HCI^2 Framework: A Software Framework for Multimodal Human-Computer Interaction Systems

Jie Shen and Maja Pantic

Abstract—**This paper presents a novel software framework for the development and research in the area of multimodal human-computer interaction (MHCI) systems. The proposed software framework, which is called the HCI^2 Framework, is built upon publish / subscribe (P/S) architecture. It implements a shared-memory-based data transport protocol for message delivery and a TCP-based system management protocol. The latter ensures that the integrity of system structure is maintained at runtime. With the inclusion of 'bridging modules', the HCI^2 Framework is interoperable with other software frameworks including Psyclone and ActiveMQ. In addition to the core communication middleware, we also present the integrated development environment (IDE) of the HCI^2 Framework. It provides a complete graphical environment to support every step in a typical MHCI system development process, including module development, debugging, packaging, and management, as well as the whole system management and testing. The quantitative evaluation indicates that our framework outperforms other similar tools in terms of average message latency and maximum data throughput under a typical single PC scenario. To demonstrate HCI^2 Framework's capabilities in integrating heterogeneous modules, we present several example modules working with a variety of hardware and software. We also present an example of a full system developed using the proposed HCI^2 Framework, which is called the CameGame system and represents a computer game based on hand-held marker(s) and low-cost camera(s).**

***Index Terms*—Publish / Subscribe Architecture, Multimodal Human-Computer Interface, Software Framework**

I. INTRODUCTION

ALONG with the rapid increase in computational power and network bandwidth during the past decades, the trend in the computing industry started to shift from PC-centred applications to services delivered through ubiquitous computing in a more human centred manner [1] [4] [7]. With this recent development, multimodal human-computer interfaces (MHCI) became an emerging research topic. Unlike traditional human-computer interfaces (e.g. based on keyboard, mouse, and so on), MHCI interact with users through natural modalities including facial expression, body gesture, verbal and non-verbal vocal cues [4] [7]. Arguably, MHCI not only simplify the use of computer systems, but also reduce user distraction and increase user satisfaction and productivity. Hence, MHCI are naturally seen as a (necessary) step towards future pervasive systems [1] [2].

Nevertheless, developing MHCI is not an easy task. The difficulty comes from two issues. Firstly, machine interpretation of human behaviour and multimodal human-computer dialogue modelling are closely related to context sensing [3] [4] and, in turn, to the famous hard AI problem - the frame problem of AI [6] (roughly, the frame problem is knowing which facts are relevant to the current reasoning problem and which facts are irrelevant). Secondly, because MHCI systems are normally constructed from a large number of highly interdependent and interwoven heterogeneous algorithmic units, the system integration is often cumbersome. While the former has been acknowledged and investigated by the research community, the latter is largely overlooked. More specifically, most published works use custom methods for system integration, resulting in application-specific and non-extendable systems. In this paper, we try to alleviate this problem by proposing a novel publicly-available software framework for development of MCHI systems being easily extendable, robust, and transparent.

A. Requirements for the Software Framework

An ideal MHCI system is expected to be, extendable, responsive, 'transparent', and robust [1] – [4], [7]. Hence, we propose a number of requirements for the software framework with which the MHCI system is to be developed, summarized into the following aspects.

1) Flexibility

Due to the complex nature of MHCI systems and the algorithms they utilize, flexibility on both system level and module level is essential.

On system level, any complex system structure should be supported. Because feature-level and model-level (rather than decision-level) multimodal-data-fusion-based approaches are receiving increasing attention [4] [5], algorithms dealing with different modalities are becoming more and more interdependent. With this trend, complex spatial and temporal

Manuscript received 19/04/2012. This work has been supported by the European Research Council under the ERC Starting Grant agreement no. ERC-2007-StG-203143 (MAHNOB).

Jie Shen and Maja Pantic are with Imperial College London, 180 Queen's Gate, London, SW7 2AZ, UK.

Maja Pantic is also with University of Twente, Twente, NL

module relationships within MHCI systems should be expected and therefore should be supported by the framework.

In addition, the framework should also support dynamic system structure reconfiguration. Considering that most algorithms only work well under very specific conditions, dynamic system structure reconfiguration would be an effective approach towards achieving an adaptive and robust performance at system level. For instance, consider a general facial feature point detector (FFPD) which works well for both frontal view and profile view faces. Complexity of such an algorithm is usually much higher than that of a specialized detector (e.g. two FFPDs optimized for frontal and profile images respectively which are activated / inactivated at runtime depending on the current face view).

On module level, because algorithms dealing with different modalities may vary drastically, the framework should not pose restrictions on the modules' internal structure. In other words, local / remote procedure call (LPC/RPC)-based approaches, which often require modules to be written in a predefined format based on specific call-back mechanisms, do not suffice.

2) Middleware Performance

Since audio and video signals, which are both high-bit-rate streams, are the primary information sources in most MHCI systems, the framework's underlying communication middleware should be able to efficiently deliver large amount of data.

Moreover, because MHCI systems are expected to react in real-time to users' (interactive) actions, a long time spent on message delivery is unwanted, especially in large systems where message latencies at each level of the processing cycle accumulates over time in which the MHCI system is used. This requires the communication middleware to facilitate message delivery with short latency.

In addition, to achieve high overall efficiency for developed MCHI systems, it is important for the framework to keep its resource consumption low and support compiled modules (modules written in languages such as C/C++).

3) Communication Reliability

Data loss may not have severe impact for systems having fixed structure where only data messages are transmitted between modules. However, for a system which may reconfigure its structure based on triggers, loosing such messages would result in significant performance loss. Therefore, the framework should guarantee successful message exchange or, at least, it should notify the sender if the delivery fails.

4) Error Tolerance

It is not uncommon for prototype modules to crash (or being terminated by a third-party debugger) at run-time due to BUGs or invalid / unexpected input. In such cases, the framework should be able to quarantine the error and keep other parts of the system (including the framework it self) unaffected.

5) Module Reusability

Reusability is crucial for rapid prototyping and testing. This is especially true for the front-end and low-level modules (e.g. video / audio capturers, face detectors, audio feature extractors, etc.), which are commonly a part of MHCI systems. To facilitate module reusability, the modules should be implemented as application / system – independent, preferably in (compiled and packaged) binary form.

6) Software Usability

A good user interface is essential for satisfactory user satisfaction. The software framework, despite the fact that its targeted users are software developers and researchers, is no exception from this rule. Instead of providing a set of loosely correlated utilities, the software framework should deliver an easy-to-use integrated graphical working environment enabling the developers to build, test, and maintain their systems easily.

B. An Overview of Existing Tools

There are a number of existing tools of the kind we describe here. These tools can be categorized into two types: software frameworks based on LPC/RPC [8] [11] [25], and message passing middleware based on publish / subscribe (P/S) architecture [9] [10] (and their extensions such as Fleeble [13] and SEMAINE API [26] [27]).

The LPC/RPC-based software frameworks generally have good performance in terms of data throughput, message latency, resource consumption level, and communication reliability. They also provide good support to the development of reusable modules and systems through GUI-enabled IDE. However, they usually lack flexibility (on both system and module levels) and are easily crashed by faulty modules.

Since the data transport in LPC/RPC-based approach is usually achieved by a direct-call function of the module's data export interface, the connection between modules often involves exchange of pointers. This scheme may result in tightly coupled systems. One limitation of this approach is it hardly supports feedback loops and dynamic system reconfiguration (i.e., modifying the system's internal structure at runtime). In addition, because all modules are required to follow a predefined call-back mechanism to facilitate data transport, the possible ways in which the modules can be implemented internally are limited. Last but not least, because tightly-coupled systems are prone to cascading failure, one faulty module may cause the entire system to break down.

In comparison, the existing message passing middleware usually support flexible spatial and temporal structure of the system and are more robust against module crashes. However, these tools often have poor performance in terms of data throughput, message latency, and resource consumption level. For example, both Psyclone and ActiveMQ are designed for the development of large scale distributed systems and internet applications [9] [10], hence their data-transport protocols are derived from TCP/IP, which is a suboptimal mean of inter-process communication (IPC). Although Fleeble [13] had a different design goal, it relies on Java Message Service (JMS), which is also based on TCP/IP and resulted in a similar performance penalty. The current version of SEMAINE API shares the same problem because it uses ActiveMQ as its underlying message passing middleware [26] [27].

In addition to the common performance problem, Psyclone and ActiveMQ also lack built-in support to the development of reusable modules. Although the P/S architecture naturally eliminates the dependency between modules [14], the modules are still dependent on their (usually hard-coded) local environment (i.e., the channels they subscribe and / or publish

TABLE I
AN OVERVIEW OF EXISTING TOOLS

Software Framework	Flexibility			Middleware Performance			Communication Reliability
	Restrictions to System Structure	Dynamic System Structure Reconfiguration	Restrictions to Module's Internal Structure	Data Throughput	Message Latency	Resource Consumption Level	
Microsoft DirectShow [8]	No feedback loops	Not supported	Must comply to predefined callback mechanism	> 800 MB/s	< 1ms	CPU usage < 1%	Guaranteed by design
Open-Interface [11]	No multicast	Not supported	Must comply to predefined callback mechanism	Could not be tested due to the lack of working examples in public domain.			Guaranteed by design
EyesWeb [25]	None	Not supported	Must comply to predefined callback mechanism	> 800 MB/s	< 1ms	CPU usage < 5%	Guaranteed by design
Psyclone AIOS [9]	None	Supported	None	< 140 MB/s	Up to 6900 ms	CPU usage < 80%	Message loss detected
ActiveMQ [10]	None	Supported	None	< 100 MB/s	Up to 650 ms	CPU usage < 50%	Timed-out messages may be discarded without notice
Fleeble [13]	None	Supported	None	Did not test because it does not support C++ (see subsection I.A.2).			Timed-out messages may be discarded without notice
SEMAINE API [26]	None	Supported	None	Same as ActiveMQ			
HCI^2 Framework	None	Supported	None	> 800 MB/s	< 1ms	CPU usage < 1%	Guaranteed by design

Software Framework	Error Tolerance	Module Reusability	Usability		
			GUI-Enabled IDE	Supported Languages	Note
Microsoft DirectShow [8]	The system will crash if any module crashes	Modules can be reused in other systems without modification	Provided	C++, C#, VB	Module development is relatively hard
Open-Interface [11]	The system will crash if any module crashes	Modules can be reused in other systems without modification	Provided	C++, Java, Matlab	Poor documentation
EyesWeb [25]	The system will crash if any module crashes	Modules can be reused in other systems without modification	Not provided	C++	
Psyclone AIOS [9]	Unaffected by module crashes	The source code may need to be changed if a module is to be reused	Not provided	C++, Java	BUGs including deadlock, access error and connection failure were detected
ActiveMQ [10]	Unaffected by module crashes	The source code may need to be changed if a module is to be reused	Not provided	C++, Java	Memory leak were detected
Fleeble [13]	Unaffected by module crashes	The source code may need to be changed if a module is to be reused	Limited (no graphical system structure representation)	Java	
SEMAINE API [26]	Unaffected by module crashes	Modules can be reused in other systems without modification	Provided	C++, Java	
HCI^2 Framework	Unaffected by module crashes	Modules can be reused in other systems without modification	Provided	C++	

a. All tests were conducted with Intel Core i5 CPU (4 cores) and 4 GB of memory.

to), hence, lack of reusability support. Moreover, neither Psyclone nor ActiveMQ featured a compact (visual) representation of the system structure and an easy mean for users to control the system at runtime.

A comprehensive overview of these tools is provided in Table I (note that the HCI^2 Framework is also included in the table for direct comparison). As shown in the table, none of them fulfils all of the aforementioned requirements.

C. Contributions

Our work proposes a new software framework, which is called the HCI^2 Framework ('HCI^2' stands for Human-Centred Intelligent Human-Computer Interaction), fulfilling all of the aforementioned requirements.

In order to meet the requirements regarding the system flexibility, data rate, latency, and reliability, we design a protocol for both runtime system management and data transport. The runtime system management protocol is proposed in compliance with the adopted Publish / Subscribe (P/S) architecture, which brings natural support to complex and dynamic systems [14]. To achieve reliable and efficient message delivery, we develop a data transport protocol based on shared-memory, which is shown to be the most efficient inter-process communication (IPC) method in terms of data throughput and average latency [12]. These protocols are implemented by the framework's core communication middleware.

We developed further a self-contained and easy-to-use integrated development environment (IDE) to facilitate the entire development cycle of MHCI systems. This tool, which is called the HCI^2 Framework IDE, facilitates module reusability and software usability. In particular, the HCI^2 Framework IDE embodies the following features:

1) Complete development support of highly flexible and reusable modules. To increase module reusability, we discriminate between the concepts of module class and module instance as explained in section II-D.
2) An easy-to-use centralized graphical user interface (GUI) facilitating module management, system configuration, module and system testing, and system redistribution.

D. *Organization of the Paper*

The rest of this paper is organized as follows. Section II discusses the middleware design of the HCI^2 Framework, including the P/S architecture, the runtime protocols, issues regarding robustness and interoperability. Section III the concepts of module class and module instance, and the centralize system management scheme. The implementation of this tool is described in section III. The evaluation of the HCI^2 Framework's performance is presented in section IV. Section V demonstrates the usage of HCI^2 Framework with several example modules and the CamGame demo system. Section VI concludes this paper.

II. CONCEPTUAL DESIGN

A. *Publish / Subscribe Architecture*

The core of the HCI^2 Framework consists of a middleware facilitating Publish / Subscribe (P/S) communication between modules at runtime. Fig. 1 illustrates an example system containing three modules built with the HCI^2 Framework. Each module is built as a standalone executable, which internally calls the module-side (communication) adapter of the framework to exchange messages with other modules. Different from local / remote procedure-call-based approaches, in which modules are implemented as components (DLLs, COM objects, and so on), and are called by the framework or other modules, modules in the HCI^2 Framework are granted explicit control over their own execution route. In other words, these modules do not have to follow any predefined internal structure model as long as they can correctly call the framework's module-side adapter whenever communication is needed. In this way, a high degree of flexibility at module level is achieved.

As illustrated in Fig. 1, modules do not send messages directly to each other, but do so via logical message dispatchers, which are called channels. Channels are named entities that allow a single message to be dispatched to any number of receivers which have previously shown 'interest' in receiving information from the channel in question [13] [14]. The mechanism behind is as follows. A module informs the framework if it is 'interested' in messages of a certain type by subscribing to the channel dedicated to that type of messages. Then, whenever a message is sent (published) to that channel, the message is automatically routed to all subscribers. With this

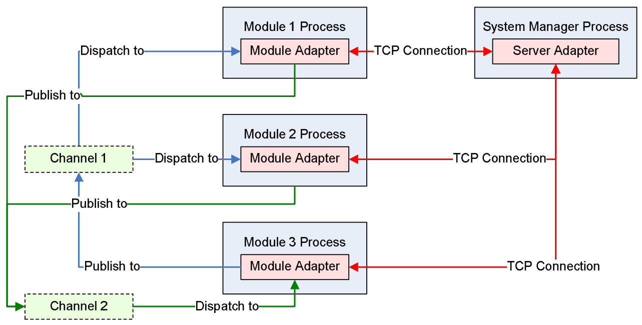


Fig. 1. Structure of an example system built with the HCI^2 Framework.

P/S mechanism, modules at both sending and receiving ends are effectively isolated, which means that their dependency on the presence of assumed upstream and / or downstream modules is eliminated. In other words, a module can be used in any circumstance as long as appropriate channels, which are always the same type of entities but with different names, exist. Therefore, development and using of context- and state-free modules become possible [14].

With this P/S architecture, the structure of the system is fully defined by the collection of channels and modules subscribed to those channels. This simple and intuitive representation of system configuration brings great flexibility since it does not impose any explicit restriction on the topology of the network of modules.

Dynamic system reconfiguration is facilitated by allowing modules to initiate and / or cancel subscriptions to channels at runtime. Execution of remaining modules is completely unaffected because each module only sees its input and output channels but not the upstream and downstream modules. Therefore, dynamic system reconfiguration is implicitly and naturally supported by the P/S architecture of the HCI^2 Framework.

All channels reside on a runtime system manager (server). It represents a central repository which stores all information regarding current system configuration at runtime, including a list of channels, a list of working modules, and their subscriptions. Although this central repository is not required in theory, it effectively represents the system configuration at runtime. In fact, the data transport protocol requires each module to carry a copy of a subset of this information. In order to maintain consistency between all these copies, TCP connection is established between every module and the system manager. A system management protocol is then used to synchronize each module's local copy of configuration information with the original copy stored in system manger whenever changes occur. More details on this issue are given below.

B. *Data Transport*

Recall that achieving high data rate, low latency, and reliable data transport, which was not met by tools like Psyclone and ActiveMQ, is one of the most important requirements of a MHCI-supportive software framework.

1) Choosing Inter-Process Communication Method

Because the data transport between modules in our framework is basically inter-process communication (IPC), it is important to choose a proper underlying IPC method in order to fulfil all aforementioned requirements. The following criteria are crucial when choosing an appropriate method.

1) The method should be general enough to support any number of concurrent communication sessions, with messages having arbitrary length.
2) The method should be reliable enough to guarantee ordered data delivery (first sent, first received).
3) The method should be efficient enough, which means it should have the potential to support high data rate and low latency communication. This also means that more fundamental methods would be favoured to avoid performance overhead.

Cross-platform support is another concern for the underlying IPC method. In general, it is desirable to use an IPC method available to most widely-used operating systems including Windows, Linux and OS X. This limits our choices to a subset of POSIX-compliant IPC methods including TCP socket, UDP socket, pipes, RPC and shared memory. Among these options, UDP is not reliable enough (since packet-dropping is allowed), while RPC and pipes can be inefficient (because they are often implemented in terms of shared-memory and / or TCP), only TCP and shared memory meet our requirements.

In practice, TCP is favoured by many exiting tools including Psyclone and ActiveMQ due to its convenience of use and its ability to connect multiple computers. However, our experimental comparison between TCP and shared memory, summarised in table II (results shown are obtain on Windows), shows that shared memory can support much higher (up to 10 times) data throughput than TCP under every CPU consumption constraint.

The major limitation of shared memory is it cannot be used to exchange data between different computers. Nonetheless, this disadvantage is considered acceptable for two reasons. Firstly, since our framework is primarily designed to facilitate single-computer applications, network communication is not a major concern for data transport. Secondly, in cases when message exchanging between sub-systems running on different computers becomes necessary, it would be more efficient to reuse other TCP-based tools to handle network-communication while still using the shared memory-based protocol for local data transport than to delivery all messages via TCP.

Hence, we choose shared memory as the underlying IPC method.

2) Data Transport Protocol

Unlike most IPC methods, shared memory is hardly a communication method. It simply allows developer to create named global memory block, which can be mapped into processes' address space in order to share data across process boundary [18] [28]. There is no automatic locking for the memory block for data corruption prevention, but the content of mapped buffer is guaranteed to remain consistent when it is accessed from different processes [18] [28].

TABLE II
COMPARISON BETWEEN TCP AND SHARED MEMORY

CPU Usage[a]	Data Rate of TCP	Data Rate of Shared Memory
5%	9 MB/s	90 MB/s
20%	34 MB/s	230 MB/s
35%	40 MB/s	245 MB/s
50%	55 MB/s	225 MB/s
65%	73 MB/s	245 MB/s
80%	80 MB/s	280 MB/s

a. Conducted on a ThinkPad T43 laptop with 2.0 GHz Pentium M CPU and 1 GB of memory.

Therefore, as the basis for the P/S communication, we define a protocol implementing a reliable peer-to-peer (PTP) communication through shared memory. We do so as follows.

1) Each peer allocates a named shared memory block, to be used as its local inbox. Within the block, the first 4 bytes in the address space are used to store an unsigned integer representing the total amount of data currently stored in the inbox. The remaining space is used to queue received data messages.
2) Each message consists of a variable-length content string, with a header section storing the message's type ID, sending time, and the sender's name. The messages are serialized into byte strings when they are written into the receiving peer's inbox.
3) In order to prevent data corruption caused by simultaneous multiple access, each peer creates a named mutex object to protect its shared memory block. The mutex object is used as the shared memory block's access-control token, which must be acquired by any reader / writer before it can access the shared memory block's content.
4) Each peer exploits a named event object as an indicator (flag) of whether there is any message pending in its local inbox. As a less resource-demanding alternative to busy waiting strategy, this event is used to ensure that every message will be retrieved as soon as it is pushed into the inbox.
5) When a message is sent, the sender obtains first the target inbox's access permission by acquiring its access-control token (i.e., the aforementioned mutex object). Then, it pushes the serialized data message into the receiving peer's message queue, modifies the shared memory block's leading bytes to reflect the new data length, sets the inbox's flag event to indicate that there is a message waiting and finally releases the token.
6) Each peer uses a thread to constantly monitor its flag-event's state. Whenever a 'message waiting' state is detected, the thread parses the local inbox's content (through protected reading operations), retrieves and splits the buffered data into separate messages, and then resets the flag-event. The split messages are then delivered to another buffer residing in the receiving peer's private memory space. Because shared memory facilitates secured synchronous data exchange, the procedure contains neither resynchronization step nor data validation step.

In addition to message exchange, the PTP communication scheme also features a built-in flow control mechanism. Since most algorithms used in a typical MHCI system are rather time-consuming, it is not unusual for a module to lack sufficient capacity to process all incoming data in time. In such case, the unprocessed messages will be queued in the module's internal

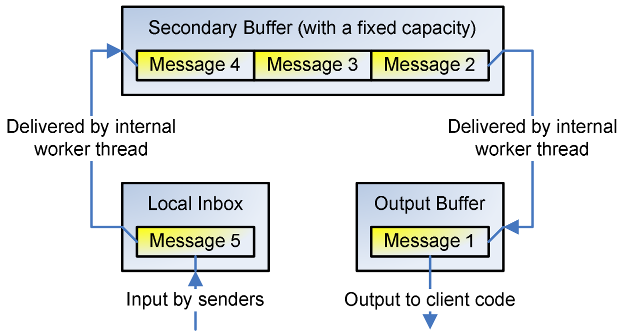

Secondary Buffer (with a fixed capacity)
Message 4
Message 3
Message 2
Delivered by internal worker thread
Delivered by internal worker thread
Local Inbox
Message 5
Output Buffer
Message 1
Input by senders
Output to client code

Fig. 2. The internal buffer layout of a receiving peer.

buffer. This waste of memory is not only unnecessary but also harmful to the system's stability. To solve this problem, we insert a secondary buffer between each peer's local inbox and its output buffer, as illustrated in Fig. 2. The secondary buffer is organized into an automatically growing queue with a fixed maximum capacity. If the module's message retrieval rate cannot catch up with the input rate, the secondary buffer will be eventually piled full and consequently block new income of messages. Hence, the message source will be forced to wait before subsequent messages can be successfully sent. In this way, the message rate in each processing pipeline will converge to an optimal level over time.

We extend this protocol to P/S communication as a collection of individual PTP message sending sessions. Let us explain this in more detail. In the HCI^2 Framework, channel only exists as a logical concept. It is actually a collection of subscriptions used to guide message routing. In practice, the runtime system manager stores these collections of subscriptions. Whenever a module needs to publish a message, it simply retrieves the list of the target channel's subscribers and then sends the message directly to every subscriber through the PTP protocol described above. In this way, message publishing is reduced to a number of PTP message sending sessions, where the reliability is guaranteed. To further improve communication reliability, the sequence of sending operations is bound into a single transaction. Namely, it is guaranteed that a published message would be received by either all of the subscribers (if successful) or none of them (otherwise). To prevent potential deadlock caused by overlapping publishing attempts, a timeout is added to each request.

C. *Runtime System Management Protocol*

In order to support dynamic system reconfiguration and maintain consistency between the configuration information stored in the system manager and copies of that held by the modules, a runtime system management protocol is used. This protocol utilizes TCP for its ease of use. Given that the messages used for these purposes are normally short, choosing TCP, which is shown to be suboptimal in terms of data throughput (see Table II), should not lead to significant performance penalty.

As explained above, each module maintains a TCP connection to the system manager during its entire life cycle. This connection is used by the runtime system management protocol for message exchanging. The messages are called runtime-system-management-messages and are transparent to module developers. Three types of runtime system management messages are defined:

1) Request messages include: module registration request, remote channel creation request, remote channel destruction request, module subscription request, and subscription cancellation request. These messages are sent from a module to the system manager when the module requests a change in the system structure. The system manager is then required to answer every request with an acknowledgement message. Note that module logoff request is unnecessary because shutting down the TCP connection carries the same information.
2) Notification messages include: channel creation notification, channel destruction notification, module subscription notification, and subscription cancellation notification. These messages are sent from the system manager to all modules in order to indicate changes in the system structure. Upon receiving, a module should update its local copy of the channel list to reflect the new configuration. Note that there is no module registration notification and module logoff notification as such; the system manager sends out appropriate subscription cancelation notification whenever a module logs off.
3) Acknowledgement messages include: ACK (approved) and NACK (rejected) sent by the system manager as the answer to a module's request.

Based on the runtime system management messages, protocol operations are defined including module registration, channel creation, module subscription. These operations have the form of a standard request-process-acknowledge procedure, and a further description of these is therefore unnecessary in our opinion.

D. *Robustness against Faulty Modules*

It is not uncommon for a prototype module to crash (or being terminated by a third-party debugger) at run-time due to software BUGs. Assuming the aforementioned protocols are correctly implemented by all modules, our design guarantees the execution of other modules will not be affected by the crashed one.

When a module crashes, its TCP connection to the runtime system manager would be lost. Upon detecting such event, the runtime system manager would initiate the log-off procedure on behalf of the crashed module and send notification messages to other modules in the same way as if the crashed module was logging off gracefully.

If a module crashes in the middle of a message publishing operation, the termination of the publishing thread would trigger the automatic release of all mutex objects it holds [23], hence transferring the access permission of the shared memory blocks to other modules. Therefore, the potential deadlock caused by a crashed module is eliminated. Furthermore, to prevent the shared memory blocks from being corrupted by half-written messages produced by crashed modules, we exploit the fact that writing to a well-aligned 32-bit memory chunk takes only one machine instruction and is hence atomic [24]. Therefore, as long as we do not write intermediate value to the data length indicator residing in the first 4 bytes (which are guaranteed to be well-aligned) of the shared memory block, all half-written messages would be automatically discarded or overwritten by subsequent accesses to the buffer.

Last but not least, since each module executes in its own virtual address space, there is no danger of private-memory corruption when another module crashes.

E. Interoperability with Other Message Passing Middleware

One limitation of the shared-memory-based data transport protocol is all communicating modules must reside on the same computer. However, this limitation can be overcome by interoperating with existing TCP/IP based message passing middleware.

Another advantage of interoperating with existing middleware is the users of the HCI^2 Framework may conveniently reuse the modules and / or subsystems developed using these tools in their new system built upon the HCI^2 Framework.

Because the HCI^2 Framework does not impose any restrictions to the behaviour of the modules as long as they are capable of communicating with other modules through the aforementioned protocols, special 'bridging modules' may be developed by implementing the protocols defined by both the HCI^ 2 Framework and the existing middleware to be interoperated with. As an example, a pair of modules (a sender and a receiver) are developed to facilitate message exchanging between the HCI^2 Framework and ActiveMQ. In particular, the sender module subscribes to an input channel in the HCI^2 Framework, translates all received messages to ActiveMQ format, and publishes the messages to a specified ActiveMQ topic, while the receiver module relays the messages along the opposite direction.

With the inclusion of these 'bridging modules', new systems developed using the HCI^2 Framework may delegate a part of its functionality to an old system build upon an existing message passing middleware, or simply take advantage of the middleware's communication capability to pass messages between multiple subsystems running on different computers.

F. Module Class and Module Instance

Although the P/S architecture eliminates dependencies between the modules and improves flexibility of the developed system, it does not necessarily lead to reusable modules. The key issue here is that of the structural dependency. Consider the following case. Suppose that both a face detection module and an object tracking module take input from a single video stream related to a 'Video_In' channel. In this case, if the face detector runs on greyscale images while the object tracker requires colour information, their requirement to the 'Video_In' channel's message format will be different. Consequently, these two modules will not be able to co-exist in the same system without being modified. The hard-coded channel ID in a module's implementation generates unnecessary dependency to its local environment, hence limits its reusability.

A module developed using Psyclone [9], ActiveMQ [10], or Fleeble [13] may contain even more hard-coded information including module ID, algorithm parameters (e.g. classifier coefficients), input / output message specification (e.g. video resolution) and so on. As a remedy to this problem, we refine the original concept of module and introduce two terms, the module class and the module instance. The basic idea here is to distinguish between task-specific requirements and generic specifications of a module. This concept is lent from object-oriented programming paradigm, hence the similar terminology.

1) Module Class

A module class consists of a generic implementation of a function unit. It usually contains a group of files including the algorithm implementation, a help document, instance-invariant data / binary files, and similar. Hence, a module class can be called a module package. In addition, the module class provides a template specifying which task-dependent information it needs to create an instance for specific use.

To store module class information including the instantiation template into what we call a module description file, a standard XML semantics is defined. In particular, each module description file encapsulates the following.

1) Content file list, which specifies the files included in the module package. We define three types of files: the algorithm (also known as the module programme), the help document, and the miscellaneous dependencies (e.g. DLLs, data files, etc.). The module programme is executed each time a module class' instance is activated. All other files are optional.
2) Input / output (I/O) specification, which defines the input / output channels to which the module may subscribe and to which it may publish. Instead of using a fixed ID, each channel is identified by its local (within the scope of the module class only) alias. During instantiation, the channel aliases will be mapped to the IDs of the channels that are actually used in the particular system. By using this channel mapping mechanism, the modules no longer display structural dependency on their local environment, and may be reused in different systems.
3) Parameter template specifies the rest of the task-dependent information. In terms of parameters, each parameter is defined by its name, type, default value, and range.

Note that in this paper we use the term module as an abbreviation of module class, unless the discussion is about a particular system, when 'module' refers to a module instance.

2) Module Instance

In addition to the generic data and operations provided by its module class, each module instance also contains task-specific information. This information is stored in the corresponding module configuration file. Specifically, a module configuration file contains the following fields:

1) Registration information including the module instance's ID, inbox size, and secondary buffer capacity.
2) Channel mappings, which specify the correspondence between the channel aliases and the actual channel IDs used by the particular system.
3) Value of the parameters defined in the module class specification.

With an appropriate module configuration file, a module may be easily reused in any system without modification.

G. Centralized System Management

While the distributed nature of the P/S architecture brings flexibility, as a side effect, it may also lead to potentially counter-intuitive and cumbersome system integration and testing process. Specifically, if all modules within the system need to be executed as standalone applications (as is the case in Psyclone [9] and ActiveMQ [10]), the system would be hard to build, configure, test and redistribute. To be able to conveniently construct a system from existing module classes, an explicit (visual) system representation is vital. This is achieved in the HCI^2 Framework by means of the module warehouse and the system configuration files.

The module warehouse serves as a global storage of the module classes to be used as the system building blocks. The module warehouse maintains a content list in the form of a XML file. This simple structure allows a module class to be easily imported and / or exported, which hardly requires more operations than copying a folder.

The system configuration files are used to represent the actual systems. Each such file specifies the channels, the module instances, and the P/S relations between them. Similar to the module description files and the module configuration files, we define a standard XML semantics for the system configuration files as well. In essence, a system configuration file consists of concatenated module configuration files specifying the system's constituent module instances and a list of the channels it uses. Since each system configuration file serves as a self-contained repository of all information defining a given system, the procedure for system reconfiguration and redistribution are reduced to mere file operations.

By using the module warehouse and the system configuration files, building a GUI-enabled IDE becomes a rather easy task. An intuitive visual display (in the form of a dynamic block diagram) of a given system' structure can be easily produced based on the relevant system configuration file. Activating a system or a part of it is also straightforward. The IDE merely needs to split the system configuration file into module configuration files, pass them as command-line parameters, and execute the appropriate module programmes stored in the module warehouse.

III. HCI^2 Framework Implementation

The HCI^2 Framework is implemented as a self-contained open-source software development tool (currently build for Windows only). The overall architecture of the HCI^2 Framework is illustrated in Fig. 3. The software package is divided into two major parts: the HCI^2 Framework SDK and the HCI^2 Framework IDE.

The HCI^2 Framework SDK comes as a set of libraries implementing the protocols described in section II. The HCI^2 Framework IDE is divided into three parts: the module packaging tools for module debugging and packaging, the system construction workbench for system integration and

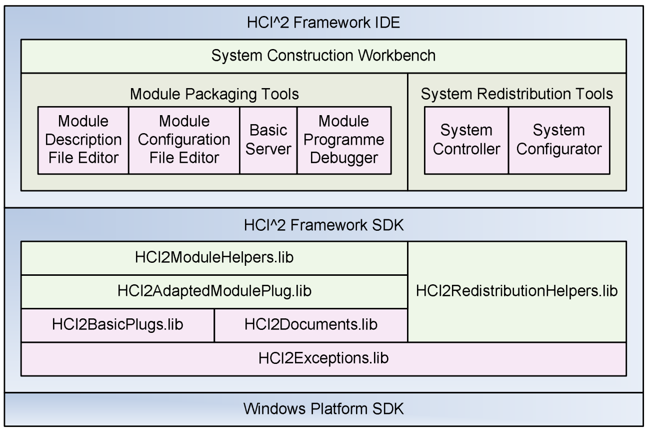


Fig. 3. Architecture of the HCI^2 Framework.

testing, and the system redistribution tools for exporting the developed systems as self-contained readily-deployable application packages.

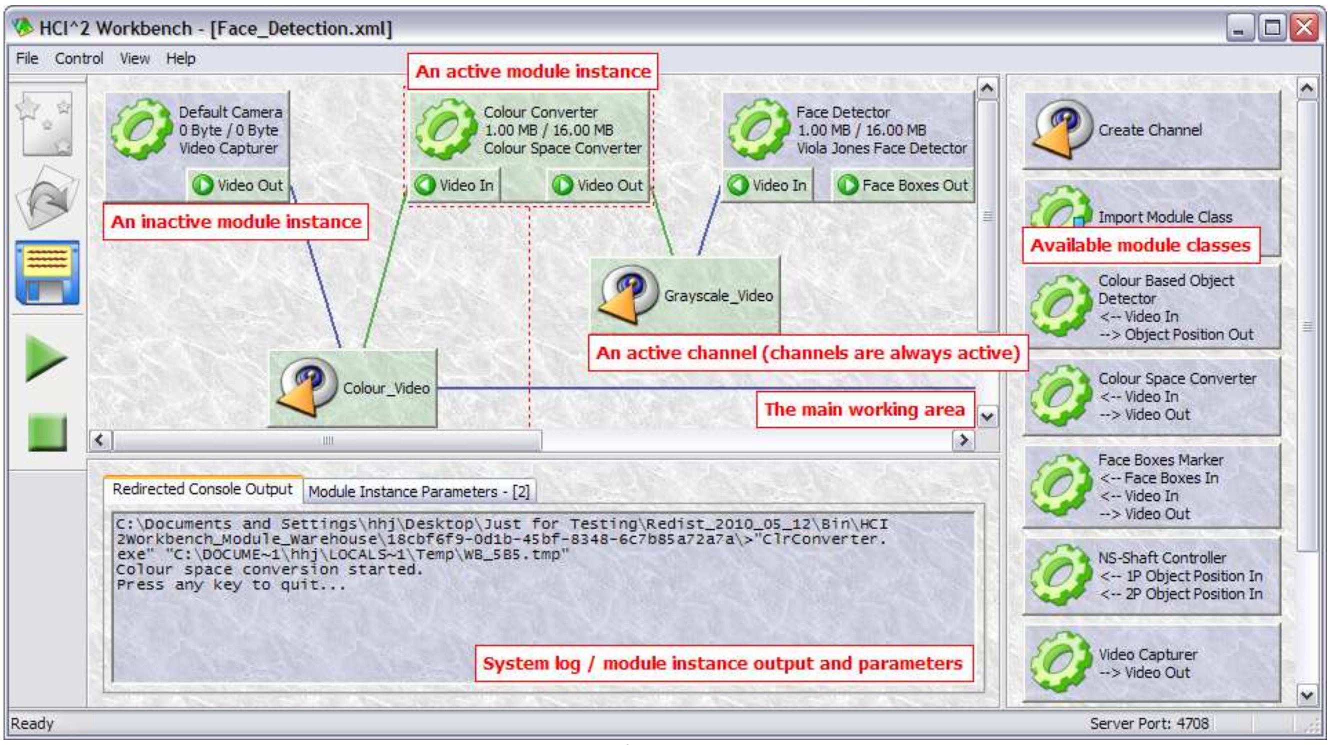
As shown in Fig. 3, the entire HCI^2 Framework is directly built upon Windows Platform SDK (and C++ Standard Template Library) with no additional dependency on any other software. Moreover, the HCI^2 Framework is delivered in both source-code and binary form.[1] Both features greatly simplify the installation of the framework and the redistribution procedure of the systems developed using it.

A. HCI^2 Framework SDK

The HCI^2 Framework SDK is a set of libraries facilitating module development. It includes several C++ classes implementing all protocols described in Section II. These classes are categorized into the following six groups.

1) Exception hierarchy used to represent various runtime errors. In addition to textual description, these classes also implement call stack tracing and cross-thread exception handling (i.e., allowing an exception thrown from one thread to be handled by another thread) to provide developers with more accurate information about the nature of the error.
2) Basic communication adapters implementing the data transport protocol and the runtime system management protocol.
3) File handler classes for parsing and composing module description files, module configuration files, and system configuration files.
4) Integrated plug-in component to further simplify the development of HCI^2 Framework IDE-compatible modules by integrating the module-side communication adapter and the module configuration file parser into a single component.
5) Module development helper classes providing additional functionalities including enhanced flow control and data synchronisation between multiple input streams.
6) System redistribution helper classes facilitating module warehouse management, file system object management, and console output redirection. These classes are useful for developing task-specific system controller / configuration utilities.

[1] Available at: http://ibug.doc.ic.ac.uk/resources/hci2-framework/



```
C:\Documents and Settings\hhj\Desktop\Just for Testing\Redist_2010_05_12\Bin\HCI
2Workbench_Module_Warehouse\18cbf6f9-0d1b-45bf-8348-6c7b85a72a7a\>"ClrConverter.
exe" "C:\DOCUME~1\hhj\LOCALS~1\Temp\WB_5B5.tmp"
Colour space conversion started.
Press any key to quit...
```

Fig. 4. The main window of the system construction workbench.

Note that most classes included in the HCI^2 Framework SDK follow the design pattern of interface / implementation separation. This allows us to eliminate references to OS-specific datatypes from the SDK's exported classes, thus resulting in a platform-independent interface. Consequently, all module programmes developed using the HCI^2 Framework SDK are naturally platform-independent on source-code level. Therefore, these modules may be easily migrated to another operating system as soon as the framework is re-implemented on that platform.

B. *HCI^2 Framework IDE*

The HCI^2 Framework IDE is a GUI-enabled, easy-to-use integrated development environment for module development and system integration. The IDE is divided into the module packaging tools, the system construction workbench, and the system redistribution tools.

1) *Module Packaging Tools*

Since every module class is fully specified by its module description file, creating such a file is the only step of module packaging. Hence, the central component of the module packaging tools is the module description file editor, which enables users to create and edit module description files using an easy-to-understand GUI. Other programmes included in the module packaging tools are provided to support early-stage module debugging, that is, testing the module programme before importing it into the module warehouse. Specifically, the module configuration file editor enables users to create example module configuration file for the module programme under testing, and the module programme debugger facilitates console output redirection and enables the module programme to be tested in a simulated environment.

2) *System Construction Workbench*

The system construction workbench provides an intuitive GUI for system integration, testing, and redistribution. Its main window (see Fig. 4) is split into three areas: the system editing area (top-left), the status area (bottom-left), and the module class list (right).

The system editing area displays the visual representation of the system under construction and enables users to easily modify its structure (e.g., adding / removing module instances / channels, establishing / cancelling channel mappings, etc.) using mouse clicks. As shown in Fig. 4, every module instance is represented by a button, which can be pressed to activate or deactivate it. Toolbar buttons are also provided to enable users to activate or deactivate the whole system using a single mouse click. The activation state of the module is indicated by the button's colour (green means active, while purple means inactive). Subscriptions are represented by the lines linking modules to the channels they are subscribed to.

The status area displays the details of the currently selected module instance, including the module's redirected console output and its parameters. If no module instance is selected, a log of system events will be presented in this area.

The module class list shows all available module classes stored in the module warehouse. Each module class is represented by a button with a popup menu facilitating module instantiation, removal, exporting, and updating. Module updating is particularly useful for module testing. Since the system construction workbench allows a module class to be updated without affecting the current system structure and the activation state of the module instances, users may efficiently fine-tune a module class without being required to reconfigure or restart the whole system before testing each revision. All accesses to the module warehouse are synchronized. Hence, there is no danger of data corruption when two or more instances of the system construction workbench are running at the same time.

The system construction workbench enables developers to save / load the current system structure into / from a system configuration file. This file can be copied to another machine for system redistribution.

3) System Redistribution Tools

To deploy a system on a machine which does not have the HCI^2 Framework installed, the following items must be included in the redistribution package.

1) The system configuration file.
2) All module classes used by the system.
3) A system controller for parsing the system configuration file and managing the system at runtime.
4) A system configuration utility enabling users to adjust the system's parameters.

The system configuration file and the module classes can be produced by the system construction workbench, while the system redistribution tools provide a generic implementation of the other two items. Since the programmes included in the system redistribution tools do not rely on any task-specific knowledge, they can be used in all system redistribution packages.

IV. Middleware Performance Evaluation

This section provides a quantitative comparison between the HCI^2 Framework, Psyclone, and ActiveMQ in terms of maximum data throughput and average message latency.

The experiment was conducted using a mock-up system running on a single computer. The system contains several data sources and data sinks. A variety of data sources were used during the experiment, each outputting at a specific data-rate to simulate a typical type of audio or video stream. The data rate we considered include 8 KB/s (8 kHz / 8-bit / 1 channel audio), 187.50 KB/s (48 kHz / 16 bit / 2 channel audio), 7.25 MB/s (352x288 / 3 channel / 25 FPS video), 26.37 MB/s (640x480 / 3 channel / 30FPS video) and many other values within the range between 8KB/s and 26.37 MB/s. In order to simulate the behaviour of real-time audio and / or video sources, the data sources were allowed to drop messages. We then added a number of data sinks into the systems to simulate data processing modules. By changing the number of these modules and the data sources' output data rate, we were able to adjust the overall communication workload in the mock-up system. We then measured the actual data rate (which can be lower than the overall source data rate due to message dropping) and average message latency in the mock-up system under different levels of communication workload. The test was repeated five times, each time for a different type of dispatcher (HCI^2 Framework channel, ActiveMQ topic / queue, or Psyclone whiteboard / stream) provided by the three tested frameworks.

Results of this experiment (conducted on a Dell Inspiron N5010 laptop with Intel Core i5 M430 CPU @ 2.27 GHz and 4 GB of memory) are shown in Fig. 5 - Fig. 7. Fig. 5 indicates that both ActiveMQ and Psyclone dispatchers start to cause the data sources to drop messages when the overall source data rate is above 102.4 MB/s. This is because these dispatchers are no

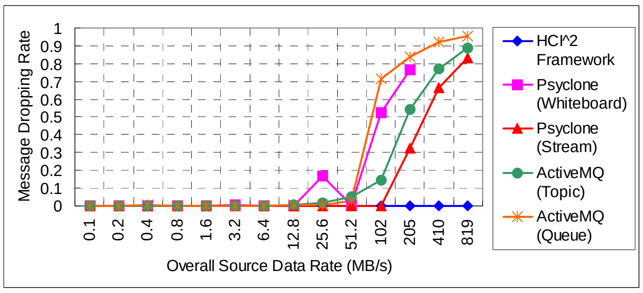


Fig. 5. Message dropping rate of the data sources in HCI^2 Framework, Psyclone, and ActiveMQ at different levels of overall source data rate.

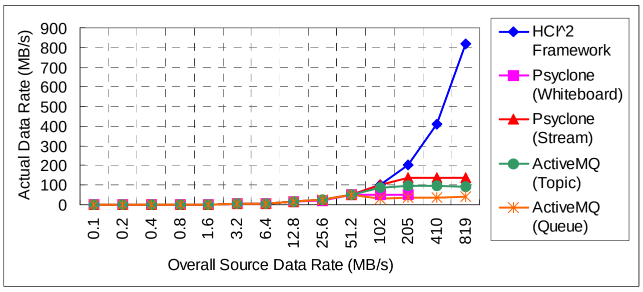


Fig. 6. Actual data rate achieved by HCI^2 Framework, Psyclone, and ActiveMQ at different levels of overall source data rate.

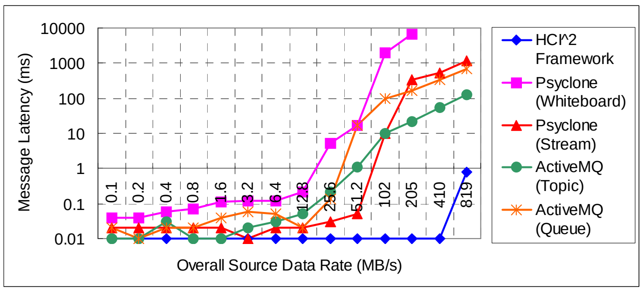


Fig. 7. Average message latency in HCI^2 Framework, Psyclone, and ActiveMQ at different levels of overall source data rate.

longer capable of delivering all messages produced by the sources at these rates. In comparison, the average message dropping rate for the data sources running in the HCI^2 Framework is constantly zero during the entire experiment.

The same trend is also visible in Fig. 6. As shown in the figure, only the HCI^2 Framework is able to achieve the same actual data rate (up to 818.4 MB/s) as the overall source data rate in the mock-up system during the whole test. All other frameworks / dispatchers reach their maximum data throughput when the overall source data rate is 204.8 MB/s. Specifically, the maximum data throughput is 138.1 MB/s for Psyclone (when using streams as dispatchers) and 94.8 MB/s for ActiveMQ (when using queues as dispatchers), respectively. Note that the maximum data throughput of both Psyclone and ActiveMQ, which are 5.24 and 3.59 times the data rate of a 30FPS 640x480 RGB video stream, respectively, may not be high enough to meet the requirement of MHCI systems for it is not uncommon for such systems to capture several high-resolution video streams simultaneously from multiple views.

Regarding the message latency (shown in Fig. 7), if we consider only values obtained when all frameworks were performing well (i.e., when no message dropping occurs), the HCI^2 Framework outperforms all tested frameworks but ActiveMQ. The HCI^2 Framework also outperforms ActiveMQ if heavier workload situations are taken into consideration.

V. Demonstration of Usage

To demonstrate the ease of use of the HCI^2 Framework for integrating heterogeneous modules, we developed a number of example modules working with a variety of hardware and software. We then integrated some of the modules into a readily-applicable system called the CamGame. This open-source demo system[2] is an interactive system allowing users to play a computer game using hand-held marker(s) and low-cost camera(s) instead of keyboard and mouse.

A. *Inclusion of Modules*

The implemented example modules are summarized in Table III. Two of these modules – the Video Capturer and the Video File Reader – are built upon Microsoft DirectShow. This decision was made because DirectShow is considered the optimal choice for developing Windows applications dealing with video and / or audio streams. Note, however, that due to its complex architecture, using DirectShow is not a trivial task. This is often the reason for developers to use simpler tools such as OpenCV and / or Windows Platform SDK's video accessing interfaces, which are less powerful than DirectShow and often suffer from compatibility issues. By incorporating the DirectShow-based video capturing and decoding into our modules, we effectively enable future users of the HCI^2 Framework to utilize DirectShow for their system development while freeing them from the workload of getting knowledgeable in using DirectShow.

The ActiveMQ Receiver / Sender and Psyclone Receiver / Sender are the 'bridging modules' used to facilitate message exchanging between HCI^2 Framework and ActiveMQ and Psyclone, respectively. Specifically, the sender module relays messages from a HCI^2 Framework channel to an ActiveMQ topic (or a Psyclone whiteboard) while the receiver module delivers messages in the opposite direction. One application of these modules is to pass (high level) messages between multiple subsystems residing on different computers through ActiveMQ or Psyclone, hence to compensate for the limitation of our shared-memory-based data transport protocol. In addition, these modules also encourage developers to reuse and extend existing systems built upon ActiveMQ and / or Psyclone by eliminating the need for rebuilding every existing component in the HCI^2 Framework.

Two other modules worth mentioning are the Tobii Eye Tracker and the MS Kinect Adapter. Built upon Tobii SDK, the Tobii Eye Tracker module tracks the user's gaze movement in real-time using the Tobii Eye Tracker X120 device with a sample rate of either 60 or 120 Hz. Apart from the gaze tracking algorithm, the module also delivers a GUI-based 'initialization wizard' tool which helps the user to choose, initialize, calibrate, and test the eye tracker device in a step-by-step manner before starting the gaze tracking process. Similar to DirectShow, the Tobii SDK is a powerful and comprehensive tool, yet rather hard to use (especially when it comes to the functions and classes related to calibration). With the inclusion of the Tobii Eye Tracker module, future HCI^2 Framework users may ignore the cumbersome details of the Tobii SDK altogether and only focus on their system development and the analysis of the captured gaze data.

TABLE III
A List of Example Modules

Module Name	Purpose	Hardware Dependencies	Software Dependencies
Video Capturer	Captures realtime video from a USB webcam	USB webcam	DirectShow and OpenCV 2.3.1
Video File Reader	Extracts video frames from an offline video file	None	DirectShow, CODECs and OpenCV 2.3.1
CMS Receiver / Sender	Exchanges messages between HCI^2 Framework and ActiveMQ	None	ActineMQ 5.3.2 and ActiveMQ / CMS SDK
Psyclone Receiver / Sender	Exchanges messages between HCI^2 Framework and Psyclone	None	Psyclone AIOS 1.1.7 and OpenAIR SDK
Message Relay	Implements a variaty of flow cotrol mechanisms	None	None
Video Synchroniser	Synchronises input video stream to a trigger signal	None	None
Tobii Eye Tracker	Tracks the user's gaze movement in realtime	Tobii Eye Tracker X120	Tobii SDK 2.4.12 [20]
MS Kinect Adapter	Captures realtime video with per-pixel depth information	Microsoft Kinect	'Kinect for Windows' SDK 1.0.3.190 [21]
Video Renderer	Renders realtime video	None	None
Viola Jones Face Detector	Detects faces from the input video stream	None	OpenCV 2.3.1
Colour Based Object Tracker	Tracks a user-selected object in the input video stream	None	OpenCV 2.3.1
Image Converter	Converts input frames to a different format	None	OpenCV 2.3.1

The MS Kinect Adapter module is based on the 'Kinect for Windows' SDK. Using Kinect and the SDK, the module captures real-time colour video with per-pixel depth information. The depth information can be delivered as either greyscale images (which are more suitable for subsequent analysis) or colour-coded images (which may be preferable for recording). In order to minimize the temporal misalignment between the colour video stream the depth field stream caused by frame dropping (which is an inherent problem of the Kinect device), the module buffers several raw frames captured from both streams and pairs them based on the frames' time-stamp. This solution helps to set up an upper bound, which is 17ms (half of the camera's frame interval), for the temporal disparity between the two image streams.

B. *CamGame: A Demo System*

To demonstrate the usage of the HCI^2 Framework IDE and to provide a tutorial example for future users, we have developed an open-source demo system called the CamGame (see Fig. 8). The baseline CamGame system enables one or two players to play a computer game using low-cost camera(s) (e.g., USB webcams) and hand-held marker(s) (any brightly-coloured rigid object may be used) instead of keyboard and mouse. The

[2] Available at: http://ibug.doc.ic.ac.uk/resources/hci2-framework/

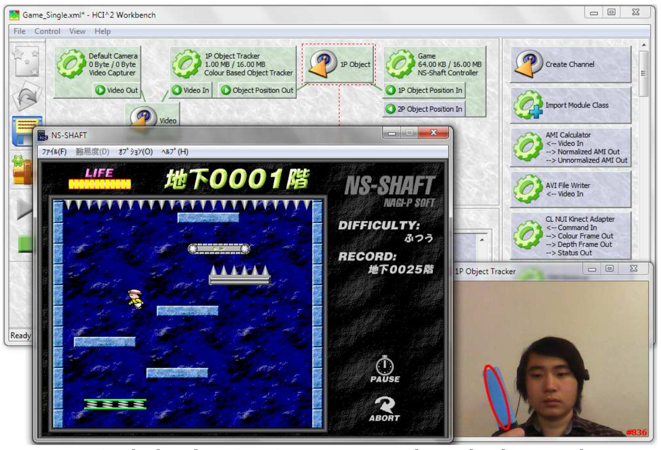


Fig. 8. The baseline CamGame system in thesingle-player mode.

CamGame system is customizable to exploit other input modalities as well, such as face position and eye gaze. In addition, since the overall structure of the system is independent from the game being controlled, the CamGame system can be reconfigured to control any conventional computer game with few modifications to the command mapper module.

The baseline system requires three processing stages: video capturing, marker tracking, and command mapping. We have developed a separate module class for each of the three tasks.

Video capturing is performed by the Video Capturer module, a console programme which captures real-time video stream using DirectShow. The video capturer defines three parameters enabling users to specify the source (the camera) and the frame size (width and height) of the captured video stream.

There are many object tracking algorithms available today [29] [30]. We intended to keep the actual processing as simple as applicable. Thus we used the mean-shift based algorithm described in [22], which is relatively easy to implement while yielding good tracking results. A brief description of the object tracking algorithm is as follows.

1) Initialization: We model the target object as a Gaussian distribution based on the colour histogram extracted from the region of interest. In order to keep the algorithm simple, we assume that the target is a uniformly coloured object. Hence, the object model is only based on the predominant hue.
2) Tracking: in each new frame, we calculate every pixel's likelihood of belonging to the target object (based on the target object colour model). We then apply the CAMSHIFT algorithm [22] to the resulting probability map to obtain an estimate of the target object position in the current frame (i.e., its centre, size, and orientation).
3) Updating: To be able to handle gradual change in illumination, the object colour model is continuously updated (with a small learning rate) using the histogram extracted from the image patch within the area of the tracking result obtained in the latest frame.

In practice, the marker tracker has been built as a module (the Colour Based Object Tracker module) that takes input from a video channel and displays the tracking results indicated by an ellipse. It requires the player to initialize the tracking process by selecting the marker in the first input video frame. Once initialized, the module will continuously produce tracking

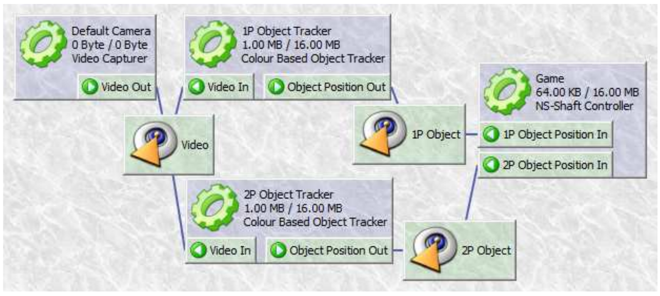


Fig. 9. The basline CamGame system in a cooperative-mode using a single camera.

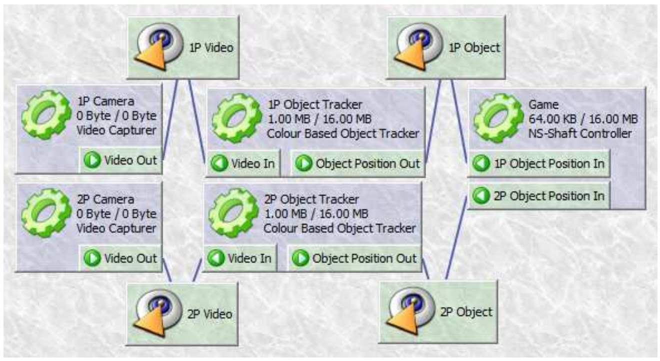


Fig. 10. The baseline CamGame system in a cooperative-mode using two cameras, one for each player.

results (including the marker's position, size, and orientation) and output them to an appropriate output channel. The module also allows the player to refine the target model by manually adjusting the track box on-the-fly.

A command mapper is used to map the input object position into an appropriate keyboard message as to control the game (note that we did not develop a game but a game controller). Since the assumption is that the game itself is only controlled by two keys, which are 'left' and 'right', the mapping scheme is rather simple, which merely maps the marker position to 'left' and 'right' according to its horizontal orientation.

The actual structure of the baseline system is illustrated in Fig. 8. It consists of 3 module instances, one of each class introduced above, connected by 2 channels. This version is used to play the game in the single-player mode. The system can be easily upgraded to enable two players to play the game in a cooperative-mode. Fig. 9 and Fig. 10 illustrate two possible versions of such an upgraded system. In the first version, a single camera is used to capture both players, while in the second version, two cameras are used, one per player. In both versions, there are two instances of the marker tracker and marker-position channel as to enable the production and delivery of the control messages for both players.

Since the game controlling mechanism and the input acquisition method are decoupled, the CamGame system can be further customized to incorporate other input modalities by reconfiguring the front-end multimodal input acquisition subsystem. As examples, the system configurations of CamGame variants using face position and eye gaze position are shown in Fig. 11 and Fig. 12, respectively. In the first variant, the object tracker is replaced by the Viola & Jones face detector [16]. This is the most commonly used face detector in the field [5] [31] [32]. Here it is used to detect the largest face in the scene and output the face position to the command mapper's

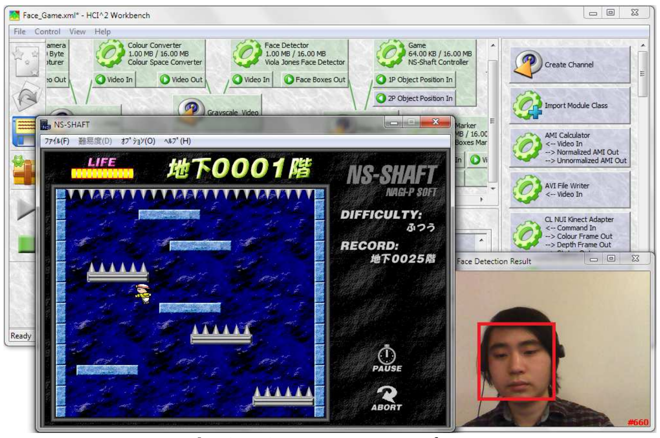
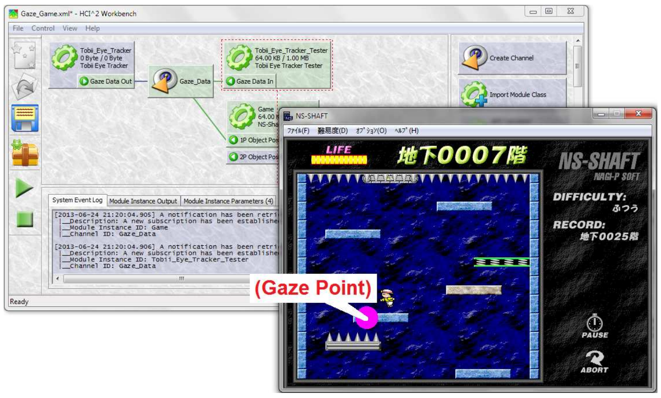


Fig. 11. The CamGame variant using face position.



```
[2013-06-24 21:20:04.905] A notification has been retri
 |__Description: A new subscription has been establishe
 |__Module Instance ID: Game
 |__Channel ID: Gaze_Data

[2013-06-24 21:20:04.906] A notification has been retri
 |__Description: A new subscription has been establishe
 |__Module Instance ID: Tobii_Eye_Tracker_Tester
 |__Channel ID: Gaze_Data
```

Fig. 12. The CamGame variant using eye gaze position.

input channel. In the second variant, the entire front-end subsystem is substituted by a Tobii eye tracker [33] to track user's eye gaze position in real-time [34]

Currently, the CamGame system is bond to a particular game called NS-SHAFT (freely available at: http://www.nagi-p.com/eng/nsshaft.html). A more general command mapper module can be developed to enable keyboard- and mouse-free control for an arbitrary game. This can be achieved by enabling the player to customize the mapping relations between input object positions and the underlying Windows messages used to control the game of his / her choice. More complex scheme may be proposed by taking temporal information into account (i.e., how long is the object on a certain location and / or how fast it changes the location). Nonetheless, according to our preliminary tests, the simple method we described here tends to produce satisfactory results for many games that do not feature intense action (e.g., Super Mario and other classic arcade games of the kind).

Last but not least, the 3rd-party software used in the CamGame system is not limited to games. With few modifications, any software accepting standard Windows keyboard and / or mouse messages can be used. Hence, the CamGame system may be easily modified for integration into various (existing) human-computer interaction systems.

VI. Conclusion

We have proposed a software framework to facilitate the development of multimodal human-computer interaction systems. This software framework, which is called the HCI^2 Framework, consists of a combination of SDK and GUI-enabled utilities to provide complete support for the system development procedure, including module programme development, system construction, testing, and redistribution.

Internally, the HCI^2 Framework adopts a P/S architecture to support flexible system structures while uses a shared-memory based data transport to guarantee reliable delivery of high data-rate message stream with low latency. The protocol is robust against crash-prone faulty modules and ensures the execution of other modules will not be affected by crashed ones. In addition, with the introduction of 'bridging modules', the HCI^2 Framework is interoperable with some existing message passing middleware including ActiveMQ and Psyclone.

To facilitate module reusing, we have refined the concept of module into two related terms, which are the module class and the module instance. Based on the design of module warehouse and the XML semantics for system configuration files, we have proposed a centralized system management scheme, which greatly simplifies system construction, testing, and redistribution.

The HCI^2 Framework is implemented as a self-contained open-source software development tool. It consists of the HCI^2 Framework SDK, which implements the aforementioned protocols, and the HCI^2 Framework IDE, which provides a complete GUI-enabled environment facilitating module development and system integration.

The quantitative comparison shows that our framework outperforms the other similar tools including Psyclone and ActiveMQ in terms of maximum data throughput and message latency under a typical single PC scenario.

To demonstrate the HCI^2 Framework's ease of use in integrating heterogonous modules, we have developed a number of example modules interacting with a variety of hardware and software, including Microsoft DirectShow, OpenCV, Tobii Eye Tracker, and Microsoft Kinect. Using these modules, we further built a readily-applicable demo system called the CamGame, which enables players to play a computer game using hand-held marker(s) and ordinary low-cost camera(s) instead of keyboard and mouse.

The redistribution package of the HCI^2 Framework (including the CamGame system) is now publicly available at: http://ibug.doc.ic.ac.uk/resources/hci2-framework/ (under BSD licence: http://opensource.org/licenses/BSD-3-Clause).

Acknowledgement

The authors would like to thank Wenzhe Shi of Imperial College London for his involvement in the development and testing of the CamGame system. We would also like to thank all members of the Intelligent Behaviour Understanding Group for testing the HCI^2 Framework. This work has been supported by the European Research Council under the ERC Starting Grant agreement no. ERC-2007-StG-203143 (MAHNOB). The current work by Jie Shen is also supported by the European Union's 7th Framework Programme [FP7/2007-2013] under grant agreement no. 288235 (FROG).

REFERENCES

[1] M. Pantic, and L. J. M. Rothkrantz, “Towards an affect-sensitive multimodal human-computer interaction”, Proceedings of the IEEE, vol. 91, no. 9, pp. 1370-1390, September 2003.

[2] A. Jaimes, and N. Sebe, "Multimodal human–computer interaction: a survey", Computer Vision and Image Understanding, vol. 108, no.1-2, pp. 116-134, 2007.

[3] M. Pantic, A. Pentland, A. Nijholt and T.S. Huang, “Human computing and machine understanding of human behavior: a survey”, Artificial Intelligence For Human Computing, T.S. Huang, A. Nijholt, M. Pantic and A. Pentland, Eds. Springer, Lecture Notes in Artificial Intelligence, vol. 4451, pp. 47-71, 2007.

[4] M. Pantic, A. Nijholt, A. Pentland and T. Huang, “Human-centred intelligent human-computer interaction (HCI^2): how far are we from attaining it?”, Int’l Journal of Autonomous and Adaptive Communications Systems, vol. 1, no. 2, pp. 168-187, 2008.

[5] Z. Zeng, M. Pantic, G.I. Roisman and T.S. Huang, “A survey of affect recognition methods: audio, visual, and spontaneous expressions”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 1, pp. 39-58, 2009.

[6] A. Pentland, “Looking at People: Sensing for Ubiquitous and Wearable Computing”, Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 1, pp. 107-119, 2000.

[7] L. Maat, and M. Pantic, “Gaze-X: adaptive affective multimodal interface for single-user office scenarios”, Artificial Intelligence for Human Computing, T. S. Huang, A. Nijholt, M. Pantic, and A. Pentland, Eds. Springer, Lecture Notes in Artificial Intelligence, vol. 4451, pp. 251-271, 2007.

[8] “MSDN: DirectShow (Windows)”, Dec. 4, 2008. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd375454(VS.85).aspx.

[9] “Communicative Machines: Pscylone”, 2007. [Online]. Available: http://www.cmlabs.com/psyclone/.

[10] “Apache ActiveMQ”, 2009. [Online], Available: http://activemq.apache.org/.

[11] J-Y. Lawson, J. Vanderdonckt, and B. Macq, “Rapid prototyping of multimodal interactive applications based on off-the-shelf heterogeneous components”, Adjunct Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology, pp. 41-42, 2008.

[12] J. Shen, and M. Pantic, “A Software Framework for Multimodal Human-Computer Interaction Systems”, Proceeding of IEEE Inrernational Conference on Systems, Man and Cybernectis, pp. 2038-2045, 2009.

[13] M. Pantic, R.J. Grootjans, and R. Zwitserloot, “Fleeble agent framework for teaching an introductory course in AI”, IADIS International Conference Cognition and Exploratory Learning in Digital Age, pp. 525-530, 2004.

[14] K. R. Thorisson, H. Benko, D. Abramov, A. Arnold, S. Maskey, and A. Vaseekaran, “Constructionist design methodology for interactive intelligences”, AI Magazine, vol. 25, no. 4, pp.77-90, 2004.

[15] S. Meyers, “Minimize compilation dependencies between files”. Efective C++: 50 Specific Ways to Improve Your Programs and Designs 2nd Edition, pp. 140-148, Addison Wesley, October, 1997.

[16] P. Viola, and M. J. Jones, “Robust real-time face detection”, International Journal of Computer Vision, vol. 57, no. 2, pp. 137-154, 2004.

[17] D. Comaniciu, V. Ramesh, P. Meer, “Real-time tracking of non-rigid objects using mean-shift”, IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 438-445, 2000.

[18] “MSDN: Inter-Process Communications”, Feb. 12, 2009. [Online]. Available:http://msdn.microsoft.com/en-us/library/aa365574(VS.85).aspx.

[19] P. Dabak, S. Phadke, and M. Borate, “Local procedure call”, Undocumented Windows NT, Foster City: M&T Books, 1999, pp. 143-189.

[20] Tobii SDK, 2010. [Online]. Available: http://www.tobii.com/en/assistive-technology/global/products/partner-software/third-party-program/sdk/

[21] Microsoft Kinect for Windows SDK, Feb. 01, 2012. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx.

[22] G. R. Bradski, "Computer Vision Face Tracking for Use in a Perceptual User Interface", Intel Technology Journal, No. Q2, 1998. [Online]. Available: ftp://download.intel.com/technology/itj/q21998/pdf/camshift.pdf

[23] “MSDN: Mutex Objects”, Mar. 07, 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/ms684266(v=vs.85).aspx.

[24] “MSDN: Interlocked Variable Access”, Mar. 07, 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/ms684122(v=vs.85).aspx.

[25] A. Camurri, S. Hashimoto, M. Ricchetti, A. Ricci, K. Suzuki, R. Trocca and G. Volpe, “Eyesweb: Toward gesture and affect recognition in i.nteractive dance and music systems”, Computer Music Journal vol. 24, no. 1, pp. 57-69, 2000.

[26] M. Schröder, “The SEMAINE API: Towards a Standards-Based Framework for Building Emotion-Oriented Systems,” Advances in Human-Computer Interaction, vol. 2010, Article ID 319406, 21 pages, 2010. doi:10.1155/2010/319406.

[27] M. Schroder, E. Bevacqua, R. Cowie, F. Eyben, H. Gunes, D. Heylen, M. Maat, G. Mckeown, S. Pammi, M. Pantic, C. Pelachaud, B. Schuller, E. Sevin, M. F. Valstar and M. Woellmer, “Building Autonomous Sensitive Artificial Listeners”, IEEE Transactions on Affective Computing, vol. 3, no. 2, pp. 165-183, 2011

[28] W. Richard Stevens, “Chapter 12. Shared Memory Introduction”. Unix Network Programming: Interprocess Communicattion, pp. 303-323, Prentice Hall PTR, 1999.

[29] A. Yilmaz, O. Jamed and M. Shah, “Object Tracking: A Survey”, ACM Computing Surveys, vol. 38, no. 4, article 13, 45 p., 2006.

[30] H. Yang, L. Shao, F. Zhen and L. Wang and Z. Song, “Recent advances and trends in visual tracking: A review”, Neurocomputing, vol. 74, no. 18, pp. 3823-3831, 2011.

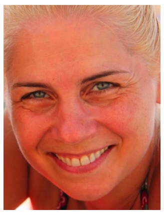
[31] M. Pantic, “Machine Analysis of Facial Behaviour: Naturalistic and Dynamic Behaviour”, Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 364 no. 1535, pp. 3505-3513, 2009.

[32] MF Valstar, M Mehu, B Jiang, M Pantic and K Scherer, “Meta-Analysis of the First Facial Expression Recognition Challenge”, IEEE Transaction on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 42, no. 4: pp. 966-979, 2012.

[33] Tobii T60 & T120 Eye Tracker. [Online]. Available: http://www.tobii.com/en/eye-tracking-research/global/products/hardware/tobii-t60t120-eye-tracker/

[34] J. Lichtenauer, J. Shen, M. F. Valstar and M. Pantic, "Cost-Effective Solution to Synchronised Audio-Visual Data Capture Using Multiple Sensors", Image and Vision Computing, vol. 29, no. 9, pp. 666-680, 2011.

Jie Shen is a Ph.D. candidate at Imperial College London, Department of Computing, U.K. He received B.Eng. in Electric Engineering in 2005 from Zhejiang University, China, and M.Sc. in Advanced Computing in 2008 from Imperial College London, U.K. He worked as a research assistant, in intelligent video surveillance, at Institute of Automation, Chinese Academy of Sciences from 2005 to 2007. His current research interests include affect-sensitive human-computer interaction and software / hardware platform for HCI systems. He is a student member of IEEE.



Maja Pantic is Professor in Affective and Behavioural Computing at Imperial College London, Department of Computing, UK, and at the University of Twente, Department of Computer Science, the Netherlands. She received various awards for her work on automatic analysis of human behaviour including the European Research Council Starting Grant Fellowship 2008 and the Roger Needham Award 2011. She currently serves as the Editor in Chief of Image and Vision Computing Journal and as an Associate Editor for both the IEEE Transactions on Systems, Man, and Cybernetics Part B and the IEEE Transactions on Pattern Analysis and Machine Intelligence. She is a Fellow of the IEEE.