an element “emotion”, embedded in a dialogue act, to represent the emotion. The emotion category and its intensity can be expressed, as well as the three emotion dimensions “activation”, “evaluation” and “power”. In addition, there is a conceptual distinction between feeling and expressing an emotion:

```
<dialogueAct>
  ...
  <emotion>
    <emotionExpressed type="afraid" intensity="0.3"
                activation="0.3" evaluation="-0.6" power="-0.3"/>
  </emotion>
 <sentence><text>Do I have to go to the dentist?</text>...</sentence>
</dialogueAct>
```

All these languages include the representation of an emotional state as one aspect in a complex representation oriented towards the generation of behaviour for an embodied conversational agent (ECA). None of the representations aim for reusability in different contexts, and none reach a representational power coming anywhere near the complexity considered to be necessary in emotion research (see e.g. Cowie et al., 2010).

An interesting contribution to the domain of computerised processing and representation of emotion-related concepts is A Layered Model of Affect, ALMA, provided by Gebhard (2005). The model encompasses the concepts of emotion (short-term affect), mood (medium-term affect), and personality (long-term affect). Following the OCC model (Ortony et al., 1988), ALMA uses *appraisal* mechanisms to trigger emotions from events, objects and actions in the world. Emotions have an intensity varying over time. Each individual emotion influences mood as a longer-term affective state. ALMA uses an XML-based markup language named AffectML in two places: to represent the antecedents to emotion, i.e. the appraisals leading to emotions, and to represent the impact that emotions and moods have on a virtual agent's behaviour.

The following snippet of markup shows how AffectML is used to describe a given character's affective predispositions, i.e. its propensity to react emotionally to different kinds of events (from Gebhard (2005)):

```
<CharacterAffect name="Valerie" monitored="true" docu="Valerie ">
  <Personality open="0.4" con="0.8" extra="0.6" agree="0.3" neur="0.4"/>
  <Appraisal>
    <Basic>
      <GoodEvent desirability="0.7"/>
      ...
    </Basic>
```

```
    <SelfAct type="Calm">
      <GoodActSelf agency="self" praiseworthiness="0.5"/>
    </SelfAct>
    <DirectAct type="Attack" performer="Sven">
      <BadEvent desirability="-0.5"/>
      <BadActOther agency="other" praiseworthiness="-0.3"/>
    </DirectAct>
    <SelfEmotion emotion="ReproachDisplay">
      <BadEvent desirability="-0.3"/>
    </SelfEmotion>
    ...
```

The current affective state of a character, to be expressed in the character's behaviour, is represented in AffectML as follows (from Gebhard (2005)):

```
<AffectOutput>
  <CharacterAffect name="Sven">
    <DominantEmotion name="Disliking" value="0.46"/>
    <Mood moodword="Exuberant" intensity="slightly" p="0.35" a="0.39" d="0.34"/>
    <Personality open="0.3" con="-0.6" extra="0.7" agree="0.4" neu="-0.1"/>
  </CharacterAffect>
  ...
</AffectOutput>
```

The focus in ALMA has been on providing a working implementation of a particular model of affect, based on OCC appraisals and emotions (Ortony et al., 1988), mood represented using Mehrabian's Pleasure-Arousal-Dominance (PAD) space (Mehrabian, 1996), and personality described using the five-factor model (McCrae and John, 1992). Mappings are used to relate the different models to one another. The AffectML language used for representing the various aspects of the model's data in the system is not described in detail; its focus has not been on generic reuse or interoperability, but on encoding the concepts relevant to this specific model. In recent work (Kipp et al., 2010), the output of ALMA has been represented using EmotionML.

The Emotion Annotation and Representation Language EARL (Schröder et al., 2006, 2011c) was introduced as an attempt to address reusability and to provide a representation approaching what is considered scientifically necessary. It can represent emotions alternatively in terms of categories, dimensions or appraisals; the

intensity of the state can be indicated; several kinds of regulation are previewed, e.g. the simulation, suppression or amplified expression of an emotional state; complex emotions can be represented, as in situations of regulation or when more than one emotion is present. For example:

```
<emotion category="afraid" intensity="0.4" suppress="0.6"
                activation="0.3" evaluation="-0.6" power="-0.3">
  Do I have to go to the dentist?
</emotion>
```

The following sections show how the ideas embedded in EARL were broadened and made more generic and flexibly usable in the development of the EmotionML specification. The resulting syntax of EmotionML (see Section 9.5) has changed quite substantially from the original EARL ideas; nevertheless, the motivating ideas have largely stayed the same.

9.3 Use cases

The types of technology in which an Emotion Markup Language might be used are very diverse. The 39 individual use cases collected by the Emotion Incubator group (Schröder et al., 2007b) include such diverse topics as the annotation of the emotional connotation in words and sentences, in pictures, or in audio recordings; the description of the emotional state of participants in a multi-party conversation as it changes over time; emotion detection for social robots; the use of computer games to induce emotions in the player; the reasoning about the emotional consequences of events; and the generation of emotional expressivity in synthetic faces and voices. The group structured the individual use case descriptions into three main types of use case (Schröder et al., 2007a):

Use Case 1: manual annotation of emotions in data;

Use Case 2: automatic detection of emotions;

Use Case 3: generation of emotion-related system behaviour.

This way of structuring the use cases seemed appropriate because the requirements arising from all the exemplars of a given use case are relatively similar. For

example, the type of detail that humans tend to annotate (Use Case 1) is orders of magnitude more fine-grained than what machines can detect (Use Case 2). Both of these have a natural notion of *confidence*, i.e. of certainty that the annotation is correct; on the other hand, this notion makes little sense in the context of synthesising system behaviour (Use Case 3).

9.4 Requirements

The Emotion Incubator group extracted requirements from the different use cases in an iterative process. First, each of the three use cases produced a separate set of requirements. These sets were then combined and aligned. The alignment process yielded an interesting exercise of aligning vocabulary: for example, the expressive behaviour related to an emotion would be called "input" in Use Case 2 (emotion detection) but it would be considered to be "output" in Use Case 3 (synthesis). The following principles were agreed upon and used in order to align and consolidate the sets of requirements (Schröder et al., 2007b):

(1) The emotion language should not try to represent sensor data, facial expressions, environmental data etc., but define a way of interfacing with external representations of such data.

(2) The use of system-centric vocabulary such as "input" and "output" should be avoided. Instead, concept names should be chosen by following the phenomena observed, such as "experiencer", "trigger", or "observable behaviour".

The process of aligning requirements and concepts yielded consensus terms (e.g., "observable behaviour" instead of "input" and "output") and agreement to avoid ambiguous or context-specific terms such as "input" or "output". Other terms were easier to align: the term "confidence" from Use Case 1 (manual annotation) was considered to be identical in its intended meaning with the term "probability" from Use Case 2; the consensus term in this case was "confidence" since it was felt to be the more generally applicable term.

The process was also useful to establish the intended boundaries of the Emotion Markup Language, according to the principle (1) above. With the broad range of

targeted use cases, describing the respective domain concepts or modality-specific expressions was clearly unrealistic.

The emphasis in the Emotion Incubator group was on coverage, in the sense of including as broad a list of requirements as it seemed reasonable. The group's final report (Schröder et al., 2007b) included a list of 22 requirements structured into five sections: (1) information about the emotion properties, (2) meta-information about the individual emotion annotations, (3) links to the rest of the world, (4) information about a number of global metadata, and (5) ontologies.

After collecting this broad and encompassing list, the Emotion Markup Language Incubator group focused on extracting a manageable subset by means of a collaborative prioritisation process involving the research community at large. The 22 requirements were presented in a publicly accessible questionnaire advertised via the portal `emotion-research.net`. For each requirement, the respondents had the choice between the following answers (Burkhardt and Schröder, 2008):

1. must have: The specification must define the feature.
2. should have: The specification should define the feature, if possible.
3. nice to have: The specification may optionally define the feature.
4. future revision: The feature needs additional study before specification.
5. no need: I don't see the need for this feature in the specification
6. no opinion

Ten group members and four individuals from outside the group filled in the questionnaire (Burkhardt and Schröder, 2008). The responses were used to establish a first provisional distinction between *mandatory* and *optional* requirements: if at least 70% of the respondents classified a requirement as a "must have" or a "should have", the requirement was included in the *mandatory* list, otherwise in the *optional* list. A subsequent discussion in the group made individual and well-motivated modifications to this classification, which resulted in a final list of fourteen mandatory and eight optional requirements.

The prioritised list of requirements (Burkhardt and Schröder, 2008; Schröder et al., 2008b) is presented in the following subsections.

9.4.1 Emotion Core

The description of the emotion or related state itself naturally receives the most prominent place in an emotion markup language. Most of the items on the original requirements list (Schröder et al., 2007b) are considered mandatory, allowing the user to represent most of the multiple facets of an emotion (Cowie and Cornelius, 2003), but also less intense affective states (Scherer, 2000).

Type of emotion-related phenomenon

The emotion markup language must be suitable for representing different types of emotion-related states – an *emotion* in the strong sense, i.e. a momentary, intense episode triggered by a concrete event, or rather a mood, an attitude, an interpersonal stance, etc.

Which taxonomy to use for distinguishing types of emotion-related phenomena is an open research question, and as for other elements of the emotion markup language, any standard will only be able to propose a "default" answer. Users must be able to replace the suggested taxonomy with one that fits their own needs. One possible starting point for proposing a default set, from the literature on emotion theory, is provided by Scherer (2000).

Emotion categories, dimensions, appraisals, and action tendencies

The emotion markup language must provide representations of emotions in terms of the most important descriptions from the scientific literature, namely categories, dimensions, appraisals, and action tendencies. A brief summary of the facets of emotion that are distinguished in the scientific literature is given below (Section 9.6).

Emotion categories can be chosen from a set of discrete labels. Dimensions, appraisals and action tendencies seem to be best represented as "scale" values. Depending on the use case, scales may be either continuous-valued or discrete.

Each of the four types of representation needs a vocabulary of names for the categories, dimensions etc.; again, the aim will be to propose a meaningful "default", and allow users to use a different set if they have specific needs.

Multiple and/or complex emotions

The markup must provide a mechanism to represent multiple emotions that are co-occurring in the same experiencer. Such co-occurrence may be due to the fact that co-occurring emotions have different triggers (e.g. when a person is angry about one thing and sad about another); it may be the case that different emotions are expressed through different modalities (such as when the face shows one emotion and the voice another).

A more difficult case of multiple or complex emotions is the phenomenon of emotion regulation which due to its complexity has been demoted to an optional requirement (see below).

Emotion intensity

The emotion markup must provide an emotion attribute to represent the intensity of an emotion. The intensity of an emotion is a unipolar scale.

Emotion timing

The emotion markup must provide a generic mechanism for temporal scope. The temporal scope of an emotion markup may be defined through a combination of start and end times, or by linking to items located on the time line such as utterances or gestures.

The *time course* of an emotion markup may be defined through a sampling mechanism, providing values at fixed intervals.

Optional requirements

Emotion regulation. Regulation covers a range of manipulations of an emotion or its expression by the experiencer. In a very basic interpretation, this includes a difference between the internal and the externalised state, i.e. cases of simulation and suppression. However, considerably more complex models of emotion regulation are described in the literature. For example, the display of an emotional state may be impeded due to some socio-cultural rules (Ekman, 2003): the expression of one emotion may be masked by another one, it may be inhibited, minimised or even exaggerated. Alternatively, it is possible to regulate, to some extent, the

emotion itself rather than its expression, through a process of re-appraisal (Gross, 2001). Representing regulation in a scientifically appropriate way appears to be a non-trivial challenge; for this reason, we avoided making regulation a mandatory requirement for EmotionML, despite its importance for modelling certain types of complex emotions.

9.4.2 Meta-information about emotion annotation

Confidence / probability

The emotion markup must provide a representation of the degree of confidence or probability that a certain element of the representation is correct. This is required by both machine classifiers and human annotators.

Modality

The emotion markup must be able to represent the modalities in which the emotion is reflected. Emotion may be expressed specifically in a certain modality, e.g. face, voice, body posture or hand gestures, but also lighting, font shape, etc.

Optional requirements

Acting. The emotion markup should provide a mechanism to add special attributes for acted emotions such as perceived naturalness, authenticity, quality, etc.

9.4.3 Links to the "rest of the world"

Emotion markup is always about something; therefore, providing suitable links to external entities is essential for the interpretation of the emotion markup.

Links to media, and the position on a time line in externally linked objects

The emotion markup must be able to refer to external media of various kinds. A generic linking mechanism is envisaged, which may point to a media object, such as a picture, an audio or video file, or a node in an XML document. This may be complemented with timing information, such as a start time and a duration.

The semantics of links to the "rest of the world"

The emotion markup must provide a mechanism for assigning meaning to those links. Initially, the following meanings are envisaged: the experiencer (who "has" the emotion); the observable behaviour "expressing" the emotion; the trigger, cause or eliciting event of the emotion; and the object or target of the emotion (i.e., the thing that the emotion is about).

9.4.4 Global metadata

In order to facilitate communication between a producer and a consumer of emotional data with respect to application-specific information, the emotional markup may need to contain global metadata.

A generic mechanism to represent global metadata

The emotion markup must provide a generic mechanism for representing metadata on a global (per document) and on a local (per annotation) level. This is needed in order to facilitate communication between a producer and a consumer of emotional data with respect to application specific information.

Optional requirements

A number of specific types of metadata are listed as optional requirements. These include information about persons, including descriptions of the person or algorithm that produced the annotation; a description of the social and communicative environment, including the situational context; the purpose of classification (in Use Case 2); and the technical environment, such as frame rates, sensor descriptions, etc.

9.4.5 Ontologies of emotion

Ontologies are structured representations that provide a definition of terms and their relation to one another. In principle, therefore, it would seem appropriate to define emotion concepts in an ontology.

However, given the practical constraints that would have arisen from the decision to use ontologies, the group has decided to declare these requirements as optional. Similarly to the case of *regulation*, these are optional for practical rather than principled reasons: their relevance is undisputed, but including them in a first version of an Emotion Markup Language would have borne the risk of severely slowing down progress.

Optional requirements

Mappings between different emotion representations. It should be possible to map between different emotion representations, i.e. to convert data from one emotion description (categories, dimensions, appraisals, or action tendencies) to another. These different emotion representations are not independent; rather, they describe different aspects of the complex phenomenon emotion. Insofar, it is conceptually possible to map from one representation to another one in some cases; in other cases, mappings are not fully possible. Some use cases require mapping between different emotion representations: e.g., from categories to dimensions, from dimensions to coarse categories (a lossy mapping), from appraisals onto dimensions, from categories to appraisals, etc. Such mappings may either be based on findings from emotion theory or they can be defined in an application-specific way. The requirement concerns first of all the mapping mechanism as such, and in a number of reasonable cases, the mappings themselves.

Relationships between concepts in an emotion description. The concepts in an emotion description are usually not independent, but are related to one another. For example, emotion words may form a hierarchy, as suggested e.g. by prototype theories of emotions. For example, Shaver et al. (1987) classified cheerfulness, zest, contentment, pride, optimism, enthrallment and relief as different kinds of joy, whereas irritation, exasperation, rage, disgust, envy and torment represent different kinds of anger, etc. Such structures, be they motivated by emotion theory or by application-specific requirements, may be an important complement to the representations in an Emotion Markup Language. In particular, they would allow for a mapping from a larger set of categories to a smaller set of higher-level categories.

9.5 Syntax

Based on the requirements listed above, a syntax for EmotionML has been produced in a sequence of steps, first in the Emotion Markup Language Incubator group (Schröder et al., 2008a), then as formal working drafts (Schröder et al., 2009a, 2010a, 2011a). The following snippet exemplifies the principles of the EmotionML syntax.

```
<sentence id="sent1">
  Do I have to go to the dentist?
</sentence>
<emotion xmlns="http://www.w3.org/2009/10/emotionml"
    category-set="http://www.w3.org/TR/emotion-voc/xml#everyday-categories">
  <category name="afraid" value="0.4"/>
  <reference role="expressedBy" uri="#sent1"/>
</emotion>
```

The following properties can be observed.

- The emotion annotation is self-contained within an '<emotion>' element;
- all emotion elements belong to a specific namespace;
- it is explicit in the example that emotion is represented in terms of categories;
- it is explicit from which category set the category label is chosen;
- the link to the annotated material is realised via a reference using a URI, and the reference has an explicit role.

In the following subsections, we will discuss the properties of the EmotionML syntax in some more detail.

9.5.1 Design principles: self-contained emotion annotation

EmotionML is conceived as a plug-in language, with the aim to be usable in many different contexts. Therefore, proper encapsulation is essential. All information concerning an individual emotion annotation is contained within a single '<emotion>' element. All emotion markup belongs to a unique XML namespace.

EmotionML differs from many other markup languages in the sense that it does not *enclose* the annotated material. In order to link the emotion markup with the annotated material, either the reference mechanism in EmotionML or another mechanism external to EmotionML can be used.

Structurally, EmotionML uses element and attribute names to indicate the type of information being represented; attribute values provide the actual information. The use of attribute values (e.g., '<category name="joy"/>') was preferred over enclosed text (e.g., '<category>joy</category>') so that adding EmotionML to an XML node does not change that node's text content.

A top-level element '<emotionml>' enables the creation of stand-alone EmotionML documents, essentially grouping a number of emotion annotations together, but also providing document-level mechanisms for annotating global metadata and for defining emotion vocabularies (see below). It is thus possible to use EmotionML both as a standalone markup and as a plug-in annotation in different contexts.

9.5.2 Representations of emotion

Emotions can be represented in terms of four types of descriptions taken from the scientific literature (see Section 9.6): '<category>', '<dimension>', '<appraisal>' and '<action-tendency>'. An '<emotion>' element can contain one ore more of these descriptors; each descriptor must have a 'name' attribute and can have a 'value' attribute indicating the intensity of the respective descriptor. For '<dimension>', the 'value' attribute is mandatory, since a dimensional emotion description is always a position on one or more scales; for the other descriptions, it is possible to omit the 'value' to only make a binary statement about the presence of a given category, appraisal or action tendency.

The following example illustrates a number of possible uses of the core emotion representations.

```
<category name="affectionate"/>
<category name="amused" value="0.7"/>
<dimension name="valence" value="0.9"/>
<appraisal name="agent-self"/>
<appraisal name="urgency" value="0.2"/>
```

```
<action-tendency name="approach"/>
<action-tendency name="dominating" value="0.8"/>
```

9.5.3 Mechanism for referring to an emotion vocabulary

Since there is no single agreed vocabulary for each of the four types of emotion descriptions (see Section 9.7), EmotionML provides a mandatory mechanism for identifying the vocabulary used in a given '<emotion>'. The mechanism consists in attributes of '<emotion>' named 'category-set', 'dimension-set' etc., indicating which vocabulary of descriptors for annotating categories, dimensions etc. are used in that emotion annotation. These attributes contain a URI pointing to an XML representation of a vocabulary definition (see Section 9.7). In order to verify that an emotion annotation is valid, an EmotionML processor must retrieve the vocabulary definition and check that every 'name' of a corresponding descriptor is part of that vocabulary (see also Section 9.8.1).

For example, the following annotation uses Mehrabian's PAD model (Mehrabian, 1996) for representing a position in three-dimensional space.

```
<emotion dimension-set="http://www.w3.org/TR/emotion-voc/xml#pad-dimensions">
    <dimension name="arousal" value="0.3"/>   <!-- lower-than-average arousal -->
    <dimension name="pleasure" value="0.9"/>  <!-- very high positive valence -->
    <dimension name="dominance" value="0.8"/> <!-- relatively high potency    -->
</emotion>
```

9.5.4 Meta-information

Several types of meta-information can be represented in EmotionML.

First, each emotion descriptor (such as '<category>') can have a 'confidence' attribute to indicate the expected reliability of this piece of the annotation. This can reflect the confidence of a human annotator or the probability computed by a machine classifier. If several descriptors are used jointly within an '<emotion>', each descriptor has its own 'confidence' attribute. For example, it is possible to have high confidence in, say, the arousal dimension but be uncertain about the pleasure dimension:

```
<emotion dimension-set="http://www.w3.org/TR/emotion-voc/xml#pad-dimensions">
    <dimension name="arousal" value="0.7" confidence="0.9"/>
```

```
    <dimension name="pleasure" value="0.6" confidence="0.3"/>
</emotion>
```

Each '<emotion>' can have an 'expressed-through' attribute providing a list of modalities through which the emotion is expressed. Given the open-ended application domains for EmotionML, it is naturally difficult to provide a complete list of relevant modalities. The solution provided in EmotionML is to propose a list of human-centric modalities, such as 'gaze', 'face', 'voice', etc., and to allow arbitrary additional values. The following example represents a case where an emotion is recognised from, or to be generated in, face and voice:

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#everyday-categories"
         expressed-through="face voice">
    <category name="satisfaction"/>
</emotion>
```

For arbitrary additional metadata, EmotionML provides an '<info>' element which can contain arbitrary XML structures. The '<info>' element can occur as a child of '<emotion>' to provide local metadata, i.e. additional information about the specific emotion annotation; it can also occur in standalone EmotionML documents as a child of the root node '<emotionml>' to provide global metadata, i.e. information that is constant for all emotion annotations in the document. This can include information about sensor settings, annotator identities, situational context etc. How to represent this information below '<info>' is up to the user.

9.5.5 References to the "rest of the world"

Emotion annotation is always about something. There is a subject "experiencing" (or simulating) the emotion. This can be a human, a virtual agent, a robot, etc. There is observable behaviour expressing the emotion, such as facial expressions, gestures, or vocal effects. With suitable measurement tools, this can also include physiological changes such as sweating or a change in heart rate or blood pressure. Emotions are often caused or triggered by an identifiable entity, such as a person, an object, an event, etc. More precisely, the appraisals leading to the emotion are triggered by that entity. And finally, emotions, or more precisely the emotion-related action tendencies, may be directed towards an entity, such as a person or an object.

EmotionML considers all of these external entities to be out of scope of the language itself; however, it provides a generic mechanism for referring to such entities. Each '<emotion>' can use one or more '<reference>' elements to point to arbitrary URIs. A '<reference>' has a 'role' attribute, which can have one of the following four values: 'expressedBy' (default), 'experiencedBy', 'triggeredBy', and 'targetedAt'. Using this mechanism, it is possible to point to arbitrary entities filling the above-mentioned four roles; all that is required is that these entities be identified by a URI.

9.5.6 Time

Time is relevant to EmotionML in the sense that it is necessary to represent the time during which an emotion annotation is applicable. In this sense, temporal specification complements the above-mentioned reference mechanism.

Representing time is an astonishingly complex issue. A number of different mechanisms are required to cover the range of possible use cases.

First, it may be necessary to link to a time span in media, such as video or audio recordings. For this purpose, the '<reference role="expressedBy">' mechanism can use a so-called Media Fragment URI (Troncy et al., 2010) to point to a time span within the media. In the following example, the emotion is expressed from seconds 3 to 7 in the video 'party.avi':

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#big6">
    <category name="happiness"/>
    <reference uri="party.avi#t=3,7"/>
</emotion>
```

Second, time may be represented on an absolute or relative scale. EmotionML follows EMMA (Johnston et al., 2009) in representing time in these cases. Absolute time is represented in milliseconds since 1 January 1970, using the attributes 'start' and 'end'. A combination of the 'start' and 'duration' attributes can also be used to represent time intervals. For example:

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#big6"
         start="1268647331" end="1268647831">
    <category name="joy"/>
</emotion>
```

or, equivalently,

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#big6"
         start="1268647331" duration="500">
    <category name="joy"/>
</emotion>
```

Absolute times are useful for applications such as affective diaries, which record emotions throughout the day, and whose purpose it is to link back emotions to the situations in which they were encountered.

Other applications require relative time, for example time since the start of a session. Here, the mechanism borrowed from EMMA is the combination of '`time-ref-uri`' and '`offset-to-start`'. The former provides a reference to the entity defining the meaning of time 0; the latter is time, in milliseconds, since that moment. In case the entity pointed to by '`time-ref-uri`' is itself a time span, it is possible to indicate using '`time-ref-anchor-point`' whether the start or the end of that time span is supposed to be the reference for the relative time. The following example represents an emotion observed from seconds 3 to 7 of the session identified by the URI '`#my_session_id`':

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#big6"
         time-ref-uri="#my_session_id" offset-to-start="2000" duration="5000">
    <category name="surprise"/>
</emotion>
```

9.5.7 Representing continuous values and dynamic changes

A mentioned above, the emotion descriptors '`<category>`', '`<dimension>`', '`<appraisal>`' and '`<action-tendency>`' can have a '`value`' attribute to indicate the position on a scale corresponding to the respective descriptor. In the case of a dimension, the value indicates the position on that dimension, which is mandatory information for dimensions; in the case of categories, appraisals and action tendencies, the value can be optionally used to indicate the extent to which the respective item is present.

In all cases, the '`value`' attribute contains a floating-point number between 0 and 1. The two end points of that scale represent the most extreme possible values, for example the lowest and highest possible positions on a dimension, or

the complete absence of an emotion category vs. the most intense possible state of that category.

The 'value' attribute thus provides a fine-grained control of the position on a scale, which is constant throughout the temporal scope of the individual '<emotion>' annotation. It is also possible to represent changes over time of these scale values, using the '<trace>' element which can be a child of any '<category>', '<dimension>', '<appraisal>' or '<action-tendency>' element. The following example illustrates the use of a trace to represent an episode of fear during which intensity is rising, first gradually, then quickly to a very high value. Values are taken at a sampling frequency of 10 Hz, i.e. one value every 100 ms.

```
<emotion category-set="http://www.w3.org/TR/emotion-voc/xml#big6">
  <category name="fear">
    <trace freq="10Hz" samples="0.1 0.1 0.15 0.2 0.2 0.25 0.25
                                0.25 0.3 0.3 0.35 0.5 0.7 0.8 0.85 0.85"/>
  </category>
</emotion>
```

9.6 Scientific descriptions of emotion

In the scientific literature on emotion research, there is no single agreed description of emotions. In part this is due to different research traditions – for example, Cornelius (2000) distinguishes four traditions with deep conceptual differences. The *Darwinian* tradition views emotions as evolutionarily shaped responses that benefit the survival of the individual, such as attack or flight responses. The *Jamesian* tradition, which goes back to William James' work in the late 19th century, considers emotions to be the subjective feeling experience of bodily response patterns. The *cognitive* perspective emphasises the process of *appraisal* which generates emotions depending on the relevance of events for the individual. The *social constructivist* perspective, finally, considers emotions to fulfil the role of regulating the social structure, of shaping the interaction among individuals by enforcing social norms.

To make matters even more challenging, Cowie (2010) suggests that these traditions focus on the so-called *emergent* emotions – short-lived, intense response patterns triggered by clearly identifiable events, whereas the more relevant con-

cept for technology, according to Cowie, is the notion of "emotional colouring" or "pervasive" emotion. Whereas the states of "emergent" emotions are rare, most of human experience appears to show some emotional colouring. Relevant "emotion-related" states, according to Cowie, include 'mood', 'stance towards object/situation', 'altered states of arousal', 'interpersonal bonds', 'altered states of control', 'emergent emotion', and 'interpersonal states'. These terms are listed in decreasing order of frequency according to a study in which each of ten participants was called 50 times over the phone to indicate their current type of state, at random times over the course of several weeks. This means that according to this study, 'mood', 'stance towards object/situation' and 'altered states of arousal' are the most frequent types of emotion-related condition, and the 'emergent emotion' that is studied by traditional emotion theories is a rather rare phenomenon.

Insofar, it appears important for an Emotion Markup Language to be able to represent the most relevant aspects of emotions in the broader sense, including the emotion-related conditions. Given the lack of agreement in the literature on the most relevant aspects of emotion, it is inevitable to provide a relatively rich set of descriptive devices.

Despite the diversity of approaches, however, there seems to be reasonable agreement in the scientific literature on a number of 'components' or 'facets' that play an important role in relation to emergent emotion and, to some extent, also for the other emotion-related states. Scherer (2005) distinguishes the following components:

- cognitive component (appraisal);
- neurophysiological component (bodily symptoms);
- motivational component (action tendencies);
- motor expression component (facial and vocal expression);
- subjective feeling component (emotional experience).

According to this view, which appears to be established consensus in the research community, an emotional episode includes the following relevant aspects. An emotion-eliciting event is *appraised* as somehow relevant for the individual,

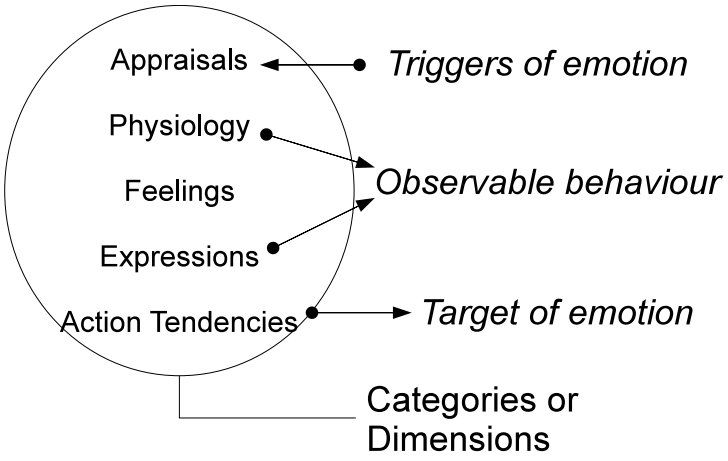

Figure 9.1: Components of emotion and how they are linked to external representations (from Schröder et al. (2007b))

which 'causes' or 'triggers' the emotion. The experiencer has a *subjective experience* of the emotion; this may be accompanied by *bodily symptoms* and *expressive behaviour* in a number of modalities. Finally, the emotion may also induce in the experiencer a *tendency to act* in a particular way; such action tendency may be directed towards an object or target of some sort.

Figure 9.1 illustrates these concepts and how they are taken into account in the EmotionML specification. Descriptions in terms of categories and dimensions provide a global account of all five components of the emotion. Appraisals are represented explicitly, and are complemented with references to the 'triggers' of emotion that are being appraised. Similarly, action tendencies are represented explicitly, and it is possible to refer to the 'targets' towards which they are directed. Physiology and expressions are not represented in EmotionML itself, but using a mechanism to refer to 'observable behaviour'. Feelings are covered by the more global representations of categories and dimensions.

In conclusion, it can be said that the mechanisms in EmotionML are able to capture the main elements of what scientific theory considers important of emotion.

9.7 Vocabularies for EmotionML

Section 9.6 has shown the key concepts from scientific emotion research that are taken into account in EmotionML. Four types of descriptions are available: categories, dimensions, appraisals, and action tendencies. Depending on the tradition of emotion research and on the use case, it may be appropriate to use any single one of these representations; alternatively, it may also make sense to use *combinations* of descriptions to characterise more fully the various aspects of an emotional state that are observed: how an appraisal of triggers caused the emotion; how it can be characterised using a global description in terms of a category and/or a set of dimensions; and the potential actions the individual may be executing as a result. Insofar, EmotionML is a powerful representational device.

This description glosses over one important detail, however. Whereas emotion researchers may agree to some extent on the types of facets that play a role in the emotion process (such as appraisals, feeling, expression, etc.), there is no general consensus on the descriptive vocabularies that should be used. Which set of emotion *categories* is considered appropriate varies dramatically between the different traditions; and even within a tradition such as the Darwinian tradition of emotion research, there is no agreed set of emotion categories that should be considered as the most important ones (see e.g. Cowie and Cornelius, 2003). Similarly, dimensional accounts of emotion do not agree on either the number or the names that should be given to the different dimensions.

For this reason, any attempt to enforce a closed set of descriptors for emotions would invariably draw heavy criticism from a range of research fields. Given that there is no consensus in the community, it is impossible to produce a consensus annotation in a standard markup language.

An obvious alternative would have been to define in EmotionML only the structure of the representation, but not the descriptors used. Every user would be free to use the descriptors they consider most appropriate given their theoretical stance and application needs. This approach, however, would have dramatically limited the suitability of EmotionML to serve as an interchange format enabling the interoperability of components processing emotion, for two reasons. On the one hand, a fully open set would make it impossible to predict and interpret the multitude of terms that may be defined. On the other hand, the same term may be

used with markedly different meaning in different research traditions. For example, the term 'anger' would be considered to represent an attack-type reaction in the Darwinian tradition; it would be seen as the result of an appraisal of an event as goal-obstructive in appraisal theories; and it would be seen as a societal device for ensuring conformity with the social norms in the social constructivist tradition. Another piece of evidence showing the inherent ambiguity of a term such as 'anger' comes from the area of speech emotion research. For example, Banse and Scherer (1996) distinguish between 'hot' and 'cold' anger, which show markedly different vocal profiles but also different action tendencies.

It is thus neither possible to standardise a closed set of emotion terms, nor is it desirable to leave the choice of labels completely undefined and up to the user. For this reason, the solution pursued in EmotionML is of a third kind. The notion of an 'emotion vocabulary' is introduced: any specific emotion annotation must be specific about the vocabulary that is being used in that annotation. This makes it possible to define in a clear way the terms that make sense in a given research tradition. Computer systems that want to interoperate need to settle on the emotion vocabularies to use; whether a given piece of EmotionML markup can be meaningfully interpreted by an EmotionML engine can be determined.

The specification includes a mechanism for defining emotion vocabularies. It consists of a '<vocabulary>' element containing a number of '<item>' elements. A vocabulary has a 'type' attribute, indicating whether it is a vocabulary for representing categories, dimensions, appraisals or action tendencies. A vocabulary item has a 'name' attribute. Both the entire vocabulary and each individual item can have an '<info>' child to provide arbitrary metadata.

A separate W3C Working Draft (Schröder et al., 2011b) complements the specification to provide EmotionML with a set of emotion vocabularies taken from the scientific literature. When the user considers them suitable, these vocabularies rather than arbitrary other vocabularies should be used in order to promote interoperability. Whenever users have a need for a different vocabulary, however, they can simply define their own custom vocabulary and use it in the same way as the vocabularies listed in the Vocabularies document. This makes it possible to add any vocabularies from scientific research that are missing from the pre-defined set, as well as application-specific vocabularies. This approach promotes inter-

operability where this is considered meaningful by the users, but leaves users the freedom to use the most suitable representations for their application.

In selecting emotion vocabularies, the group has applied the following criteria. The primary guiding principle has been to select vocabularies that are either commonly used in technological contexts, or represent current emotion models from the scientific literature. A further criterion is related to the difficulty to define mappings between categories, dimensions, appraisals and action tendencies. For this reason, groups of vocabularies were included for which some of these mappings are likely to be definable in the future.

The following vocabularies are defined. For categorical descriptions, the "big six" basic emotion vocabulary by Ekman (1972), an everyday emotion vocabulary by Cowie et al. (1999), and three sets of categories that lend themselves to mappings to appraisals, dimensions and action tendencies: the OCC categories from Ortony et al. (1988), the categories used by Fontaine et al. (2007), and the categories from the work by Frijda (1986). Three dimensional vocabularies are provided, the pleasure-arousal-dominance (PAD) vocabulary by Mehrabian (1996), the four-dimensional vocabulary proposed by Fontaine et al. (2007), and a vocabulary providing a single '`intensity`' dimension for such use cases that want to represent solely the intensity of an emotion without any statement regarding the nature of that emotion. For appraisal, three vocabularies are proposed: the OCC appraisals from Ortony et al. (1988), the Stimulus Evaluation Checks by Scherer (1984, 1999), and the EMA appraisals by Gratch and Marsella (2004). Finally, for action tendencies, only a single vocabulary is currently listed, namely that proposed by Frijda (1986).

While these vocabularies should provide users with a solid basis, it is likely that additional vocabularies or clarifications about the current vocabularies will be requested. Due to the rather informal nature of a non-Recommendation-track Working Draft, it is rather easy to provide future versions of the document that provide the additional information required.

9.8 Validating EmotionML

One important question for a standard representation format is the issue of *validation*, i.e. of verifying in an automatic way whether a given document adheres

to the specification. A well-established approach is to use XML Schema (Thompson et al., 2004) for validation. A Schema document defines the correct form of a document following the specification.

The specific challenge in the case of EmotionML lies in the fact that users can select the vocabularies they wish to use in the emotion annotation, including custom vocabularies which by principle cannot be known at the time of writing the specification. How, then, can it be automatically verified that the EmotionML document contains only names for categories, dimensions etc. that are defined in the vocabulary that the markup declares to be used?

9.8.1 Schema and processor validation in EmotionML

After extensive discussion, the EmotionML specification does not perform Schema validation of vocabulary items. Instead, the Schema defines and verifies only the structure of the EmotionML document. The consistency between the reference to an emotion vocabulary and the vocabulary items used in the annotation needs to be verified by the emotion engine.

The reason for this decision is not that it would not have been possible to perform Schema validation of custom vocabularies. In fact, a solution based solely on XML Schema, without requiring more complex validation machinery, was developed and shown to work in principle (see Section 9.8.2 below).

The reason is rather one of consistency with established practice. Where possible, XML specifications in the W3C should attempt to define a single XML namespace (Bray et al., 2009) for all their markup. This is an important issue specifically in view of keeping open the door for potential future interoperability with the Hypertext Markup Language HTML (Raggett et al., 1999), in which the concept of namespaces is not supported. In line with this preference, the EmotionML specification as described above uses a single namespace for its markup.

A second aspect of established practice is that in W3C specifications, user-defined strings should occur only in attribute values. Element names and attribute names, on the other hand, should be fully defined by the specification. The reason behind this custom is to make it easy to recognise a given markup format as such. User-defined element names would change the appearance of a language to an

extent that might make it difficult to recognise the markup used. As a result, all vocabulary items in the EmotionML specification are contained in attribute values.

9.8.2 An alternative solution based on XML namespaces

In the process of discussing the options, we had developed a solution for supporting custom vocabularies in validation based purely on XML. Even though the solution was discarded in the context of EmotionML for the reasons of consistency outlined above, it is presented here as an example of technical feasibility which may be useful in a different context at some point in the future.

The solution described here uses *qualified names* (QNames, see Bray et al. (2009)), to distinguish the different vocabulary sets by namespace. The QName of an element is that element's local name, optionally prefixed with a *namespace prefix* identifying the element's namespace.

The central idea is that the plug-in vocabulary is in a different namespace defined by its own Schema file, but at the same time the vocabulary items are part of a substitution group defined in the main emotion markup Schema. The following minimalistic example illustrates the mechanism for emotion dimensions.

Figure 9.2 shows a sample document which can be Schema-validated including the names and syntax of the vocabulary items. Figure 9.3 provides a minimalistic central Schema for the emotion markup, which defines only a simplistic dimension annotation. Figure 9.4, finally, is an exemplar of the set of vocabulary-defining Schemas, in this case for the dimension vocabulary consisting of the identifiers 'arousal' and 'valence'.

When looking at Figure 9.2, the points to notice are the following.

- EmotionML structural elements (`<emotionml>`, `<emotion>`, `<dimensions>`) are in a different namespace than the specific vocabulary items (`<arousal>`, `<valence>`);
- for the two namespaces, two schema locations are indicated in the document.

The combination of the central Schema `emo.xsd` (Figure 9.3) and the vocabulary-specific Schema `arousalvalence.xsd` (Figure 9.4) works as follows. The structure of the markup is defined in `emo.xsd` including an element

```
<!-- emotion-document.xml -->
<e:emotionml xmlns:e="http://www.example.com/emotionml"
    xmlns:av="http://www.example.com/arousalValence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/emotionml file:emo.xsd
  http://www.example.com/arousalValence file:arousalvalence.xsd">
  <e:emotion>
    <e:dimensions>
      <av:arousal value="0.5"/>
      <av:valence value="0"/>
    </e:dimensions>
  </e:emotion>
</e:emotionml>
```

Figure 9.2: Sample document of a variety of emotion markup for which the correct use of vocabulary items can be Schema-validated.

<dimension> with type dimensionType. That is, emo.xsd defines the places where the dimension element can occur (in this case, within the <dimensions> element wrapping one or more dimension elements), and what internal structure it can have (in this case, a required, string-valued attribute value). The element <dimension> itself, however, is abstract, which means it cannot actually occur in the markup.

The vocabulary-specific Schema arousalvalence.xsd (Figure 9.4) reuses these definitions as follows.

- It imports the generic markup namespace, which allows it to use the definitions in the generic Schema emo.xsd;
- it declares each of the dimension elements in the vocabulary as part of the substitutionGroup of the generic <dimension> element, which allows the new dimensions to occur at the same places in the markup structure as the generic element;
- it defines the type of the new dimension to be the same dimensionType used for all dimensions.

As a result, the new dimension elements in the vocabulary can occur exactly where dimension elements are allowed, and they must have the exact internal structure defined for dimension elements in the generic Schema.

```
<!-- emo.xsd -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:emo="http://www.example.com/emotionml"
            targetNamespace="http://www.example.com/emotionml">

<xsd:complexType name="dimensionType">
  <xsd:attribute name="value" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:element name="dimension" type="emo:dimensionType" abstract="true"/>

<xsd:element name="dimensions">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="emo:dimension" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="emotion">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="emo:dimensions" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="emotionml">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="emo:emotion" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Figure 9.3: Minimalistic generic schema `emo.xsd` for a variety of emotion markup for which the correct use of vocabulary items can be Schema-validated.

```
<!-- arousalvalence.xsd -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:emo="http://www.example.com/emotionml"
            targetNamespace="http://www.example.com/arousalValence">

<xsd:import namespace="http://www.example.com/emotionml" />

<xsd:element name="arousal" substitutionGroup="emo:dimension"
             type="emo:dimensionType"/>
<xsd:element name="valence" substitutionGroup="emo:dimension"
             type="emo:dimensionType"/>

</xsd:schema>
```

Figure 9.4: Schema `arousalvalence.xsd` for one of the emotion vocabularies for a variety of emotion markup for which the correct use of vocabulary items can be Schema-validated.

When placing all three documents into the same folder, this can be Schema-validated for example using Xerces-Java:

```
export CLASSPATH=xercesSamples.jar:$CLASSPATH
java dom.Counter -v -s -f emotion-document.xml
```

Defining a new vocabulary in this framework consists in creating a new vocabulary Schema as a variant of Figure 9.4. That new vocabulary could then be used in documents in the same way as the namespace `http://www.example.com/arousalValence` and the Schema `arousalvalence.xsd` have been used in Figure 9.2. In particular, no change to the generic Schema (Figure 9.3) is required to use a new vocabulary.

The mechanism generalises immediately to appraisals and action tendencies, which have the same structure as dimensions. For emotion categories, the mechanism can be applied if a category is represented by an element whose element name is the category name. For example, a categorical annotation using the proposed solution would look as follows:

```
<e:emotion>
  <e:category>
    <big6:happiness/>
  </e:category>
</e:emotion>
```

Here the generic Schema would need to provide a similar mechanism for categories as has been made explicit for dimensions; the namespace prefix 'big6' refers to a namespace tied to a Schema document defining the 'big 6' vocabulary of emotion categories.

9.9 Issues for future work

The EmotionML 1.0 specification appears to be successful at resolving the majority of the requirements that arise from use cases. This was confirmed from the side of users as well as by psychological experts in emotion research, at the W3C EmotionML workshop (http://www.w3.org/2010/10/emotionml/cfp.html). The process of standardisation is therefore being pursued as described in Section 9.1.

A number of important issues have been noted as important but too difficult to handle in the first version of EmotionML. Among these is a careful solution for representing *regulation* in EmotionML (see Section 9.4.1), in order to be able to represent the fact that an emotion was suppressed, simulated, masked by another emotion, etc. Another requirement that is not covered in EmotionML 1.0 is the use of ontologies to define the terms in an emotion vocabulary, to relate the terms to one another, and to define mappings between emotion vocabularies where possible. A further difficulty regards the specification of scales. Should a given scale be discrete, continuous, unipolar or bipolar, etc.? Due to the difficulty of finding a consensus in the emotion community on best practice for scales, we have postponed a more detailed definition of scales.

With the increasingly broad discussion of EmotionML in different communities, further use cases are becoming apparent. In the context of accessibility for people with disabilities, EmotionML may be useful for making emotional information available to people who could not otherwise perceive it. Examples include emotional annotation of material for persons who do not have a natural intuition about the emotional meaning of other person's expressive behaviour, such as people with autism; the equivalent of subtitles for emotions that are expressed only in the voice, for people with impaired hearing; and the equivalent of descriptive text to capture emotion that is only expressed in visual modalities, for people who cannot see. In each case, the annotation of emotion in EmotionML would have to be rendered into an easy-to-understand surface form for actual consumption.

Another interesting potential application area for EmotionML is user modeling (Rich, 1979). Generic user modeling systems attempt to collect general as well as domain-specific information about a human user in order to enable a computer system to adapt to the user's needs (Kobsa, 2001). One generic but highly relevant aspect of the user's properties is his or her emotional state. For example, Heckmann (2006) presented a General User Model Ontology (GUMO) which includes the user's emotional state as one of the basic user dimensions. In GUMO, emotion categories are included as individual instances of a generic class "emotional state".

EmotionML is of potential relevance for user modeling on several levels. Irrespective of the question whether the XML representation of EmotionML is appropriate for a given user model system, the concepts embedded in EmotionML can be used as a guideline for representing the most relevant aspects of an emotional experience in a well-defined way. For example, rather than referring to an unqualified and potentially ambiguous emotion category by a simple label such as "happiness", it is possible to indicate precisely the emotion vocabulary which defines the meaning of that emotion label and the research tradition from which it stems. The reference mechanism defined in EmotionML makes it explicit that it can be relevant to know who experiences the emotion, how it is expressed, which object or event has triggered it, and towards which entity any actions resulting from the emotion may be targeted. In addition to the emotion itself, it may be useful to make explicit the appraisals that are formed on the basis of events in the world. EmotionML provides a formalism and several possible vocabularies for representing the appraisals as such. Affective reasoning components such as those by Gebhard (2005) and Gratch and Marsella (2004) can be used to derive the user's presumed emotion from those appraisals. Furthermore, the action tendencies that may arise from the emotion could be predicted using a similar reasoning component. Insofar, the user model could implement to some extent the emotional aspect of the computer's "Theory of Mind" (Baron-Cohen et al., 1999) of the user.

When extending user modeling towards the modeling of social relationships (Eagle and Pentland, 2006), EmotionML could potentially be used to represent one person's perception of another person's emotional expression. Here it becomes very important to distinguish the encoding from the decoding aspect of emotional expression (Cowie and Cornelius, 2003; Scherer, 2000): the user model should capture person A's presumed perception of person B's expressive behaviour,

which may well differ from person B's actual intentions[1]. If a user model includes a representation of emotion perception in this way, affective reasoning models could be extended to include appraisals of that behaviour in context. Furthermore, emotional contagion models (Hatfield et al., 1994) could be implemented to capture, for example, mimickry and imitation effects.

As these use cases are explored further, it remains to be seen whether the existing representations in EmotionML are appropriate and sufficient, or if additional functionality is required to address all relevant aspects of these additional areas of application.

A challenge on a different level is the investigation and possible definition of emotion-related representations for use in web applications. Important issues in this context are the minimal representation of small, pertinent pieces of information to be represented in a way that goes naturally with HTML, such as stylesheet-type annotations. Of high importance in this context are also the events that would need to be fired in relation to emotional material, such as 'emotion detected', 'emotion changed', 'emotion ended' or similar.

Once EmotionML 1.0 has reached its full maturity, the above-mentioned directions can be developed in future versions of EmotionML, or in complementary specifications that are more appropriate for the respective use cases.

9.10 Conclusion

The present chapter has presented an account of the thinking behind and the definition of the Emotion Markup Language EmotionML.

We have first set the scene by outlining the process of standardising EmotionML at the W3C and by presenting related work. We have then reported on the use cases and the resulting requirements as identified by two Incubator groups at the W3C, before describing the key properties of the EmotionML syntax as defined in the Last Call Working Draft version of the specification. We have compared the specification to scientific descriptions of emotion, concluding that the key concepts can be represented in EmotionML. Referring to the lack of agreement in the community regarding concrete vocabularies of emotion descriptors, we have mo-

[1] In particular, culture-specific display rules (Ekman, 1972) are likely to cause divergence between encoding and decoding in cross-cultural interactions.

tivated and described the mechanism in EmotionML to choose a suitable emotion vocabulary. Finally, we have discussed the issue of validating EmotionML and pointed out issues for future work.

This concludes the Communication part of the thesis. In the following chapter, we will look at an assessment of the SEMAINE API, firstly from the perspective of performance.

Part IV

Assessment

Chapter 10

Performance

A crucial aspect for a component integration framework is its performance. The low-level performance of the underlying communication middleware, in terms of message routing times, provides a lower bound to the possible reaction times a system can achieve if its processing takes no time. We will investigate this aspect in Section 10.1.

For a distributed component integration framework, a number of additional factors are crucial determinants of reaction times. We will investigate the effects of system architecture, of distribution across more than one computer, and of optimising individual components on the system's reaction times, in Section 10.2.

10.1 Middleware

A basic but essential aspect of performance in a component integration framework is the time it takes for a message to get from one component to another. Especially in a highly modular system consisting of many components, the number of messages passed from one component to another one can be quite high; any time spent on message passing is a lower bound on reaction times which cannot be improved even with components that can process in zero time. For example, if ten messages passed between the component that decides to execute an action and the component that actually renders the physical action, then ten times the message routing time is a lower bound on reaction times. If it took a message 50 ms to be sent and received, then the lower bound on reaction time would be 500 ms in this case –

String length	Psyclone	ActiveMQ
10	16.00	0.34
100	16.78	0.25
1,000	16.08	0.28
10,000	16.08	0.51
100,000	41.44	2.90
1,000,000	407.53	55.05

Table 10.1: Message routing times for Psyclone and ActiveMQ, in milliseconds, for string messages of different lengths.

half a second lost by just sending messages. Half a second is a very noticeable delay in real-time interaction.

We compared the message-oriented middleware ActiveMQ, used in the SEMAINE API, with the blackboard middleware Psyclone (CMLabs, 2007), which is used in systems similar to ours (e.g., Niewiadomski et al., 2009). In order to compute the mere message routing time ignoring network latencies, we ran either ActiveMQ 5.1.0 or Psyclone 1.1.7 on a Windows Vista machine with a Core2Duo 2.5 GHz processor and 4 GB RAM with no other load, and connected to each using a Java client sending and receiving messages in sequence from the localhost machine. We sent text messages of different lengths to each middleware in a loop, averaging measures over 100 repetitions for each message length. We used plain string messages with lengths between 10 and 1,000,000 characters. The message routing times are shown in Table 10.1 and Figure 10.1. Between 10 and 1,000 characters, round trip message routing times for ActiveMQ are approximately constant at around 0.3 ms; the times rise to 0.5 ms for 10,000 characters, 2.9 ms for 100,000, and 55 ms for messages of 1,000,000 characters length. Psyclone is substantially slower, with routing times approximately constant around 16 ms for messages from 10 to 10,000 characters length, then rising to 41 ms at 100,000 characters length and 408 ms at 1,000,000 characters message length.

These results show that in this task, ActiveMQ is approximately 50 times faster than Psyclone for short messages, and around 10 times faster for long messages. While it may be possible to find even faster systems, it seems that ActiveMQ is reasonably fast for our purposes.

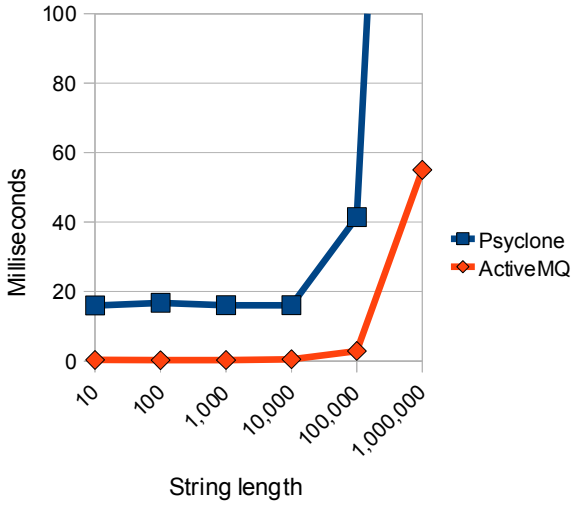

Figure 10.1: Round-trip message routing times as a function of message length.

10.2 Architecture

In assessing the performance of a real-time interactive system, many factors influence the reaction times as observable by the human user, including:

- the message routing times;
- the processing times of the components;
- the available hardware resources;
- the way the components are wired together, i.e. the system architecture;
- and the algorithms for triggering system actions as a function of user actions and other information.

The first item in this list (message routing) has been investigated above; the last item (algorithms) is outside the intended scope of this thesis, as defined in Section 1.3.

In order to assess the effect of the remaining three factors on the system's reactivity, we proceeded as follows. We ran the SEMAINE-3.0 system in four configurations differing in available hardware resources and the processing times

of components. One hardware setup consisted in all components running on a single Windows laptop with 4 GB RAM and a dual-core 2.5 GHz CPU; the second hardware setup was a distributed system, with video input and output components running on the Windows laptop, and speech input, dialogue, and speech synthesis components running on a Mac Book Pro with 8 GB RAM and a dual-core 3.06 GHz CPU. The two laptops were connected via a direct Ethernet cable in order to minimise the latency introduced by network routing.

The component processing times were controlled by switching on and off the cache in one of the most processing-intensive components, the MARY TTS speech synthesis. In one configuration, the cache was enabled, so that TTS processing could be effectively skipped for previously generated utterances; in the other configuration, the cache was deactivated, forcing TTS processing for every utterance. In the cache-enabled condition, the system was left running for a while before the experiment, in order to make sure that many of the typical system utterances were contained in the cache.

In order to assess the effect of system architecture, we compared, in each of the four system setups, the measurements for utterances generated along the direct branch and utterances generated along the prepare-and-trigger branch (see Sections 7.3.3 and 7.3.4).

The measure used is time-to-animation: the time it takes from the moment when the action selection has decided that an action should be performed to the moment the action starts playing. A dedicated measurement component was implemented to observe the messages passed between components. It measured the time between the action selection's message triggering an utterance and the time when the player emits a callback message stating that the playback of that utterance has started.

The measurements were taken during actual system runs, i.e. while the system was interacting with a user, over a period of several minutes. This approach yields measures that are only approximately comparable: since in every system run, the user's behaviour is different, the utterances generated in response are not the same. For this reason, neither the identity nor the number of system utterances is exactly the same across the different system runs. Furthermore, the number of utterances generated along the direct branch and the prepare-and-trigger branch vary substantially, depending on the ability of the utterance action proposer to anticipate

Test setup	Direct branch	Prepare-and-trigger branch
Distributed system, no cache in MARY TTS	687 ms (n=82)	7 ms (n=16)
Distributed system, cache enabled in MARY TTS	201 ms (n=80)	5 ms (n=22)
Single PC, no cache in MARY TTS	846 ms (n=46)	7 ms (n=22)
Single PC, cache enabled in MARY TTS	407 ms (n=41)	18 ms (n=39)

Table 10.2: Median time-to-animation in various system setups, in milliseconds. n is the number of utterances observed.

its own future decisions in time to allow for the use of the prepare-and-trigger branch. The extent to which the utterance action proposer is able to make such predictions is part of the application logic and therefore outside the scope of this investigation. Therefore, the number of utterances generated through one or the other branch cannot be controlled or assessed; the interesting information is any difference in time-to-animation between these branches. These measurement conditions mean that results for individual utterances cannot be compared. However, any consistent difference between the averages over many utterances per condition are likely to be generic and robust.

Table 10.2 shows the results of the measurements of time-to-animation in the different system setups. All three factors show observable effects.

The most important effect is achieved by using the prepare-and-trigger branch in the architecture. Whenever it is used, this reduces the time-to-animation to around 10 ms for all test setups. This suggests that the most effective means for a real-time interactive system to improve its reactivity is for the action selection system to make as extensive use of the prepare-and-trigger facility as possible. For utterances realised through the prepare-and-trigger branch, no substantial effects of hardware resources and component processing times can be observed.

The time-to-animation along the direct branch is much longer, of the order of hundreds of milliseconds. Any differences at this scale are likely to be perceivable by human users (Ward and Tsukahara, 2000), so that any improvements are

highly relevant. Here, we can observe that both the hardware resources and the component processing times play a substantial role.

The reduced processing time of the TTS component in the cache-enabled condition is reflected in a reduction of median processing times to less than half, for both hardware settings. This is fully consistent with expectations for the direct branch: where the components are executed in sequence, speeding up one of the major elements in the sequence should have an observable effect on overall reaction times.

Regarding hardware resources, it can be seen that distributing the load across two machines also helps speed up the generation of output. The effect is clearly visible, but it is the smallest of all three. This suggests that the hardware resources in the single PC setup were scarce to some extent but not excessively.

The best improvement to the system's reaction times is achieved by combining all three effects. In the best setup – the distributed system with cache enabled –, the median time-to-animation is only 200 ms in the direct branch, and below 10 ms in the prepare-and-trigger branch.

10.3 Conclusion

This chapter has assessed the system's performance on a technical level. Four aspects were addressed: the low-level message-routing times of the middleware used for communication between the components; the system architecture, with respect to direct and prepare-and-trigger rendering of the system's behaviour; the effect of limited hardware resources; and the effect of processing times in a single component.

Findings confirmed that all these aspects play an important role for the reactivity of the system. It was also shown that in the best system configuration, with the SEMAINE-3.0 system running as a distributed system with the MARY TTS cache enabled, the median time-to-animation for system utterances is only 200 ms, which is less than the 350 to 700 ms that Ward and Tsukahara (2000) found to be the typical delays for human backchannel reactions. Furthermore, the underlying middleware has been shown to be very fast – less than one millisecond – for messages up to 10 kB size, and still good – with message routing time of about 3 ms – for messages of 100 kB size.

In essence, we have shown that the SEMAINE API provides a good basis for implementing highly reactive real-time interactive systems.

Chapter 11

Re-use: Building new emotion-oriented systems with the SEMAINE API

One aim of the SEMAINE API is to support the simple creation of new emotion-oriented systems, composed of new and existing components. The use of standard interfaces between components, as well as the component integration and XML handling support provided by the SEMAINE API, are aimed at making this task comparatively easy.

This section presents three emotion-oriented example systems, in order to corroborate the claim that the SEMAINE API is easy to use for building new emotion-oriented systems out of new and/or existing components. Source code is provided in order to allow the reader to follow in detail the steps needed for using the SEMAINE API. The code is written in Java, and can be obtained from the SEMAINE download page at `http://semaine.opendfki.de`. A C++ version of the first example is also available. Since the relevant parts of the code look very similar in C++ and in Java, we only show the Java version here.

11.1 Hello world

The "Hello" example realises a simple text-based interactive system. The user types arbitrary text; an analyser component spots keywords, and deduces an af-

```
 1 public class HelloInput extends Component {
 2
 3   private Sender textSender =
         new Sender("semaine.data.hello.text", "TEXT", getName());
 4   private BufferedReader inputReader =
         new BufferedReader(new InputStreamReader(System.in));
 5
 6   public HelloInput() throws JMSException {
 7     super("HelloInput", true/*is input*/, false);
 8     senders.add(textSender);
 9   }
10
11   @Override protected void act() throws IOException, JMSException {
12     if (inputReader.ready()) {
13       String line = inputReader.readLine();
14       textSender.sendTextMessage(line, meta.getTime());
15     }
16   }
17 }
```

Figure 11.1: The HelloInput component sending text messages via the SEMAINE API.

fective state from them; and a rendering component outputs an emoticon corresponding to this text. Despite its simplicity, the example is instructive because it displays the main elements of an emotion-oriented system.

The input component (Figure 11.1) simply reads one line of text at a time, and sends it on. It has an input device (Figure 11.1, line 4) and a Sender writing TEXT data to the Topic '`semaine.data.hello.text`' (line 3). In its constructor, the component registers itself as an input component (l. 7), and registers its sender (l. 8). Its `act()` method, which is automatically called every 100 ms while the system is running, checks for new input (l. 12), reads it (l. 13), and sends it to the Topic (l. 14).

As a simplistic central processing component, the HelloAnalyser (Figure 11.2) makes emotional judgements about the input. It registers a Receiver (l. 7) for the Topic that HelloInput writes to, and sets up (l. 3) and registers (l. 8) an XMLSender producing data of type EmotionML. Whenever a message is received, the method `react()` is called (l. 11). It receives (l. 13) and analyses (l. 14-17) the input

		Valence		
		-	**0**	**+**
Arousal	**+**	8-(	8-|	8-)
	0	:-(	:-|	:-)
	-	*-(	*-|	*-)

Table 11.1: Ad hoc emoticons used to represent positions in the arousal-valence plane.

text, and computes values for the emotion dimensions arousal and valence from the text. Finally, it creates an EmotionML document (l. 18) and sends it (l. 19).

As the SEMAINE API does not currently provide built-in support for standalone EmotionML documents, the component uses a generic `XMLSender` (l. 3) and uses the `XMLTool` to build up the EmotionML document (l. 23-30).

The output of the Hello system should be an emoticon representing an area in the arousal-valence plane as shown in Table 11.1. The EmoticonOutput component (Figure 11.3) registers an XMLReceiver (l. 5) to the Topic that the HelloAnalyser sends to. Whenever a message is received, the `react()` method is called (l. 8), which analyses the XML document in terms of EmotionML markup (l. 10-12), and extracts the arousal and valence values (l. 14-15). The emotion display is rendered as a function of these values (l. 17-19).

In order to build a system from the components, a configuration file is created (Figure 11.4). It includes the SystemManager component as well as the three newly created components. Furthermore, it requests a visible system manager GUI providing a message flow graph.

The system is started in the same way as all Java-based SEMAINE API systems: `java -cp 'lib/*' eu.semaine.system.ComponentRunner config/example-hello.config`. Figure 11.5 shows a screenshot of the resulting message flow graph. As the communication passes via the middleware ActiveMQ, the system would behave in the exact same way if the four components were started as separate processes, on different machines, or if some of them were written in C++ rather than Java.

```
1 public class HelloAnalyser extends Component {
2
3   private XMLSender emotionSender =
        new XMLSender("semaine.data.hello.emotion", "EmotionML", getName());
4
5   public HelloAnalyser() throws JMSException {
6     super("HelloAnalyser");
7     receivers.add(new Receiver("semaine.data.hello.text"));
8     senders.add(emotionSender);
9   }
10
11  @Override protected void react(SEMAINEMessage m) throws JMSException {
12    float arousalValue = 0.5f, valenceValue = 0.5f;
13    String input = m.getText();
14    if (input.contains("very")) arousalValue = 1;
15    else if (input.contains("a␣bit")) arousalValue = 0;
16    if (input.contains("happy")) valenceValue = 1;
17    else if (input.contains("sad")) valenceValue = 0;
18    Document emotionML = createEmotionML(arousalValue, valenceValue);
19    emotionSender.sendXML(emotionML, meta.getTime());
20  }
21
22  private Document createEmotionML(float arousalValue, float valenceValue) {
23    Document emotionML = XMLTool.newDocument(EmotionML.ROOT_TAGNAME,
                  EmotionML.namespaceURI);
24    Element emotion = XMLTool.appendChildElement(
                  emotionML.getDocumentElement(), EmotionML.E_EMOTION);
25    emotion.setAttribute(EmotionML.A_DIMENSION_VOCABULARY,
                  EmotionML.VOC_FSRE_DIMENSION_DEFINITION);
26    Element arousal = XMLTool.appendChildElement(emotion,
                  EmotionML.E_DIMENSION);
27    arousal.setAttribute(EmotionML.A_NAME,
                  EmotionML.VOC_FSRE_DIMENSION_AROUSAL);
28    arousal.setAttribute(EmotionML.A_VALUE, String.valueOf(arousalValue));
29    Element valence = XMLTool.appendChildElement(emotion,
                  EmotionML.E_DIMENSION);
30    valence.setAttribute(EmotionML.A_NAME,
                  EmotionML.VOC_FSRE_DIMENSION_VALENCE);
31    valence.setAttribute(EmotionML.A_VALUE, String.valueOf(valenceValue));
32    return emotionML;
33  }
34 }
```

Figure 11.2: The HelloAnalyser component. It receives and analyses the text messages from HelloInput, and generates and sends an EmotionML document containing the analysis results.

```
 1 public class EmoticonOutput extends Component {
 2
 3   public EmoticonOutput() throws JMSException {
 4     super("EmoticonOutput", false, true /*is output*/);
 5     receivers.add(new XMLReceiver("semaine.data.hello.emotion"));
 6   }
 7
 8   @Override protected void react(SEMAINEMessage m)
                 throws MessageFormatException {
 9     SEMAINEXMLMessage xm = (SEMAINEXMLMessage) m;
10     Element emotion = (Element) xm.getDocument().getElementsByTagNameNS(
                 EmotionML.namespaceURI, EmotionML.E_EMOTION).item(0);
11     String vocabularyURI = emotion.getAttribute(
                 EmotionML.A_DIMENSION_VOCABULARY);
12     if (!vocabularyURI.equals(EmotionML.VOC_FSRE_DIMENSION_DEFINITION))
                 return;
13     List<Element> dimensions = XMLTool.getChildrenByLocalNameNS(emotion,
                 EmotionML.E_DIMENSION, EmotionML.namespaceURI);
14     float a = 0.5f, v = 0.5f; // neutral values
15     boolean haveSomething = false;
16     for (Element dim : dimensions) {
17       String name = dim.getAttribute(EmotionML.A_NAME);
18       float value = Float.parseFloat(dim.getAttribute(EmotionML.A_VALUE));
19       if (name.equals(EmotionML.VOC_FSRE_DIMENSION_AROUSAL)) {
20         a = value;
21         haveSomething = true;
22       } else if (name.equals(EmotionML.VOC_FSRE_DIMENSION_VALENCE)) {
23         v = value;
24         haveSomething = true;
25       }
26     }
27     if (!haveSomething) return;
28     String eyes  = a > 0.66 ? "8" /*active*/   :
                      a < 0.33 ? "*" /*passive*/  :
                                 ":" /*neutral*/;
29     String mouth = v > 0.66 ? ")" /*positive*/ :
                      v < 0.33 ? "(" /*negative*/ :
                                 "|" /*neutral*/;
30     System.out.println(eyes+"-"+mouth);
31   }
32 }
```

Figure 11.3: The EmoticonOutput component. It receives EmotionML markup and displays an emoticon according to Table 11.1.

```
semaine.components = \
    |eu.semaine.components.meta.SystemManager| \
    |eu.semaine.examples.hello.HelloInput| \
    |eu.semaine.examples.hello.HelloAnalyser| \
    |eu.semaine.examples.hello.EmoticonOutput|

semaine.systemmanager.gui = true
```

Figure 11.4: The configuration file `example-hello.config` defining the Hello application.

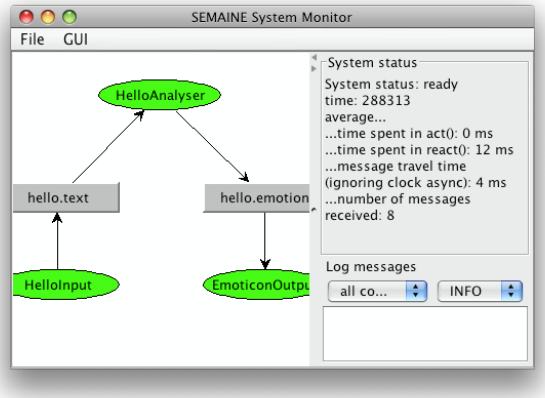

Figure 11.5: Message flow graph of the Hello system

```
 1 public class EmotionExtractor extends Component {
 2   private XMLSender emotionSender =
         new XMLSender("semaine.data.hello.emotion", "EmotionML", getName());
 3
 4   public EmotionExtractor() throws JMSException {
 5     super("EmotionExtractor");
 6     receivers.add(
              new EmmaReceiver("semaine.data.state.user.emma.emotion.voice"));
 7     senders.add(emotionSender);
 8   }
 9
10   @Override protected void react(SEMAINEMessage m) throws JMSException {
11     SEMAINEEmmaMessage emmaMessage = (SEMAINEEmmaMessage) m;
12     Element interpretation = emmaMessage.getTopLevelInterpretation();
13     List<Element> emotionElements =
           emmaMessage.getEmotionElements(interpretation);
14     if (emotionElements.size() > 0) {
15       Element emotion = emotionElements.get(0);
16       Document emotionML = XMLTool.newDocument(EmotionML.ROOT_ELEMENT,
                                                  EmotionML.namespaceURI);
17       emotionML.adoptNode(emotion);
18       emotionML.getDocumentElement().appendChild(emotion);
19       emotionSender.sendXML(emotionML, meta.getTime());
20     }
21   }
22 }
```

Figure 11.6: The EmotionExtractor component takes EmotionML markup from an EMMA message and forwards it.

11.2 Emotion mirror

The Emotion mirror is a variant of the Hello system. Instead of analysing text and deducing emotions from keywords, it uses the openSMILE speech feature extraction and emotion detection (see Chapter 7) for interpreting the user's emotion. The output is rendered using the same EmoticonOutput component from the Hello system in Section 11.1.

Only one new component is needed to build this system. EmotionExtractor (Figure 11.6) has an emotion Sender (l. 2 and l. 7) just like the HelloAnalyser had, but uses an EMMA Receiver (l. 6) to read from the topic that the Emotion detection component from the SEMAINE system (see Chapter 7) publishes to, as

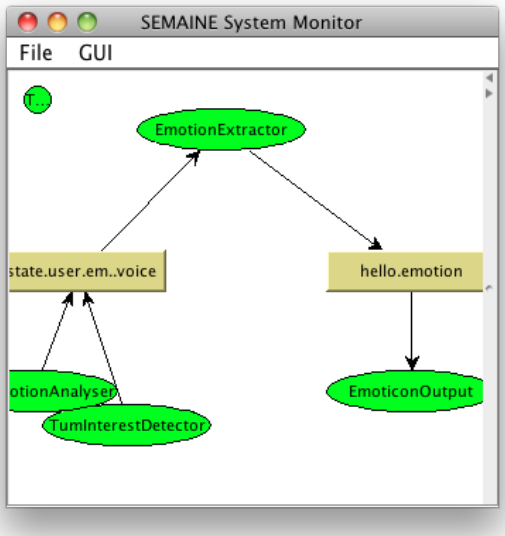

Figure 11.7: Message flow graph of the Emotion mirror system.

documented in Schröder et al. (2010b). Upon reception of an EMMA message, the method `react()` is called (l. 10). As the only receiver registered by the component is an EMMA receiver, the message can be directly cast into an EMMA message (l. 11) which allows for comfortable access to the document structure to extract emotion markup (l. 12-13). Where emotion markup is present, it is inserted into a standalone EmotionML document (l. 16-18) and sent to the output Topic (l. 19).

The config file contains only the components SystemManager, EmotionExtractor and EmoticonOutput. As the SMILE component is written in C++, it needs to be started as a separate process as documented in the SEMAINE wiki documentation at `http://semaine.opendfki.de`. The resulting message flow graph is shown in Figure 11.7.

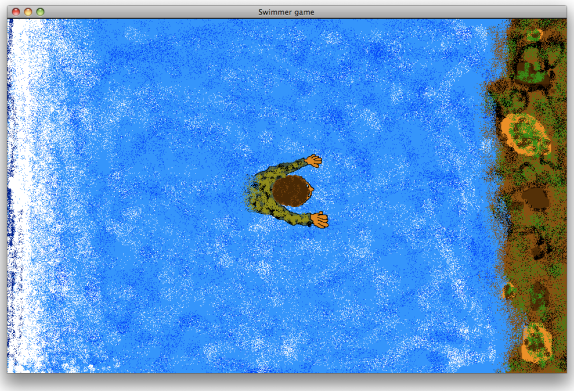

Figure 11.8: Swimmer's game user interface.

11.3 A game driven by emotional speech: The swimmer's game

The third example system is a simple game application in which the user must use emotional speech to win the game. The game scenario is as follows. A swimmer is being pulled backwards by the stream towards a waterfall (Figure 11.8). The user can help the swimmer to move forward towards the river bank by cheering him up through high-arousal speech. Low arousal, on the other hand, discourages the swimmer and drives him more quickly to the waterfall.

The system requires the openSMILE components as in the Emotion mirror system; a component computing the swimmer's position as time passes, and considering the user's input; and a rendering component for the user interface. Furthermore, we will illustrate the use of TTS output in the SEMAINE API by implementing a commentator providing input to the speech synthesis component of the SEMAINE system (Chapter 7).

The PositionComputer (Figure 11.9) combines a `react()` and an `act()` method. Messages are received via an EMMA receiver and lead to a change in the internal parameter `position` (l. 22). The `act()` method implements the backward drift (l. 29) and sends regular position updates (l. 30) as a plain-text message.

```
1 public class PositionComputer extends Component {
2   private Sender positionSender =
        new Sender("semaine.data.swimmer.position", "TEXT", getName());
3   private float position = 50;
4
5   public PositionComputer() throws JMSException {
6     super("PositionComputer");
7     receivers.add(
         new EmmaReceiver("semaine.data.state.user.emma.emotion.voice"));
8     senders.add(positionSender);
9   }
10
11  @Override protected void react(SEMAINEMessage m)
                throws MessageFormatException {
12    SEMAINEEmmaMessage emmaMessage = (SEMAINEEmmaMessage) m;
13    Element interpretation = emmaMessage.getTopLevelInterpretation();
14    if (interpretation == null) return;
14    List<Element> emotionElements =
                emmaMessage.getEmotionElements(interpretation);
15
16    for (Element emotion : emotionElements) {
17      List<Element> dimensions = XMLTool.getChildrenByLocalNameNS(
                emotion, EmotionML.E_DIMENSION, EmotionML.namespaceURI);
18      for (Element dim : dimensions) {
19        if (dim.getAttribute(EmotionML.A_NAME).equals(
                EmotionML.VOC_FSRE_DIMENSION_AROUSAL)) {
20          float arousalValue = Float.parseFloat(
                dim.getAttribute(EmotionML.A_VALUE));
21          // Arousal influences the swimmer's position:
22          position += 10*(arousalValue-0.4f);
23          break;
24        }
25      }
26    }
27  }
28
29  @Override protected void act() throws JMSException {
30    // The river slowly pulls back the swimmer:
31    position -= 0.1;
32    positionSender.sendTextMessage(String.valueOf(position),
                                     meta.getTime());
33  }
34 }
```

Figure 11.9: The PositionComputer component.

```
 1 public class SwimmerDisplay extends Component {
 2
 3   public SwimmerDisplay() throws JMSException {
 4     super("SwimmerDisplay", false, true/*is output*/);
 5     receivers.add(new Receiver("semaine.data.swimmer.position"));
 6     setupGUI();
 7   }
 8
 9   @Override protected void react(SEMAINEMessage m) throws JMSException {
10     float percent = Float.parseFloat(m.getText());
11     updateSwimmerPosition(percent);
12     String message = percent <= 0 ? "You␣lost!" :
                        percent >= 100 ? "You␣won!!!" : null;
13     if (message != null) {
         ...
       }
     }
     ...
   }
```

Figure 11.10: The SwimmerDisplay component (GUI code not shown).

The SwimmerDisplay (Figure 11.10) implements the user interface shown in Figure 11.8. Its messaging part consist of a simple text-based Receiver (l. 5) and an interpretation of the text messages as single float values (l. 10).

Due to the separation of position computer and swimmer display, it is now very simple to add a Commentator component (Figure 11.11) that generates comments using synthetic speech, as a function of the current position of the swimmer. It subscribes to the same Topic as the SwimmerDisplay (l. 7), and sends BML output (l. 2) to the Topic serving as input to the speech synthesis component of the SEMAINE system (Schröder et al., 2010b). Speech output is produced when the game starts (l. 18-20) and when the position meets certain criteria (l. 13-14). Generation of speech output consists in the creation of a simple BML document with a <speech> tag enclosing the text to be spoken (l. 25-28), and sending that document (l. 29).

The complete system consists of the Java components SystemManager, PositionComputer, SwimmerDisplay, Commentator, SpeechBMLRealiser and SemaineAudioPlayer, as well as the external C++ component openSMILE. The resulting message flow graph is shown in Figure 11.12.

```
1 public class Commentator extends Component {
2   private BMLSender bmlSender =
          new BMLSender("semaine.data.synthesis.plan", getName());
3   private boolean started = false;
4
5   public Commentator() throws JMSException {
6     super("Commentator");
7     receivers.add(new Receiver("semaine.data.swimmer.position"));
8     senders.add(bmlSender);
9   }
10
11  @Override protected void react(SEMAINEMessage m) throws JMSException {
12    float percent = Float.valueOf(m.getText());
13    if (percent < 30 /*danger*/) say("Your␣swimmer␣needs␣help!");
14    else if (percent > 70 /*nearly there*/) say("Just␣a␣little␣more.");
15  }
16
17  @Override protected void act() throws JMSException {
18    if (!started) {
19      started = true;
20      say("The␣swimmer␣needs␣your␣support␣to␣reach␣the␣river␣bank.␣" +
            "Cheer␣him␣up!");
21    }
22  }
23
24  private void say(String text) throws JMSException {
25    Document bml = XMLTool.newDocument(BML.ROOT_TAGNAME, BML.namespaceURI);
26    Element speech = XMLTool.appendChildElement(
                bml.getDocumentElement(), BML.E_SPEECH);
27    speech.setAttribute("language", "en-US");
28    speech.setTextContent(text);
29    bmlSender.sendXML(bml, meta.getTime());
30  }
31 }
```

Figure 11.11: The Commentator component, producing TTS requests.

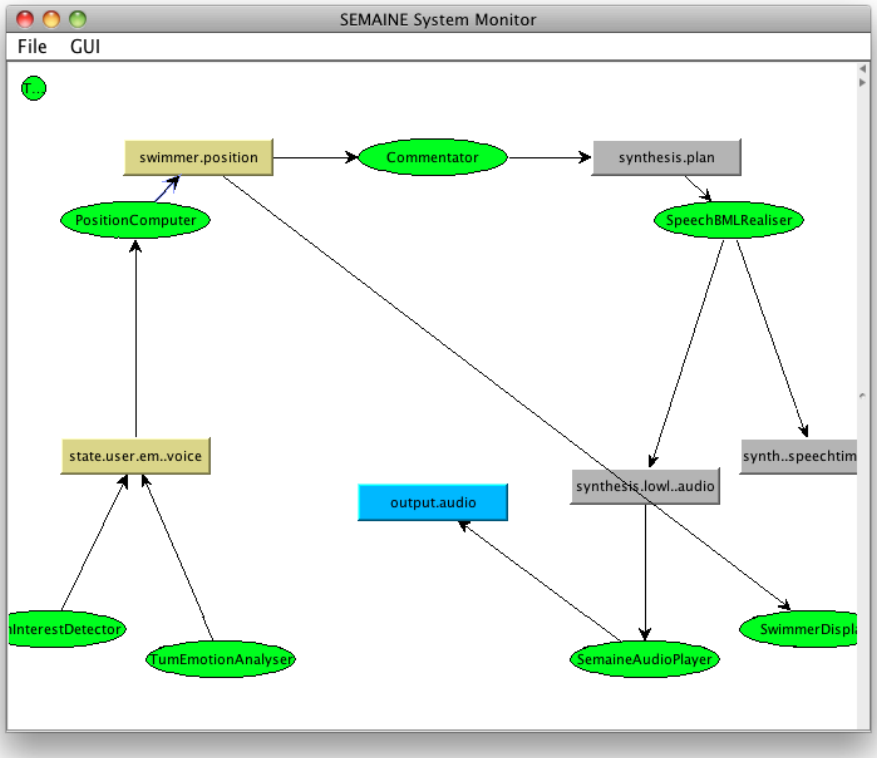

Figure 11.12: Message flow graph of the swimmer's game system.

11.4 Conclusion

This chapter has illustrated the claim that it is easy to build new emotion-oriented systems based on the SEMAINE API. Three examples have been presented that show how to build a new system from scratch consisting exclusively of new components; to re-use components in new contexts; and to combine standard and custom representation formats as appropriate. These examples should allow the interested reader to get started with using the SEMAINE API to build their own emotion-oriented systems.

Chapter 12

Summary and Outlook

The present thesis has described a Component Integration Framework for an Embodied Conversational Agent (ECA) that supports the provision of capabilities for natural interaction and emotionally competent behaviour.

The thesis has started with an introduction and motivation, setting the scene for the reasons why research in emotional competence for interactive systems is required. Notably, we have pointed out the relevance of emotional and social intelligence for machines, due to an apparent human tendency to treat complex interactive entities as social entities. In this context we have also motivated the interest in standards, as a means for promoting re-use and interoperability of technology, which makes cumulative research easier.

We have reviewed the state of the art in ECA research with a special emphasis on natural interaction and emotion-related capabilities. We reviewed different types of ECA systems before focusing on one-to-one human-machine interaction scenarios since these are most relevant for the current thesis. We discussed the capabilities of current systems to express and perceive emotions, as well as existing work on responsiveness in interaction, and showed the limitations of current systems as well as open research questions.

Zooming in on the technology enabling these capabilities, we reviewed a number of relevant component integration frameworks for interactive ECAs and related technology. We identified a number of relevant criteria, including issues of design choice such as the types of representation used or the existence of a system manager component, but also practical issues such as the de facto availability, active

maintenance, support for operating systems and programming languages etc. We formulated requirements for the component integration framework to be used for building the SEMAINE system, and concluded that none of the existing component integration frameworks satisfied all requirements.

We then reviewed a broad range of communication middlewares which could potentially be used for the communication layer in a new component integration framework. Immediate candidates for use in the current context were messaging, remote-invocation and blackboard middlewares. We also briefly reviewed web services, including Service-Oriented Architecture and RESTful HTTP, as well as specialised middlewares for linguistics and multimedia.

As the center piece of software infrastructure developed in this the thesis, we have described the SEMAINE API, the component integration framework we developed for the SEMAINE project. On the basis of a set of criteria for choosing a middleware, we decided to use the message-oriented middleware ActiveMQ as the basis for component integration. We described the framework's architecture, which complements the immediate communication between components with a meta-communication layer through which a central system manager maintains contact with each component. This allows for a number of relevant features. For example, the system manager can become aware of defaulting components and request other components to wait until the system is ready again. The system manager's GUI can display a message-flow graph in which the components and their interconnections are shown. The system manager also provides a synchronised system time to all components. Through a centralised logging mechanism, it is possible to observe the components as well as the messages flowing between them from the system GUI. We have described the SEMAINE API's support for the simple creation of components and for sending and receiving messages in one of the supported representation languages, as feature vectors, or as generic XML or binary data. We have pointed out that the SEMAINE API exists for Java and C++, providing the same interfaces in both languages.

As a key instance of a full-scale system built on top of the SEMAINE API, we have presented the SEMAINE system, implementing a Sensitive Artificial Listener capable of interacting in real time with a user with multimodal emotion detection and generation. We have presented the simpler "pipeline" architecture of the SEMAINE-2.0 system, which basically connects the analysis, action planning

and generation components into a long pipeline. As an extension to this, we presented the "prepare-and-trigger" architecture which was added in SEMAINE-3.0, and explained how this approach, based on the concept of Competitive Queuing from cognitive neuroscience, allows us to prepare potential future utterances ahead of time so as to have them immediately ready when they are indeed selected.

Regarding the communication between system components enabled and supported by the SEMAINE API, we have first presented an overview of relevant representation languages. We have described how, in particular, EMMA is used for representing results of the analysis of user behaviour; how EmotionML is used for representing emotions detected from the user; how the domain-specific language SemaineML is used for representing non-standard information; how SSML is used for representing speech synthesis input; how FML, or its preliminary approximation FML-APML, represents elements of the meaning of multimodal utterances to be generated by the system; and how the actual multimodal behaviour is represented as BML. We have furthermore described the mechanism provided in the SEMAINE API for representing callback messages, and presented a novel mechanism for defining a customisable set of state information and its representation in XML messages for communicating state between components.

Given the central role that a representation of emotions plays for an emotionally competent system, we have described in detail how the Emotion Markup Language has been specified at the World Wide Web consortium (W3C). After an outline of the process itself, which has started five years ago and is still ongoing, we have reviewed previous and related work. We then reported on the use cases and requirements identified in the Emotion Incubator group, and on the prioritisation of requirements carried out in the Emotion Markup Language Incubator group at the W3C. We have presented the syntax of EmotionML as it was defined in the W3C's Multimodal Interaction working group. We have related the language to scientific descriptions of emotion and presented a selection of vocabularies for representing emotions in terms of categories, dimensions, appraisals and action tendencies. We have discussed the problem of Schema-validating a language such as EmotionML which allows for custom vocabularies, and presented an alternative solution, which would have allowed for Schema-validation of custom vocabularies but which was rejected in the standardisation process because it was not consistent with usual practice at the W3C with respect to XML namespaces. We concluded

the chapter on EmotionML with a short review of open issues which will require more work in future extensions of EmotionML.

We assessed the SEMAINE API as a component integration framework from two perspectives. First, we measured the performance of two critical aspects. The speed of message routing in the underlying middleware ActiveMQ was shown to be 10-50 times faster than the alternative solution Psyclone, depending on the message size. Furthermore, we investigated the time-to-animation, i.e. the time between the decision to execute a certain action and its start, as a measure of the real-time performance of the system. We found that the most important determinant of this time is the use of the prepare-and-trigger branch. The dramatic improvement achieved by using this architectural choice encourages system builders to make extensive use of this type of architectural choice. However, the execution speed of individual components and the use of a distributed system were also found to be important determinants of time-to-animation: faster components and sufficient processing resources both help to improve the system's reaction times.

As a second type of assessment of the SEMAINE API, we investigated the ease with which it is possible to use the framework to build new emotion-oriented systems. We illustrated the process using three demo systems: a "hello world" system, combining dummy input, analysis and output components of 15-30 lines of code each, using EmotionML for representing the emotion "analysed" from the data; an "emotion mirror" system, combining parts of the SEMAINE system and the "hello" system into an emotional mimicry device using a single 20-line component; and a game driven by emotional speech, which reuses emotion detection and speech synthesis components from the SEMAINE system to build a simple game. Again, the source code of each new component is short enough to fit onto a single page each. We concluded that the reuse of the SEMAINE API for novel purposes is possible with very limited effort, so that the framework can indeed serve its purpose of enabling reuse and cumulative research in the area of emotion-oriented technology.

While we believe that the SEMAINE API is a worthwhile contribution to the research landscape as it stands, there are multiple areas where improvements would be beneficial.

One area for future improvement is the automatic verification whether components respect the agreed interfaces. For XML-based interfaces as used in the current framework, a good practice solution for this task is to perform namespace-aware Schema validation of the messages passed between components. Current practice in the SEMAINE API is to use a Topic as an information "hub" through which only information of a single kind should be communicated. In order to verify this, the framework could define a Schema that all messages passing via that Topic must obey. Despite the emerging standard nature of the representations used in the SEMAINE API, for most of the formats a Schema does not yet exist, so that this method requires substantial specification efforts before it can be realised.

While automatic verification of syntactically correct messages is useful, it is only partly able to detect misbehaving components. Additional mechanisms for automated testing would ease the development and, notably, the debugging process substantially. A mechanism would be very useful which allowed for the automated "integration" testing of individual components and of sub-systems: do the components and sub-systems react to well-defined stimulus conditions in the expected way? Such integration tests could be formulated once and be automatically run from then on, for example every night. As programming progresses, any regressions of previously working functionality would be automatically detected and could be fixed without requiring long hours of debugging of errors in conditions which can sometimes be difficult to reproduce manually.

A third area for future improvements is a stronger separation of the framework from the underlying middleware. If the framework could be used with different middlewares in addition to ActiveMQ, this would likely promote its reuse in different environments. For example, the Aliz-E project uses the middleware Urbi for component interaction; if Urbi could be used as a replacement for ActiveMQ, then reusing the SEMAINE API in Aliz-E would have been an option.

A final area for improvement concerns the approach to state information that is currently implemented in the SEMAINE API. The current mechanism of enumerating and circulating all relevant elements of state information works for simple, minimalistic domain models as in SEMAINE; for more complex, ontology-based models, which might involve reasoning components, this approach does not seem realistic. Instead, the framework would have to be extended to provide a request-

response mechanism through which components could request relevant aspects of the system's information state from a dedicated reasoning component or similar.

Due to its open source nature, it is possible to extend the SEMAINE API in these or other respects if and when suitable resources become available.

Appendix A

Protocol for the Player in SEMAINE

Any player component in SEMAINE must implement the following protocol so that it supports ahead-of-time preparation of possible utterances. The player must keep a collection of ”Animations” which can be played by a ”playCommand”.

This protocol is currently implemented by two players: The audio-visual Windows native Player-Ogre using the Greta agent, and the speech-only player in Java class eu.semaine.components.mary.QueuingAudioPlayer.

A.1 Data flow

Low-level player data is sent to the player via the Topics

```
semaine.data.synthesis.lowlevel.*
```

currently

```
semaine.data.synthesis.lowlevel.audio
semaine.data.synthesis.lowlevel.video.FAP
semaine.data.synthesis.lowlevel.video.BAP
```

Incoming messages have the following properties:

- a message type specific to the payload format (currently: BytesMessage for audio, TextMessage for FAP and BAP).

- a data type (obtained by message.getDatatype()) identifying the type of message (current values are ”AUDIO”, ”FAP” and ”BAP”).
- a content ID and a content creation time (obtained by message.getContentID() and message.getContentCreationTime()) which are used to assemble an Animation, to match data and command messages, and to identify the content item in callback and log messages.

The idea is that a unit of player data (an ”Animation”) is assembled in the player from the individual data items that are coming in (currently, AUDIO, FAP and BAP). Certain data types are optional (currently: AUDIO). A message can either contain the complete data of the given type (currently the case for AUDIO) or it can contain a chunk of data (currently the case for FAP and BAP). A chunk contains information about its position in the Animation; it can be dynamically added even if the Animation is already playing.

A.2 Command messages

There are two types of command messages: messages with data types `dataInfo` and `playCommand`.

- Data info commands: For a given content ID, define the data types that must be present in the Animation: HASAUDIO, HASFAP and HASBAP. Each can be 0 for ”not needed” or 1 for ”needed”.
- Player commands: For a given content ID, define the playback conditions. This includes the following aspects:
  - STARTAT: when to start the playback of the Animation (in milliseconds from the moment when the Animation becomes ready);
  - PRIORITY: the priority of the Animation in case of competing Animations;
  - LIFETIME: the lifetime of the Animation, counting from the moment when the animation becomes ready. When the lifetime is exceeded and the animation has not started playing, it will be marked as ”dead” and removed.

Commands are sent to topic

`semaine.data.synthesis.lowlevel.command`

and have the data type `dataInfo` for data info commands and `playCommand` for player start trigger commands.

For every content ID, a playCommand is required in order to play that animation. Without a matching playCommand, an animation will never be played.

A command has the following format:

- its content ID is identical to the content ID of the Animation for which it defines playback conditions;
- message format is TextMessage; the text consists of space-separated key-value pairs, one pair per line, where the keys are strings and the values are floating point numbers.

The following features are used:

- for playCommand:
  - STARTAT (0 means start at the moment when all required parts are present, a positive number means milliseconds after that condition is met)
  - LIFETIME (in milliseconds from the moment the animation is triggered; -1 means it will never expire)
  - PRIORITY (a value between 0 and 1, where 0 is the lowest and 1 the highest possible priority)
- for dataInfo:
  - HASAUDIO (a binary feature, 0 means the Animation does not have audio, 1 means the Animation has audio data)
  - HASFAP (a binary feature, 0 means the Animation does not have FAP data, 1 means the Animation has FAP data)
  - HASBAP (a binary feature, 0 means the Animation does not have BAP data, 1 means the Animation has BAP data)

Every player command must contain all features of its respective type.

A.3 Callback messages

Event-based callback messages are sent when certain conditions are met for a given Animation. The messages go to Topic

`semaine.callback.output.Animation`

and have the following format:

```
<callback xmlns="http://www.semaine-project.eu/semaineml">
   <event id="CONTENT_ID" time="META_TIME" type="EVENT_TYPE"/>
</callback>
```

where content ID and meta time are like before, and type is one of the following:

- `ready` means the Animation has received all required data, so it is ready for playing back. This event is triggered independently of the question whether a command has been received or not.
- `deleted` means the Animation was removed before it started playing, e.g. because it has exceeded its lifetime in the output queue.
- `start` means the Animation has started playing.
- `stopped` means the Animation was stopped while playing but before it was finished, e.g. because a request to change character was received.
- `end` means the Animation has finished playing.

A.4 Error conditions

The content ID must be unique for the lifetime of a system. This leads to the following error conditions.

It is an error condition...

- if a data chunk is received for an Animation that has already been discarded (because it finished playing, or exceeded its lifetime in the queue);
- if data is received for a data type that does not form chunks but data of that type has already been received for the given content ID;

- if a playCommand is received for a content ID that has been started already, or that is already discarded;

An error condition should be reported as a WARN log message, and otherwise ignored.

It is not an error condition...

- if a second playCommand is received after an animation has become ready but before it started playing. In this case, the new priority etc. overwrites the previous values.

Bibliography

Acosta, J. C. (2009). *Using Emotion to Gain Rapport in a Spoken Dialog System*. PhD thesis, University of Texas at El Paso.

Albrecht, I., Schröder, M., Haber, J., and Seidel, H. (2005). Mixed feelings: expression of non-basic emotions in a muscle-based talking head. *Virtual Reality*, 8(4):201–212. Available from: http://portal.acm.org/citation.cfm?id=1086350.1086352.

Allen, J., Ferguson, G., and Stent, A. (2001a). An architecture for more realistic conversational systems. In *Proc. Intelligent User Interfaces*, pages 1–8, Santa Fe, USA. Available from: http://portal.acm.org/citation.cfm?id=359822.

Allen, J. F., Byron, D. K., Dzikovska, M., Ferguson, G., Galescu, L., and Stent, A. (2001b). Toward conversational human-computer interaction. *AI magazine*, 22(4):27–37.

Allwood, J., Nivre, J., and Ahlsén, E. (1992). On the semantics and pragmatics of linguistic feedback. *Journal of Semantics*, 9(1):1–26. Available from: http://jos.oxfordjournals.org/cgi/content/abstract/9/1/1.

AMQP Working Group (2010). AMQP 1-0 revision 0. Technical report. Available from: http://www.amqp.org/.

André, E., Concepcion, K., Mani, I., and Guilder, L. (2005). Autobriefer: A system for authoring narrated briefings. In Stock, O. and Zancanaro, M., editors, *Multimodal intelligent information presentation*, page 143–158.

André, E. and Pelachaud, C. (2010). Interacting with embodied conversational agents. In Chen, F. and Jokinen, K., editors, *Speech technology*. Springer, New York. Available from: http://dx.doi.org/10.1007/978-0-387-73819-2_8.

André, E., Rist, T., and Muller, J. (1999). Employing AI methods to control the behavior of animated interface agents. *Applied Artificial Intelligence*, 13(4):415–448.

André, E., Rist, T., van Mulken, S., Klesen, M., and Baldes, S. (2000). The automated design of believable dialogues for animated presentation teams. In Cassell, J., Prevost, S., Sullivan, J., and Churchill, E., editors, *Embodied conversational agents*, page 220–255. MIT Press.

Apache Software Foundation (2008). Apache ActiveMQ. Available from: http://activemq.apache.org/.

Apache Software Foundation (2009). The apache Xerces project. Available from: http://xerces.apache.org/.

Apache Software Foundation (2010). Apache UIMA. Available from: http://incubator.apache.org/uima/.

Banavar, G., Chandra, T., Strom, R., and Sturman, D. (1999). A case for message oriented middleware. In *Distributed Computing*, pages 846–863. Springer. Available from: http://dx.doi.org/10.1007/3-540-48169-9_1.

Banse, R. and Scherer, K. R. (1996). Acoustic profiles in vocal emotion expression. *Journal of Personality and Social Psychology*, 70(3):614–636.

Baron-Cohen, S., Ring, H. A., Wheelwright, S., Bullmore, E. T., Brammer, M. J., Simmons, A., and Williams, S. C. R. (1999). Social intelligence in the normal and autistic brain: an fMRI study. *European Journal of Neuroscience*, 11(6):1891–1898.

Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37:122–125. Available from: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/ba-and-emotion.ps.

Batliner, A., Steidl, S., Schuller, B., Seppi, D., Laskowski, K., Vogt, T., Devillers, L., Vidrascu, L., Amir, N., and Kessous, L. (2006). Combining efforts for improving automatic classification of emotional user states. In *Proc. First International Language Technologies Conference, IS-LTC 2006*, Ljubljana, Slovenia.

Batliner, A., Steidl, S., Schuller, B., Seppi, D., Vogt, T., Wagner, J., Devillers, L., Vidrascu, L., Aharonson, V., Kessous, L., et al. (2011). Whodunnit - searching for the most important feature types signalling emotion-related user states in speech. *Computer Speech & Language*, 25(1):4–28.

Becket, D. and McBride, B. (2004). RDF/XML syntax specification (Revised). W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/rdf-syntax-grammar/.

Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., and Siméon, J. (2007). XML path language (XPath) 2.0. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/2007/REC-xpath20-20070123/.

Bernstein, P. A. (1996). Middleware: a model for distributed system services. *Commun. ACM*, 39(2):86–98. Available from: http://portal.acm.org/citation.cfm?id=230809.

Bevacqua, E., Mancini, M., Niewiadomski, R., and Pelachaud, C. (2007). An expressive ECA showing complex emotions. In *Proceedings of the AISB Annual Convention*, page 208–216, Newcastle, UK.

Bevacqua, E., Mancini, M., and Pelachaud, C. (2008). A listening agent exhibiting variable behaviour. In *Proc. Intelligent Virtual Agents*, pages 262–269. Tokyo, Japan. Available from: http://dx.doi.org/10.1007/978-3-540-85483-8_27.

Biocca, F., Burgoon, J., Harms, C., and Stoner, M. (2001). Criteria and scope conditions for a theory and measure of social presence. In *Presence 2001*, Philadelphia.

Birdwhistell, R. L. (1970). *Kinesics and context*. University of Pennsylvania press, Philadelphia, USA.

Bray, T., Hollander, D., Layman, A., Tobin, R., and Thompson, H. S. (2009). Namespaces in XML 1.0 (Third edition). W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/2009/REC-xml-names-20091208/.

Bray, T., Paoli, J., and Sperberg-McQueen, C. (1998). Extensible markup language (XML) 1.0. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/1998/REC-xml-19980210/.

Brunet, P. M., McKeown, G., Cowie, R., Donnan, H., and Douglas-Cowie, E. (2009). Social signal processing: What are the relevant variables? and in what ways do they relate? In *Proceedings of the IEEE International Workshop on Social Signal Processing*, Amsterdam, The Netherlands.

Bullock, D. (2004). Adaptive neural models of queuing and timing in fluent action. *Trends in Cognitive Sciences*, 8(9):426–433. Available from: http://www.keck.ucsf.edu/~houde/sensorimotor_jc/DBullock04a.pdf.

Bullock, D. and Rhodes, B. J. (2003). Competitive queuing for planning and serial performance. In *The Handbook of Brain Theory and Neural Networks*, pages 241–244. MIT Press, second edition.

Burkhardt, F. and Schröder, M. (2008). Emotion markup language: Requirements with priorities. W3C Incubator Group Report, World Wide Web Consortium. Available from: http://www.w3.org/2005/Incubator/emotion/XGR-requirements-20080513/.

Burkhardt, F. and Sendlmeier, W. F. (2000). Verification of acoustical correlates of emotional speech using formant synthesis. In *Proceedings of the ISCA Workshop on Speech and Emotion*, pages 151–156, Northern Ireland.

Burkhardt, F., van Ballegooy, M., Englert, R., and Huber, R. (2005). An emotion-aware voice portal. In *Proc. Electronic Speech Signal Processing ESSP*, page 123–131, Prague, Czech Republic.

Burnett, D. C., Walker, M. R., and Hunt, A. (2004). Speech synthesis markup language (SSML) version 1.0. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/speech-synthesis/.

Cassell, J., Bickmore, T., Campbell, L., Vilhjálmsson, H., and Yan, H. (2000). Human conversation as a system framework: designing embodied conversational agents. In *Embodied conversational agents*, pages 29–63. MIT Press. Available from: http://portal.acm.org/citation.cfm?id=371555.

Cassell, J., Bickmore, T. W., Billinghurst, M., Campbell, L., Chang, K., Vilhjalmsson, H. H., and Yan, H. (1999). Embodiment in conversational interfaces: Rea. *CHI*, pages 520–527.

Cassell, J., Nakano, Y. I., Bickmore, T. W., Sidner, C. L., and Rich, C. (2001). Non-verbal cues for discourse structure. In *Proc. ACL*, page 114–123, Stroudsburg, PA, USA. Available from: http://dx.doi.org/10.3115/1073012.1073028.

Castellano, G., Bresin, R., Camurri, A., and Volpe, G. (2007). Expressive control of music and visual media by full-body movement. In *Proceedings of the 7th international conference on New interfaces for musical expression*, pages 390–391, New York. Available from: http://portal.acm.org/citation.cfm?id=1279740.1279829.

Chi, D., Costa, M., Zhao, L., and Badler, N. (2000). The EMOTE model for effort and shape. In *Proc. Computer graphics and interactive techniques*, SIGGRAPH '00, page 173–182, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. Available from: http://dx.doi.org/10.1145/344779.352172.

Chinnici, R., Moreau, J., Ryman, A., and Weerawarana, S. (2007). Web services description language (WSDL) version 2.0. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/wsdl20/.

Clavel, C., Vasilescu, I., Devillers, L., Richard, G., and Ehrette, T. (2008). Fear-type emotion recognition for future audio-based surveillance systems. *Speech Communication*, 50(6):487–503. Available from: http://portal.acm.org/citation.cfm?id=1377280.

CMLabs (2007). Psyclone. http://www.mindmakers.org/projects/Psyclone. Available from: http://www.mindmakers.org/projects/Psyclone.

Cornelius, R. R. (2000). Theoretical approaches to emotion. In *Proceedings of the ISCA Workshop on Speech and Emotion*, pages 3–10, Northern Ireland.

Cowie, R. (2010). Describing the forms of emotional colouring that pervade everyday life. In Goldie, P., editor, *The Oxford Handbook of Philosophy of Emotion*, pages 63–94. Oxford University Press. doi:10.1093/oxfordhb/9780199235018.003.0004.

Cowie, R. and Cornelius, R. R. (2003). Describing the emotional states that are expressed in speech. *Speech Communication*, 40(1-2):5–32.

Cowie, R., Douglas-Cowie, E., Appolloni, B., Taylor, J., Romano, A., and Fellenz, W. (1999). What a neural net needs to know about emotion words. In Mastorakis, N., editor, *Computational Intelligence and Applications*, pages 109–114. World Scientific & Engineering Society Press.

Cowie, R., Sussman, N., and Ben-Ze'ev, A. (2010). Emotion: Concepts and definitions. In Petta, P., Pelachaud, C., and Cowie, R., editors, *Emotion-Oriented Systems – The Humaine Handbook*, pages 9–30. Springer.

Damasio, A. R. (1994). *Descartes' Error: Emotion, Reason, and the Human Brain*. Grosset/Putnam, New York.

Dautenhahn, K. (1999). Robots as social actors: Aurora and the case of autism. In *Proc. CT99, The Third International Cognitive Technology Conference*, page 359–374, San Francisco.

de Carolis, B., Pelachaud, C., Poggi, I., and Steedman, M. (2004). APML, a markup language for believable behavior generation. In Prendinger, H. and Ishizuka, M., editors, *Life-Like Characters*, pages 65–85. Springer, New York.

Dibeklioglu, H., Kosunen, I., Hortas, M. O., Salah, A. A., and Zuzánek, P. (2010). An Affect-Responsive interactive photo frame. In *Proc. eNTERFACE'10*, Amsterdam, The Netherlands.

Dimakis, N., Soldatos, J. K., Polymenakos, L., Fleury, P., Curín, J., and Kleindienst, J. H. (2008). Integrated development of Context-Aware applications in smart spaces. *IEEE Pervasive Computing*, 7(4):71–79. Available from: http://portal.acm.org/citation.cfm?id=1477245.

Douglas-Cowie, E., Cowie, R., Sneddon, I., Cox, C., Lowry, O., McRorie, M., Martin, J., Devillers, L., Abrilian, S., Batliner, A., Amir, N., and Karpouzis, K. (2007). The HUMAINE database: Addressing the collection and annotation of naturalistic and induced emotional data. In *Proc. Affective Computing and Intelligent Interaction*, pages 488–500, Lisbon, Portugal. Available from: http://dx.doi.org/10.1007/978-3-540-74889-2_43.

Dunbar, R. I. M. (2003). The social brain: Mind, language, and society in evolutionary perspective. *Annual Review of Anthropology*, 32:163–181. Available from: http://www.jstor.org/stable/25064825.

Eagle, N. and Pentland, A. (2006). Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268. Available from: http://dx.doi.org/10.1007/s00779-005-0046-3.

Edlund, J. and Heldner, M. (2006). /nailon/-Software for online analysis of prosody. In *Proc. ICSLP*, Pittsburgh, PA, USA.

Edlund, J., Heldner, M., and Gustafson, J. (2005). Utterance segmentation and turn-taking in spoken dialogue systems. In Fisseni, B., Schmitz, H., Schröder, B., and Wagner, P., editors, *Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen*, pages 576–587. Peter Lang.

Ekman, P. (1972). Universals and cultural differences in facial expressions of emotion. In Cole, J., editor, *Nebraska Symposium on Motivation*, volume 19, pages 207–282. University of Nebraska Press. Available from: http://www.paulekman.com/wp-content/uploads/2009/02/Universals-And-Cultural-Differences-In-Facial-Expressions-Of.pdf.

Ekman, P. (2003). *The Face Revealed*. Weidenfeld & Nicolson, London.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR. Available from: http://portal.acm.org/citation.cfm?id=1088876.

Evans, J. (2008). Dual-processing accounts of reasoning, judgment, and social cognition. *Annual Review of Psychology*, 59:255–278. Available from: http://www.ncbi.nlm.nih.gov/pubmed/18154502.

Eyben, F., Wöllmer, M., Graves, A., Schuller, B., Douglas-Cowie, E., and Cowie, R. (2009). On-line emotion recognition in a 3-D activation-valence-time continuum using acoustic and linguistic cues. *Journal on Multimodal User Interfaces*, 3(1-2):7–19. Available from: http://dx.doi.org/10.1007/s12193-009-0032-6.

Ferrucci, D., Lally, A., Verspoor, K., and Nyberg, E. (2009). Unstructured information management architecture (UIMA) version 1.0. OASIS standard, Organization for the Advancement of Structured Information Standards. Available from: http://docs.oasis-open.org/uima/v1.0/os/uima-spec-os.html.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext transfer protocol – HTTP/1.1. IETF request for comment. Available from: http://tools.ietf.org/html/rfc2616.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine. Available from: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Fontaine, J. R., Scherer, K. R., Roesch, E. B., and Ellsworth, P. C. (2007). The world of emotions is not Two-Dimensional. *Psychological Science*, 18(12):1050–1057. Available from: doi:10.1111/j.1467-9280.2007.02024.x.

Frijda, N. H. (1986). *The Emotions*. Cambridge University Press, Cambridge, UK.

Gebhard, P. (2005). ALMA - a layered model of affect. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, Utrecht.

Gebhard, P., Kipp, M., Klesen, M., and Rist, T. (2003). Authoring scenes for adaptive, interactive performances. In *Proc. AAMAS*, page 725–732, New York, NY, USA.

Gebhard, P., Schröder, M., Charfuelan, M., Endres, C., Kipp, M., Pammi, S., Rumpler, M., and Türk, O. (2008). IDEAS4Games: building expressive virtual characters for computer games. In *Proc. IVA*, volume LNCS 5208, pages 426–440, Tokyo, Japan. Springer. Available from: http://dx.doi.org/10.1007/978-3-540-85483-8_43.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. S. (1994). *PVM: Parallel Virtual Machine: a users' guide and tutorial for network parallel computing*. MIT Press.

GigaSpaces (2009). GigaSpaces community contribution. Available from: http://www.gigaspaces.com/communitycontribution.

Goddeau, D., Brill, E., Glass, J. R., Pao, C., Phillips, M., Polifroni, J., Seneff, S., and Zue, V. W. (1994). Galaxy: A human-language interface to on-line travel information. In *Proc. ICSLP*, Yokohama, Japan.

Gratch, J. and Marsella, S. (2004). A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306.

Gratch, J., Wang, N., Gerten, J., Fast, E., and Duffy, R. (2007). Creating rapport with virtual agents. In *Proc. Intelligent Virtual Agents*, pages 125–138, Paris, France. Available from: http://dx.doi.org/10.1007/978-3-540-74997-4_12.

Gross, J. J. (2001). Emotion regulation in adulthood: Timing is everything. *Current Directions in Psychological Science*, 10(6):214–219. Available from: http://www.blackwell-synergy.com/doi/abs/10.1111/1467-8721.00152.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H., Karmarkar, A., and Lafon, Y. (2007). SOAP version 1.2 part 1: Messaging framework (Second edition). W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/soap12-part1/.

Gummadi, K. P., Saroiu, S., and Gribble, S. D. (2002). King: estimating latency between arbitrary internet end hosts. In *Proc. 2nd ACM SIGCOMM Workshop on Internet measurement*, IMW '02, page 5–18, New York, NY, USA.

Gunes, H. and Pantic, M. (2010a). Automatic, dimensional and continuous emotion recognition. *International Journal of Synthetic Emotion*, 1(1):68–99.

Gunes, H. and Pantic, M. (2010b). Dimensional emotion prediction from spontaneous head gestures for interaction with sensitive artificial listeners. In *Proc. Intelligent Virtual Agents*, page 371–377, Philadelphia, USA.

Gustafson, J., Lindberg, N., and Lundeberg, M. (1999). The August spoken dialogue system. In *Proc. Eurospeech*, Budapest, Hungary.

Gustavsson, C., Beard, S., Strindlund, L., Huynh, Q., Wiknertz, E., Marriot, A., and Stallo, J. (2001). VHML specification working draft v0.3. Available from: http://www.vhml.org/downloads/VHML/vhml.pdf.

Harding, C. (2006). Definition of SOA. Available from: http://opengroup.org/projects/soa/doc.tpl?gdid=10632.

Hartmann, B., Mancini, M., and Pelachaud, C. (2002). Formational parameters and adaptive prototype instantiation for MPEG-4 compliant gesture synthesis. In *Proc. Computer Animation*, Geneva, Switzerland. Available from: http://doi.ieeecomputersociety.org/10.1109/CA.2002.1017516.

Hartmann, B., Mancini, M., and Pelachaud, C. (2006). Implementing expressive gesture synthesis for embodied conversational agents. In Gibet, S., Courty, N., and Kamp, J., editors, *Gesture in Human-Computer Interaction and Simulation*, number 3881 in LNCS, pages 188–199. Springer. Available from: http://dx.doi.org/10.1007/11678816_22.

Hatfield, E., Cacioppo, J. T., and Rapson, R. L. (1994). *Emotional contagion*. Cambridge University Press.

Hawes, N., Wyatt, J. L., Sloman, A., Sridharan, M., Dearden, R., Jacobsson, H., and Kruijff, G. (2009). Architecture and representations. In Christensen, H. I., Sloman, A., Kruijff, G., and Wyatt, J., editors, *Cognitive Systems*, pages 53–95. published online at http://www.cognitivesystems.org/cosybook/.

Heckmann, D. (2006). *Ubiquitous User Modeling*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany.

Henning, M. (2006). The rise and fall of CORBA. *ACM Queue: Component Technologies*, 4(5):28–34. Available from: http://doi.acm.org/10.1145/1142031.1142044.

Henning, M. (2009). Choosing middleware: Why performance and scalability do (and do not) matter. White paper, ZeroC. Available from: http://www.zeroc.com/articles/IcePerformanceWhitePaper.pdf.

Herzog, G., Ndiaye, A., Merten, S., Kirchmann, H., Becker, T., and Poller, P. (2004). Large-Scale software integration for spoken language and multimodal dialog systems. *Natural Language Engineering*, 10(3-4):283–305. Available from: http://dx.doi.org/10.1017/S1351324904003444.

Heylen, D., Bevacqua, E., ter Maat, M., Pelachaud, C., and de Sevin, E. (2009). Updated demonstrator of the dialogue manager. Project Deliverable D4a, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D4a%20Updated%20demo%20of%20the%20Dialogue%20Manager.pdf.

Heylen, D., Kopp, S., Marsella, S., Pelachaud, C., and Vilhjalmsson, H. (2008). Why conversational agents do what they do? functional representations for generating conversational agent behavior. In *Proc. First Functional Markup Language Workshop*, Estoril, Portugal.

Heylen, D. and ter Maat, M. (2010). Final demonstrator of the dialogue manager. Project Deliverable D4b, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-3.0/D4b%20Final%20dialogue%20manager.pdf.

Hirstein, W. and Ramachandran, V. S. (1997). Capgras syndrome: a novel probe for understanding the neural representation of the identity and familiarity of persons. *Proceedings of the Royal Society B: Biological Sciences*, 264(1380):437–444.

Huang, H. H., Nishida, T., Cerekovic, A., Pandzic, I. S., and Nakano, Y. (2008). The design of a generic framework for integrating ECA components. In *Proc. AAMAS*, page 128–135, Estoril, Portugal.

Hyniewska, S., Niewiadomski, R., Mancini, M., and Pelachaud, C. (2010). Expression of affects in embodied conversational agents. In Scherer, K. R., Bänziger, T., and Roesch, E. B., editors, *A blueprint for Affective Computing*. Oxford University Press.

Höök, K., Holm, J., Tullgren, K., Sjölinder, M., Karlgren, J., and Persson, P. (1999). Spatial or narrative: A study of the Agneta & Frida system. In *Proc. Workshop on Affect in Interactions: Towards a New Generation of Interfaces*, Siena, Italy.

Inanoglu, Z. and Caneel, R. (2005). Emotive alert: HMM-based emotion detection in voicemail messages. In *Proc. Intelligent User Interfaces*, San Diego, CA, USA.

Ioannou, S. V., Raouzaiou, A. T., Tzouvaras, V. A., Mailis, T. P., Karpouzis, K. C., and Kollias, S. D. (2005). Emotion recognition through facial expression analysis based on a neurofuzzy network. *Neural Networks*, 18:423–435.

Isbister, K., Nakanishi, H., Ishida, T., and Nass, C. (2000). Helper agent: designing an assistant for human-human interaction in a virtual meeting space. In *Proc.*

SIGCHI conference on Human factors in computing systems, CHI '00, page 57–64, New York, NY, USA. ACM. ACM ID: 332407.

ISO (1998). ISO general purpose metric screw threads – general plan. ISO Standard ISO 261, International Standards Organisation. Available from: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=4165.

ISO (2006). Information technology – open document format for office applications (OpenDocument) v1.0. ISO Standard ISO/IEC 26300:2006, International Standards Organisation. Available from: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485.

Johnston, M., Baggia, P., Burnett, D. C., Carter, J., Dahl, D. A., McCobb, G., and Raggett, D. (2009). EMMA: Extensible MultiModal Annotation markup language. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/emma/.

Jonsdottir, G., Thórisson, K., and Nivel, E. (2008). Learning smooth, human-like turntaking in realtime dialogue. In *Proc. Intelligent Virtual Agents*, Tokyo, Japan. Available from: http://dx.doi.org/10.1007/978-3-540-85483-8_17.

Jordan, D. and Evdemon, J. (2007). Web services business process execution language. OASIS standard, Organization for the Advancement of Structured Information Standards. Available from: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Jurafsky, D., Martin, J. H., and Kehler, A. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. MIT Press.

Kempe, B., Pfleger, N., and Löckelt, M. (2005). Generating verbal and nonverbal utterances for virtual characters. In *Virtual Storytelling*, pages 73–76. Springer. Available from: http://dx.doi.org/10.1007/11590361_8.

Kipp, M. (2001). Anvil - a generic annotation tool for multimodal dialogue. In *Proc. Eurospeech*, pages 1367–1370, Aalborg, Denmark.

Kipp, M., Heloir, A., Schröder, M., and Gebhard, P. (2010). Realizing multimodal behavior. In *Proc. Intelligent Virtual Agents*, page 57–63, Philadelphia, USA.

Kipp, M., Neff, M., Kipp, K., and Albrecht, I. (2007). Towards natural gesture synthesis: Evaluating gesture units in a data-driven approach to gesture synthesis. In *Proc. Intelligent Virtual Agents*, page 15–28, Paris, France.

Kobsa, A. (2001). Generic user modeling systems. *User modeling and user-adapted interaction*, 11(1):49–63.

Kopp, S., Allwood, J., Grammer, K., Ahlsen, E., and Stocksmeier, T. (2008). Modeling embodied feedback with virtual humans. In Wachsmuth, I. and Knoblich, G., editors, *Modeling Communication with Robots and Virtual Humans*, number 4930 in LNAI, page 18–37. Springer.

Kopp, S., Jung, B., Lessmann, N., and Wachsmuth, I. (2003). Max - a multimodal assistant in virtual reality construction. *Künstliche Intelligenz*, 2003(4):11–17.

Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., Thórisson, K., and Vilhjálmsson, H. (2006). Towards a common framework for multimodal generation: The behavior markup language. In *Proc. Intelligent Virtual Agents*, pages 205–217, Marina del Rey, CA, USA. Available from: http://dx.doi.org/10.1007/11821830_17.

Kruijff, G. J., Lison, P., Benjamin, T., Jacobsson, H., and Hawes, N. (2007). Incremental, multi-level processing for comprehending situated dialogue in human-robot interaction. In *Proc. Language and Robots*, page 55–64.

Krämer, N., Hoffmann, L., and Kopp, S. (2010). Know your users! empirical results for tailoring an agent's nonverbal behavior to different user groups. In *Proc. Intelligent Virtual Agents*, page 468–474, Philadelphia, USA.

Lawson, J. L., Al-Akkad, A., Vanderdonckt, J., and Macq, B. (2009). An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, page 245–254, New York, NY, USA.

Le Hors, A., Le Hégaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S. (2004). Document object model (DOM) level 3 core specification. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/DOM-Level-3-Core/.

Lemon, O. and Pietquin, O. (2007). Machine learning for spoken dialogue systems. In *Proc. of Interspeech*.

Lockman, J. J. (2000). A Perception-Action perspective on tool use development. *Child Development*, 71(1):137–144. Available from: http://dx.doi.org/10.1111/1467-8624.00127.

Lohse, M. (2007). *Network Integrated Multimedia Middleware, Services, and Applications*. VDM Verlag.

Lohse, M., Winter, F., Repplinger, M., and Slusallek, P. (2008). Network-integrated multimedia middleware (NMM). In *Proceeding of the 16th ACM international conference on Multimedia*, pages 1081–1084, Vancouver, British Columbia, Canada. Available from: http://portal.acm.org/citation.cfm?id=1459359.1459576.

Luxand, I. (2010). Luxand - detect human faces and recognize facial features with luxand FaceSDK. http://www.luxand.com/facesdk/. Available from: http://www.luxand.com/facesdk/.

Maes, P. (1994). Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40. Available from: http://portal.acm.org/citation.cfm?id=176792.

Mancini, M. and Pelachaud, C. (2008). The FML-APML language. In *Proc. Workshop on Functional Markup Language at The Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, Estoril, Portugal.

Martin, D. L., Cheyer, A. J., and Moran, D. B. (1999). The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1):91–128.

McCrae, R. R. and John, O. P. (1992). An introduction to the five-factor model and its applications. *Journal of Personality*, 60:175–215.

Mehrabian, A. (1996). Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14(4):261–292. Available from: http://dx.doi.org/10.1007/BF02686918.

Mehrabian, A. and Ferris, S. R. (1967). Inference of attitudes from nonverbal communication in two channels. *Journal of Consulting Psychology*, 31(3):248–252. Available from: http://dx.doi.org/10.1037/h0024648.

Mick, D. G. and Fournier, S. (1998). Paradoxes of technology: Consumer cognizance, emotions, and coping strategies. *Journal of Consumer Research*, 25(2):123–143. Available from: http://dx.doi.org/10.1086/209531.

MIDI Manufacturers Association (1982). The complete MIDI 1.0 detailed specification. Technical report. Available from: http://www.midi.org/techspecs/midispec.php.

Mindmakers (2008). Behavior markup language (BML) wiki. Available from: http://wiki.mindmakers.org/projects:BML:main.

Morency, L., de Kok, I., and Gratch, J. (2010). A probabilistic multimodal approach for predicting listener backchannels. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(1):70–84. Available from: http://dx.doi.org/10.1007/s10458-009-9092-y.

Mystic Game Development (2010). EMotion FX. Available from: http://www.mysticgd.com/.

Nakano, Y. I., Reinstein, G., Stocky, T., and Cassell, J. (2003). Towards a model of face-to-face grounding. In *Proc. ACL*, page 553–561, Stroudsburg, PA, USA. Available from: http://dx.doi.org/10.3115/1075096.1075166.

Nass, C. and Moon, Y. (2000). Machines and mindlessness: Social responses to computers. *Journal of Social Issues*, 56(1):81–103. Available from: http://dx.doi.org/10.1111/0022-4537.00153.

Niewiadomski, R., Bevacqua, E., Mancini, M., and Pelachaud, C. (2009). Greta: an interactive expressive ECA system. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, page 1399–1400.

NIST (2008). NIST data flow system II. Available from: http://www.nist.gov/smartspace/sf_presentation.html.

Noma, T., Zhao, L., and Badler, N. I. (2002). Design of a virtual human presenter. *IEEE Computer Graphics and Applications*, 20(4):79–85.

Ortony, A., Clore, G. L., and Collins, A. (1988). *The Cognitive Structure of Emotion*. Cambridge University Press, Cambridge, UK.

Pantic, M., Eyben, F., Gunes, H., Heylen, D., Schuller, B., and Wöllmer, M. (2009a). Human conversational signals analyser. Project Deliverable D3a, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D3a%20Human%20conversational%20signals%20analyser.pdf.

Pantic, M., Eyben, F., Gunes, H., Schröder, M., Schuller, B., Valstar, M., and Wöllmer, M. (2010). User-profiled human behaviour interpreter. Project Deliverable D3c, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-3.0/D3c%20Human%20behaviour%20interpreter.pdf.

Pantic, M., Eyben, F., Gunes, H., Schuller, B., and Wöllmer, M. (2009b). Human affect analyser. Project Deliverable D3b, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D3b%20Human%20affect%20analyser.pdf.

Pantic, M. and Rothkrantz, L. J. M. (2000). Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445.

Pardo, D., Mencia, B. L., Trapote, A. H., and Hernandez, L. (2010). Non-verbal communication strategies to improve robustness in dialogue systems: a comparative study. *Journal on Multimodal User Interfaces*, 3(4):285–297. Available from: http://www.springerlink.com/content/k58267064t133350/.

Pelachaud, C., Bevacqua, E., McRorie, M., Pammi, S., Schröder, M., Sneddon, I., and de Sevin, E. (2009). SAL multimodal generation component with customised SAL characters and visual mimicking behaviour. Project Deliverable D5a, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D5a%20SAL%20multimodal%20generation%20component.pdf.

Pelachaud, C., Bevacqua, E., Pammi, S., Schröder, M., and de Sevin, E. (2010). SAL multimodal generation component optimised for real-time behaviour. Project Deliverable D5b, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-3.0/D5b%20Multimodal%20generation%20component.pdf.

Pelachaud, C., Carofiglio, V., De Carolis, B., de Rosis, F., and Poggi, I. (2002). Embodied contextual agent in information delivering application. In *Proc. AAMAS*, page 758–765, New York, NY, USA.

Peter, C. and Herbon, A. (2006). Emotion representation and physiology assignments in digital systems. *Interact. Comput.*, 18(2):139–170. Available from: http://portal.acm.org/citation.cfm?id=1220955.1221009.

Peters, C. (2005). Direction of attention perception for conversation initiation in virtual environments. In *Proc. Intelligent Virtual Agents*, page 215–228.

Poh, M. Z., Swenson, N. C., and Picard, R. W. (2010). A wearable sensor for unobtrusive, Long-Term assessment of electrodermal activity. *IEEE Transactions on Biomedical Engineering*, 57(5):1243–1252.

Porzel, R. and Baudis, M. (2004). The tao of CHI: towards effective human-computer interaction. In *Proc. HLT/NAACL*, Boston, MA, USA.

Premack, D. and Woodruff, G. (1978). Does the chimpanzee have a theory of mind? *Behavioral and Brain sciences*, 1(4):515–526.

Puckette, M. (2009). Pure data. Available from: `http://puredata.info/`.

Raggett, D., Hors, A. L., and Jacobs, I. (1999). HTML 4.01 specification. W3C Recommendation, World Wide Web Consortium. Available from: `http://www.w3.org/TR/html401/`.

Raouzaiou, A., Spyrou, E., Karpouzis, K., and Kollias, S. (2006). An intermediate expressions' generator system in the MPEG-4 framework. In Atzori, L., Giusto, D., Leonardi, R., and Pereira, F., editors, *Visual Content Processing and Representation*, number 3893 in LNCS, page 129–136. Springer.

Raux, A. (2008). *Flexible Turn-Taking for Spoken Dialog Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Reeves, B. and Nass, C. (1996). *The media equation: how people treat computers, television, and new media like real people and places*. Cambridge University Press New York, NY, USA.

Rehm, M., André, E., and Nischt, M. (2005). Let's come together—social navigation behaviors of virtual and real humans. *Intelligent Technologies for Interactive Entertainment*, page 124–133.

Reithinger, N., Gebhard, P., Löckelt, M., Ndiaye, A., Pfleger, N., and Klesen, M. (2006). VirtualHuman: dialogic and affective interaction with virtual characters. In *Proc. International Conference on Multimodal interfaces*, ICMI '06, page 51–58, New York, NY, USA.

Rich, E. (1979). User modeling via stereotypes. *Cognitive Science*, 3(4):329–354. Available from: `http://www.sciencedirect.com/science/article/B6W48-4FWF9GC-9/2/f924f793eb153d455893e8d39982ef45`.

Richardson, L. and Ruby, S. (2007). *RESTFul Web Services*. O'Reilly.

Sacks, H., Schegloff, E. A., and Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735.

Salovey, P. and Mayer, J. D. (1990). Emotional intelligence. *Imagination, cognition and personality*, 9(3):185–212.

Scherer, K. R. (1984). On the nature and function of emotion: A component process approach. In Scherer, K. R. and Ekman, P., editors, *Approaches to emotion*, pages 293–317. Erlbaum, Hillsdale, NJ.

Scherer, K. R. (1999). Appraisal theory. In Dalgleish, T. and Power, M. J., editors, *Handbook of Cognition & Emotion*, pages 637–663. John Wiley, New York.

Scherer, K. R. (2000). Psychological models of emotion. In Borod, J. C., editor, *The Neuropsychology of Emotion*, pages 137—162. Oxford University Press, New York.

Scherer, K. R. (2005). What are emotions? and how can they be measured? *Social Science Information*, 44(4):695–729. Available from: doi:10.1177/0539018405058216.

Schröder, M. (2008). Approaches to emotional expressivity in synthetic speech. In Izdebski, K., editor, *The Emotion in the Human Voice*, volume 3. Plural, San Diego, CA.

Schröder, M. (2009a). Expressive speech synthesis: Past, present, and possible futures. In Tao, J. and Tan, T., editors, *Affective Information Processing*, pages 111–126. Springer, London. Available from: http://dx.doi.org/10.1007/978-1-84800-306-4_7.

Schröder, M. (2009b). First full-scale SAL system. Project Deliverable D1c, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D1c%20First%20full-scale%20SAL%20system.pdf.

Schröder, M., Baggia, P., Burkhardt, F., Martin, J., Pelachaud, C., Peter, C., Schuller, B., Wilson, I., and Zovato, E. (2008a). Elements of an EmotionML 1.0. W3C Final Incubator Group Report, World Wide Web Consortium. Available from: http://www.w3.org/2005/Incubator/emotion/XGR-emotionml-20081120/.

Schröder, M., Baggia, P., Burkhardt, F., Oltramari, A., Pelachaud, C., Peter, C., and Zovato, E. (2010a). Emotion markup language (EmotionML) 1.0. W3C Working Draft, World Wide Web Consortium. Available from: http://www.w3.org/TR/2010/WD-emotionml-20100729/.

Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., and Zovato, E. (2009a). Emotion markup language (EmotionML) 1.0. W3C Working Draft, World Wide Web Consortium. Available from: http://www.w3.org/TR/2010/WD-emotionml-20100729/.

Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., and Zovato, E. (2011a). Emotion markup language (EmotionML) 1.0. W3C Last Call Working Draft, World Wide Web Consortium. Available from: http://www.w3.org/TR/2011/WD-emotionml-20110407/.

Schröder, M., Bevacqua, E., Eyben, F., Gunes, H., ter Maat, M., Pammi, S., de Sevin, E., Valstar, M., and Wöllmer, M. (2010b). Final SAL system. Project Deliverable D1d, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-3.0/D1d%20Final%20SAL%20system.pdf.

Schröder, M., Devillers, L., Karpouzis, K., Martin, J., Pelachaud, C., Peter, C., Pirker, H., Schuller, B., Tao, J., and Wilson, I. (2007a). What should a generic emotion markup language be able to represent? In *Proc. 2nd International Conference on Affective Computing and Intelligent Interaction (ACII'2007)*, Lisbon, Portugal.

Schröder, M., Pammi, S., and Türk, O. (2009b). Multilingual MARY TTS participation in the blizzard challenge 2009. In *Blizzard Challenge 2009*, Edinburgh, UK.

Schröder, M., Pelachaud, C., Ashimura, K., Baggia, P., Burkhardt, F., Oltramari, A., Peter, C., and Zovato, E. (2011b). Vocabularies for EmotionML. W3C working draft, World Wide Web Consortium. Available from: http://www.w3.org/TR/2011/WD-emotion-voc-20110407/.

Schröder, M., Pirker, H., and Lamolle, M. (2006). First suggestions for an emotion annotation and representation language. In *Proceedings of LREC'06 Workshop on Corpora for Research on Emotion and Affect*, pages 88–92, Genoa, Italy.

Schröder, M., Pirker, H., Lamolle, M., Burkhardt, F., Peter, C., and Zovato, E. (2011c). Representing emotions and related states in technological systems. In Petta, P., Cowie, R., and Pelachaud, C., editors, *Emotion-Oriented Systems – The Humaine Handbook*, pages 367–386. Springer.

Schröder, M., Wilson, I., Jarrold, W., Evans, D., Pelachaud, C., Zovato, E., and Karpouzis, K. (2008b). What is most important for an emotion markup language? In *Proc. Third Workshop Emotion and Computing, KI 2008*, Kaiserslautern, Germany.

Schröder, M., Zovato, E., Pirker, H., Peter, C., and Burkhardt, F. (2007b). W3C Emotion Incubator Group Final Report. W3C Incubator Group Report, World Wide Web Consortium. Available from: http://www.w3.org/2005/Incubator/emotion/XGR-emotion-20070710.

Schuller, B., Eyben, F., Gunes, H., Pantic, M., Schröder, M., Valstar, M., and Wöllmer, M. (2010). Final face and voice feature extraction component with incremental, (near) real-time processing. Project Deliverable D2b, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-3.0/D2b%20Final%20feature%20extraction.pdf.

Schuller, B., Eyben, F., Gunes, H., Pantic, M., Valstar, M., and Wöllmer, M. (2009). Improved face and voice feature extraction with speaker adaptation and learning. Project Deliverable D2a, SEMAINE. Available from: http://semaine.sourceforge.net/SEMAINE-2.0/D2a%20Improved%20face%20and%20voice%20feature%20extraction.pdf.

Schuller, B., Seppi, D., Batliner, A., Maier, A., and Steidl, S. (2007). Towards more reality in the recognition of emotional speech. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages IV–941–IV–944.

Schütz, T. (2008). Concept and design of the integration framework. Theseus Ordo deliverable report D11.1.1.b, empolis GmbH. Available from: http://www.eclipse.org/smila/docs/ORDO_D.11.1.1.b_ConceptIntegrationFramework_V1.0.pdf.

Shaver, P., Schwartz, J., Kirson, D., and O'Connor, C. (1987). Emotion knowledge: Further exploration of a prototype approach. *Journal of Personality and Social Psychology*, 52:1061–1086.

Sidner, C. L., Lee, C., Kidd, C. D., Lesh, N., and Rich, C. (2005). Explorations in engagement for humans and robots. *Artificial Intelligence*, 166(1-2):140–164.

Sonntag, D., Reithinger, N., Herzog, G., and Becker, T. (2010). A discourse and dialogue infrastructure for industrial dissemination. In Lee, G., Mariani, J., Minker, W., and Nakamura, S., editors, *Spoken Dialogue Systems for Ambient Environments*, number 6392 in LNCS, page 132–143. Springer. Available from: http://dx.doi.org/10.1007/978-3-642-16202-2_12.

Stocky, T. and Cassell, J. (2002). Shared reality: spatial intelligence in intuitive user interfaces. In *Proceedings of the 7th international conference on Intelligent user interfaces*, IUI '02, page 224–225, New York, NY, USA.

Sun Microsystems (2005). JavaSpaces service specification. Available from: http://java.sun.com/products/jini/2.1/doc/specs/html/js-spec.html.

ter Maat, M. and Heylen, D. (2009). Turn management or impression management? In *Proc. Intelligent Virtual Agents*, pages 467–473, Amsterdam, The Netherlands. Available from: http://dx.doi.org/10.1007/978-3-642-04380-2_51.

The OSGi Alliance (2009). OSGi service platform core specification. Technical Report Release 4, Version 4.2, OSGi Alliance. Available from: http://www.osgi.org/download/r4v42/r4.core.pdf.

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2004). XML schema part 1: Structures second edition. W3C Recommendation, World Wide Web Consortium. Available from: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

Thórisson, K. (1997). Gandalf: An embodied humanoid capable of Real-Time multimodal dialogue with people. In *Proc. First SCM International Conference on Autonomous Agents*, pages 536–537, Marina del Rey, CA, USA. Available from: http://xenia.media.mit.edu/%7Ekris/papers/ACM_AA97.html.

Thórisson, K. R. (2002). Natural Turn-Taking needs no manual: Computational theory and model, from perception to action. In Granström, B., House, D., and Karlsson, I., editors, *Multimodality in Language and Speech Systems*, pages 173–207. Kluwer Academic Publishers, Dordrecht.

Thórisson, K. R., Benko, H., Abramov, D., Arnold, A., Maskey, S., and Vaseekaran, A. (2004). Constructionist design methodology for interactive intelligences. *AI Magazine*, 25(4):77–90.

Traum, D. and Rickel, J. (2002). Embodied agents for multi-party dialogue in immersive virtual worlds. In *Proc. AAMAS*, pages 766–773, Bologna, Italy. ACM. Available from: http://portal.acm.org/citation.cfm?id=544922&dl=GUIDE,.

Troncy, R., Mannens, E., Pfeiffer, S., and van Deursen, D. (2010). Media fragments URI 1.0. W3C Last Call Working Draft, World Wide Web Consortium. Available from: http://www.w3.org/TR/1998/REC-xml-19980210/.

Tsapatsoulis, N., Raouzaiou, A., Kollias, S., Cowie, R., and Douglas-Cowie, E. (2002). Emotion recognition and synthesis based on MPEG-4 FAPs. In Pandzic, I. S. and Forchheimer, R., editors, *MPEG-4 Facial Animation - The standard, implementations, applications*. John Wiley & Sons, Hillsdale, NJ, USA.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

van Deemter, K., Krenn, B., Piwek, P., Klesen, M., Schröder, M., and Baumann, S. (2008). Fully generated scripted dialogue for embodied agents. *Artificial Intelligence*, 172(10):1219–1244.

von der Pütten, A., Krämer, N. C., and Gratch, J. (2009). Who's there? can a virtual agent really elicit social presence? In *Proc. 12th Annual International Workshop on Presence*, Los Angeles, CA, USA.

Wahlster, W. (2000). *Verbmobil: foundations of speech-to-speech translation*. Springer.

Wahlster, W. (2003). Smartkom: Symmetric multimodality in an adaptive and reusable dialogue shell. In *Proc. Human Computer Interaction Status Conference*, volume 3, page 47–62.

Wahlster, W. (2006). *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer.

Walker, M. A., Litman, D. J., Kamm, C. A., and Abella, A. (1997). PARADISE: a framework for evaluating spoken dialogue agents. In *Proc. EACL*, pages 271–280, Madrid, Spain. Available from: http://portal.acm.org/citation.cfm?id=979652&dl=.

Wallers, A., Edlund, J., and Skantze, G. (2006). The effect of prosodic features on the interpretation of synthesised backchannels. In *Proc. Perception and Interactive Technologies*, page 183–187, Kloster Irsee, Germany.

Ward, N. and Tsukahara, W. (2000). Prosodic features which cue back-channel responses in english and japanese. *Journal of Pragmatics*, 32(8):1177–1207. Available from: http://dx.doi.org/10.1016/S0378-2166(99)00109-5.

Wechsler, D. (1958). *The measurement and appraisal of adult intelligence*. Williams & Wilkins, Baltimore, MD, USA.

Wright, M. (2002). The open sound control 1.0 specification. Technical report, The Center For New Music and Audio Technology (CNMAT), UC Berkeley. Available from: http://opensoundcontrol.org/spec-1_0.

Yngve, V. H. (1970). On getting a word in edgewise. In *Chicago Linguistic Society. Papers from the 6th regional meeting*, volume 6, pages 567–577.

Zeng, Z., Pantic, M., Roisman, G. I., and Huang, T. S. (2007). A survey of affect recognition methods: audio, visual and spontaneous expressions. In *Proc. Multimodal Interfaces*, pages 126–133, Nagoya, Japan. Available from: http://portal.acm.org/citation.cfm?id=1322192.1322216.

Zue, V. W. and Glass, J. R. (2002). Conversational interfaces: Advances and challenges. *Proceedings of the IEEE*, 88(8):1166–1180.