A Freely Configurable, Multi-modal Sensor System for Affective Computing

Steffen Mader, Christian Peter, Roland Göcke, R. Schultz, J. Voskamp, B. Urban

Fraunhofer IGD Rostock, J.-Jungius-Str. 11, 18059 Rostock, Germany
{mader, cpeter, rgoecke, rschultz, voskamp, urban}@igd-r.fraunhofer.de

Affective computing systems consist of a varying number of components, such as sensors, data bases, and processing devices. So far, each system is set-up specifically and its architecture and configuration is as unique as its targeted purpose. With the increasing interest in affective computing, a generally applicable and easily realizable approach is desired to ease and speed up the installation process of such systems. In this paper, a Java based framework is presented which uses an adaptor-oriented approach in order to address this problem from a data perspective. An example is given for the application of this proposed framework within an affective computing system.

1 Introduction

Establishing an experimental system for acquisition and analysis of affect related data is a challenging task. Setting up several sensors, collecting data across diverse platforms and converting them into various formats demanded by different analysis tools requires a lot of resources, manpower and time. New scenarios require new sensor choices, different configurations of analysis tools and altered data flows.

In this paper, we present a framework that supports experimental set-ups for collection, analysis, and evaluation of emotions related data, which is easy to configure, to extend and to migrate to different environments. It allows the integration of arbitrary sensors, data sources and target applications.

In the following chapters, we first discuss an affective computing system from an integrative perspective and conclude requirements for an easily configurable and extendable sensor system. Following this, we shortly describe a concept for the realization of such an affective computing system as well as a sample implementation. Finally, a picture of our current work and a short outlook on the next steps is given.

2 Motivation and Goal

Affective computing systems consist of a number of highly dedicated components which, in most of the cases, were developed as independent stand-alone applications. Those components usually are designed around the task they are targeting. Decisions on the platform, the programming language to use, and ports and interfaces to be provided are made mainly with the targeted problem in focus. Other considerations, like networking capabilities or compatibility with systems or applications not directly linked with the particular task were laid aside or neglected at all.

Integrating such components into a complex system is a challenging task: the components provide a limited number of different interfaces; each component delivers or expects data in dedicated native formats; there is a huge variety in the data concerning their complexity, continuity, reliability, size, and transmission speed. Looking from the data perspective, a general approach to set-up such systems has to:

- present and describe all kinds of data in a uniform way, so that they can be handled by the system in a similar way;
- integrate the data of different sources by combining them in a sensible way;
- store and forward the data in a standardized way, preserving their time order.

As a solution, we propose a framework which can be considered as a system construction kit, where diverse components are wrapped into smoothly fitting building blocks that can be freely arranged and allow for a wide range of experiments. Our framework provides the infrastructure for connecting the components, an easy to use configuration mechanism, and unified, real-time capable data handling schemes. This makes the proposed framework also suitable as basis for modular, commercial solutions.

3 Concept

Existing components for affective computing systems often have been developed independently as stand-alone applications on different platforms. To allow them being smoothly integrated into the framework, they are extended by adaptors which provide the required uniformity in data representation. Moreover, the adaptors are designed to allow the handling of different components alike. In their entirety, these adaptors form an additional abstraction layer between the data sources and processing applications. This abstraction layer accomplishes three tasks:

1. Abstract the data from their representation, i.e. usage of an intermediate data representation for data transmission and manipulation in order to be independent from the actual implementation details of data sources and data sinks;
2. Abstract the data flow from its source and destination, i.e. build up complex data conversion and manipulation pipelines which can be re-used with different sensor configurations, storage- and analysis tools or end-user applications;
3. Abstract data sources from data destinations, i.e. the actual source of the data will be transparent to the data destination.

In order to support the above abstraction goals and the distribution of the data through the entire framework, the abstraction layer has to be able to support basic low-level operations on the data. These operations may also include recurring tasks like scaling, biasing, smoothing or re-sampling of the data. For additional flexibility, we require the framework to be configurable and extendable at runtime.

Behmaram-Mosavat and Encarnação [1] proposed a software framework for user-centered multi-modal interaction. Their framework provides multiple layers of abstraction which, besides device abstraction, are mainly focused on the support of high-level interaction schemes employing gestures, speech, and intuitive combinations of input modality. In contrast to their approach, we limit our framework to the transmission and distribution of quantitative data at a rather low semantic level. This simplifies the framework architecture and leaves the complexity of high-level processing inside the relevant components.

In 2001, Reitmayr and Schmalstieg [2] introduced the OpenTracker framework. This framework has been widely used in the fields of Virtual and Augmented Reality (AR/VR) since. Beside the abstraction of AR/VR applications from arbitrary position tracking devices, the OpenTracker framework also enables low-level pre-processing and fusion of data from different sources. However, OpenTracker is restricted to handle only a rigid data format describing position, orientation and button states of 3d-tracking devices, whereas our framework extends the OpenTracker concept for arbitrary and generic data formats.

Similar to OpenTracker, we choose an event-oriented approach, i.e. the data are handled in form of uniform *Events* and the components of the framework are classified according to their roles with respect to the data into *Sources*, *Filters* and *Sinks*. *Sources* are entry points of data into the framework. Each associated with a data producing entity (e.g. sensor), they use native drivers or open interfaces for configuration purposes and data handling. Data received from the sensor are converted into *Events* and handed over to the framework. Beside the actual data, each *Event* contains an individual timestamp which exactly describes the time the data entered the framework. *Sinks* are the exit points of the framework. They are each associated with a data processing entity (e.g. analysis tool) and implement an interface to it. Their purpose is to receive the *Events* from the framework, unwrap the contained data, convert them into the native format of the associated entity and to pass them to that entity. The interfacing of *Sources* and *Sinks* with their associated entities is done via native drivers, provided API's, existing communication interfaces (e.g. network, serial) or by direct integration for the case of self-developed entities. *Filters* perform supporting low-level operations on the transferred data, such as pre-filtering and masking operations, decomposition and fusion of data streams, statistical operations, etc.

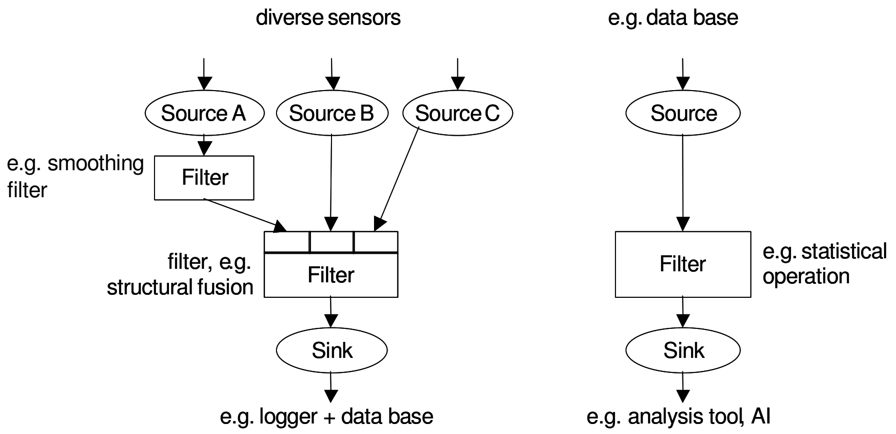

Fig. 1. Two examples of event processing configurations. **Left**: Session data from diverse sensors (e.g. blood pressure, gaze point, face mimic) are fused in order to record the session in a data base. **Right**: The previously recorded session is replayed, the data are processed with a meaningful statistical function in order to perform a run-time analysis

Sources, *Filters* and *Sinks* form the nodes of a data flow graph (see figure). The nodes are connected by directional edges which represent the routing of the data from their *Source* to the Sink. Along the graph, multiple filter nodes can be concatenated, and thus allow the construction of powerful filter chains out of simple basic filters. Passing those filter chains, the data will be manipulated according to the filters' functionality and parameters. The shape of the data flow graph is freely configurable out of the set of existing source, filter and sink modules.

Apart from specialized modules, which work on dedicated data formats, modules will be provided capable to work on generic data formats to perform general tasks, such as console monitoring, data base storage and retrieval, or network distribution. In order to allow this, all used data types have to be declared and named using a formal description. The description scheme is limited to a representative subset of basic data types and a simple structuring mechanism. Once described, the data formats may be used by *Sources*, *Filters* and *Sinks*.

4 Implementation

A prototypical system for affective user studies, which is under development at the Fraunhofer-Institute for Computer Graphics in Rostock, implements the described concept. It consists of the OmniRoute framework and several components for data acquisition, documentation and analysis. This includes the face tracking and mimic analysis platform Face☺me, a sensor board collecting ANS[1] related physiological parameters, and our RealEYES system for tracking of gaze- and mouse/keyboard data. All experiments can be documented with our logging tool EmoLogger. For the analysis of the recorded data sets we currently use data mining methods from the WEKA toolkit[3].

4.1 The OmniRoute Framework

Our OmniRoute framework has been implemented using the Java programming language. It has been developed on top of the Apache Avalon/Excalibur framework which provides an easy to use mechanism to extend the framework with new source, sink and filter modules, and to introduce new data formats. Therefore, sources, filters, sinks, and data formats have been implemented in form of Avalon Components. Based on the extension mechanisms of Avalon/Excalibur, new components and data formats are announced to the framework via XML-based description files and are immediately available without the need of re-building the framework itself.

In addition to dedicated source and sink modules, the framework provides a number of auxiliary modules such as a sink module providing formatted console output for monitoring purposes or source and sink modules for the transmission of arbitrary data through a network connection. The latter two allow the framework to span over multiple hosts, e.g. to set-up a distributed system integrating different hardware platforms.

[1] ANS – Autonomous Nervous System

Set-up and configuration of the data flow graph can be done on three ways: at compile time by hard-coding the graph into the application logic, at runtime by loading an XML-file containing the configuration, or, dynamically during runtime, by accessing the data flow graph directly. The syntax of the XML-configuration file and the construction rules for the graph roughly follow the concepts described in [1].

In order to allow the handling of generic data formats throughout the framework, we also use an XML description mechanism. For simplicity, the developed format description scheme is limited to a representative subset of basic data types and a simple structuring mechanism. These format descriptions form the basis for the construction of efficient data structures that will be instantiated during system initialization and may be used throughout the system at runtime.

4.2 Components

Data Sources

Our Face☺me system (formerly FAMIA [2],[5]) monitors and analyzes the facial mimics of the user. If a face is recognized, characteristic facial features are resolved and their position relative to each other constantly determined using template matching with normalized cross-correlation. The positional information about facial feature points (such as the eyebrows, eye corners, and the mouth) is sent out through a *FaceMeSource* module.

RealEYES [6] is our system for software usability tests. As one key feature, it synchronously records multiple channels of the user's interaction with the tested scenario. Beside mouse and keyboard events, it also records verbal utterances, facial expressions as well as the gaze position of the user and all activities on the computer screen. Mouse and keyboard data, as well as the actual gaze position are sent to the framework via the *RealEYESSource* module.

EmoBoard: For our studies on emotions in humans who are interacting with computer systems we developed a small, portable sensor board which collects physiological data of the user. Namely, the skin conductance level, the skin temperature, and the heart rate are measured. An FPGA[2] on the board collects the data from the sensors, evaluates them, and either sends them directly to a host computer or stores them in the on-board RAM to make it available to the host at a later time. An *EmoBoardSource* module running on the host computer converts the data into OmniRoute events and sends them out to the framework.

Data Sinks

The EmoLogger logging tool represents a universal data sink. It is a means to log and replay arbitrary data streams. For this purpose the associated *EmoLoggerSink* module can be configured to receive and store arbitrary data formats. Necessary or desirable meta-information, such as unit, scale, timestamp, the origin of the data, etc. can be obtained by the description of the incoming framework data type and by the framework configuration. Data and meta-information are stored within a mySQL database.

[2] FPGA – Field Programmable Gate Array; a programmable piece of hardware

5 Current Work and Outlook

In order to integrate the above mentioned components into the framework, first source and sink modules as well as the according data formats have been implemented. At a first stage, we are able to connect the sensor board as well as the gaze tracking platform with the logging tool. Several filters have been implemented providing structural operations on the data, such as extraction of data components and fusion of the used data formats. Also, the run-time configuration of the data flow graph using XML configuration files and the event processing based on a number of currently fixed data types have been implemented. This allows us, in a first step, to easily set-up and configure recording sessions for usability tests.

In a next step, the integration work will be continued by implementing further data source modules (e.g. the face mimic detector, speech input and -analysis) and a sink module for an AI analysis tool. This will enable us to set-up and configure variable experiments for the analysis of either pre-recorded or on-line sessions. A small repertory of respective filter modules will be implemented as they are needed.

In the near future, we will also add support for XML-configurable, generic data formats, which has not been realized yet. This will replace the currently used hard-coded data formats.

References

1. Behmaram-Mosavat, F.; Encarnação L. M.: A software framework for user-centered multi-modal interaction. CG topics, INI-GraphicsNet, Darmstadt, Germany, 12(4):5-6, 2000.
2. Reitmayr, G.; Schmalstieg, D.: An Open Software Architecture for Virtual Reality Interaction. In Proc. VRST'01, Banff, Canada, Nov. 15 - 17, 2001.
3. Witten, I. H. and Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco, 1999.
4. Luth, N.: Monitoring Human Faces from Multi-view Image Sequences. Lecture Notes In Computer Science, Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, pages: 173-184, Springer-Verlag, London, UK, 2002.
5. Göcke, R.: FAMIA - Facial Mimic Analysis. CG topics, INI-GraphicsNet, Darmstadt, Germany, 15(2):29-30, 2003.
6. Oertel, K.; Hein, O.; Elsner, A.: The RealEYES-Project: Usability Evaluation with Eye Tracking Data. Proceedings of the Human-Computer Interaction INTERACT '01, Tokyo, Japan, 2001.