m+m: A novel Middleware for Distributed, Movement based Interactive Multimedia Systems

Ulysses Bernardet
Simon Fraser University
Vancouver, Canada
ubernard@sfu.ca

Dhruv Adhia
H Plus Technologies
Vancouver, Canada
dhruv@hplustech.com

Norman Jaffe
Vecima Networks
Vancouver, Canada
turing@shaw.ca

Johnty Wang
McGill University
Montreal, Canada
johnty.wang@mail.mcgill.ca

Michael Nixon
Simon Fraser University
Vancouver, Canada
mna32@sfu.ca,

Omid Alemi
Simon Fraser University
Vancouver, Canada
oalemi@sfu.ca

Jordon Phillips
Simon Fraser University
Vancouver, Canada
jjp14@sfu.ca

Steve DiPaola
Simon Fraser University
Vancouver, Canada
sdipaola@sfu.ca

Philippe Pasquier
Simon Fraser University
Vancouver, Canada
pasquier@sfu.ca

Thecla Schiphorst
Simon Fraser University
Vancouver, Canada
thecla@sfu.ca

ABSTRACT
Embodied interaction has the potential to provide users with uniquely engaging and meaningful experiences. m+m: Movement + Meaning middleware is an open source software framework that enables users to construct real-time, interactive systems that are based on movement data. The acquisition, processing, and rendering of movement data can be local or distributed, real-time or off-line. Key features of the m+m middleware are a small footprint in terms of computational resources, portability between different platforms, and high performance in terms of reduced latency and increased bandwidth. Examples of systems that can be built with m+m as the internal communication middleware include those for the semantic interpretation of human movement data, machine-learning models for movement recognition, and the mapping of movement data as a controller for online navigation, collaboration, and distributed performance.

Author Keywords
Real-time interaction; middleware; movement;

MOCO'16, July 05-06, 2016, Thessaloniki, GA, Greece
© 2016 ACM. ISBN 978-1-4503-4307-7/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2948910.2948942

ACM Classification Keywords
C.3. SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS]: Real-time and embedded system, D.2.11. Software Architectures: Domain-specific architectures

INTRODUCTION
We can observe converging trends in human-computer interaction, cognitive science, and the consumer market: firstly, *affective computing*, the research and development of software systems that can recognize, interpret, process, and ultimately harness affective responses [16], has become a mainstream topic. Secondly, cognitive science has shown an increasing interest in *embodied cognition*, i.e. the proposition that the mind "is not only connected to the body but that the body influences the mind" [26]. Thirdly, in the consumer market we can observe a trend towards the engagement of individual non-experts in the self-monitoring and -analysis of biological, physical, behavioral, or environmental information referred to as "*quantified self*" [24]. What these trends share is the notion that to better understand humans, and/or to build better technology, we need to take into account the body, and with it, movement.

This motivation is met at the technological level by recent developments in the hardware and software domains. In the former we observe a proliferation and democratization of real-world behavior and movement sensors – on the one hand in the form of affordable sensors such as Microsoft Kinect, Structure Sensor, Wii Balance Board and Remote, and Leap Motion, and on the other hand through wearable technology [25]. In the latter, the software domain, systems for making inferences based on movement such as gesture recognizer

[6], or through the application of the Laban Movement Analysis [11], a well-established system for describing movement, have gained traction.

If our goal is to build real-time, distributed interactive systems that deploy heterogeneous sensors and effectors, we need the means for recording and sensing, storing and retrieving, analyzing and understanding, and displaying, sonifying, and visualizing movement data. And, crucially, we need a way to connect these elements together, and have them communicate with each other. In this manuscript we describe the "m+m: Movement + Meaning" software framework that, broadly speaking, enables users of different domains and levels of expertise to construct real-time, interactive systems that are based on movement. m+m caters to users from a range of backgrounds including, but not limited to, performance art, computer engineering, science, and health technology. At the software engineering level m+m is based on the well-established Yet Another Robot Platform (YARP) [15]). The acquisition, processing, and rendering of movement data can be local or distributed, real-time or off-line, and m+m provides a range of ready-made interfaces to devices, and existing software frameworks. A graphical user interface (GUI) provides a tool for managing and monitoring nodes in the network.

RELATED WORK

The requirements flagged above are met to varying degrees by existing software solutions. Here we will give a brief overview of existing middleware services, frameworks, communication libraries, and integrated packages. A *middleware* service is as a general-purpose service that sits between platforms and applications, and that is defined by the Application Programming Interfaces (APIs) and protocols it supports [5]. A number of classification schemas for middleware exist, e.g. [17] distinguishes between Transactional middleware for distributed synchronous transactions, Procedural middleware to execute Remote Procedure Calls (RPC), Message-oriented middleware that provide communication through messages (e.g. IBM WebSphere MQ[1], Apache ActiveMQ[2]) and object-oriented middleware that extends RPC with concepts from object-orientation (e.g. Java Remote Method Invocation (RMI[3]), and Common Object Request Broker Architecture (CORBA[4]). The advantages of most of these middleware frameworks are that they are well supported, facilitate development, and provide a solid basis for setting up and managing communication between nodes. The downside is that many of them are closed source, and have a notoriously large overhead and steep learning curve. *Frameworks* such as Processing [20], openFrameworks[5], MAX[6], and Pure Data [18] are widely used in the artistic and human-computer interaction community. These frameworks put the emphasis on output and rendering, and, while some of them provide built-in networking capabilities, they are generally confined to point-to-point networking and limited in capacity and parallelism. Last but not least, there exists a number of open source and commercial *communication libraries* that differ in the supported protocols, platforms, and level of abstraction at which they are implemented. Examples of open source libraries include Open Sound Control (OSC[7]), Torque Network Library (OpenTNL[8]), POCO C++[9], ADAPTIVE Communication Environment (ACE[10]), and ENet[11], while examples of commercial libraries are RakNet[12], and Zoidcom network[13]. Most of these libraries are agnostic as to what content they transport, in the sense that they do not provide protocol definitions and do not provide built-in means to setup, manage, and monitor connections between nodes. *Integrated solutions* closest to the approach presented in this manuscript include the StreamInput advanced sensor processing and user interaction application programming interface (API) developed by the Khronos working group[14].

In the domains of pervasive and ubiquitous computing a number of comprehensive middleware systems have been developed. Some of these systems are specialized e.g. for ubiquitous tracking, where data from spatially distributed and heterogeneous tracking sensors need to be integrated, such as the CORBA based Ubitrack framework [19] and its predecessor DWARF [13]. Other systems have wider application domains such the Proximity Toolkit that supports proxemics based interactions [14], frameworks for building distributed tangible and multi-modal interfaces such as Ensemble [7] and DynaMo [2], respectively, and the Stanford Interactive Room Operating System (iROS), a general purpose software framework which allows applications to communicate with each other and with user interface devices in a dynamically configurable way [9]. Possibly closest to m+m in terms of scope and design philosophy is the real-time Java-based middleware OSA+ [22], supports the construction of distributed, heterogeneous, and highly scalable systems.

The development of m+m is motivated by the set of specific requirements for the development of the types of systems outlined initially. The middleware should be a largely self-sufficient system, enabling users with little technical background to build interactive systems. Hence, m+m needs

[1] http://ibm.com/software/products/en/ibm-mq
[2] http://activemq.apache.org
[3] http://goo.gl/SqNX33
[4] http://omg.org/spec/CORBA/
[5] http://openframeworks.cc
[6] http://cycling74.com/products/max/
[7] http://opensoundcontrol.org
[8] http://opentnl.org
[9] http://pocoproject.org
[10] http://cs.wustl.edu/~schmidt/ACE.html
[11] http://enet.bespin.org
[12] http://jenkinssoftware.com
[13] http://zoidcom.com
[14] http://khronos.org/streaminput/

to be able to provide turnkey solutions, i.e. not merely an API. Hand in hand with this requirement goes the need to provide a library of interfaces to established, predominantly movement data acquisition sensors, ranging from Kinect to professional motion capture systems. To facilitate interoperability, a standardized protocol, specifically tailored to movement-based data has to be an integral part of the middleware. This is a key feature, that is – basic network libraries are missing. The middleware needs to provide high bandwidth data transmission that allows data to be streamed raw, or minimally processes sensor information via processing components in real time. In a fluid, exploration- and development-oriented deployment scenario, decoupling of components is essential, allowing users to connect and disconnect nodes at run time. Last but not least, the development of the middleware was motivated by the desire to provide users with an easy to install, and open source system.

M+M ARCHITECTURE

Conceptual framework

m+m is endorses a component-based architecture of logically independent entities, and is based on the well-established open source middleware “YARP” [15]. Key features of the m+m middleware are portability between different platforms, a small footprint in terms of computational resources, and high performance regarding latency and bandwidth. The first two properties are achieved by m+m being cross-platform, with support for all major operating systems (Windows, MacOS, and Linux), the core binary distribution being portable (for convenience, binary installers are provided), and m+m having a small footprint (the windows distribution requires less than 100.0MB disk space). The high performance in terms of latency and

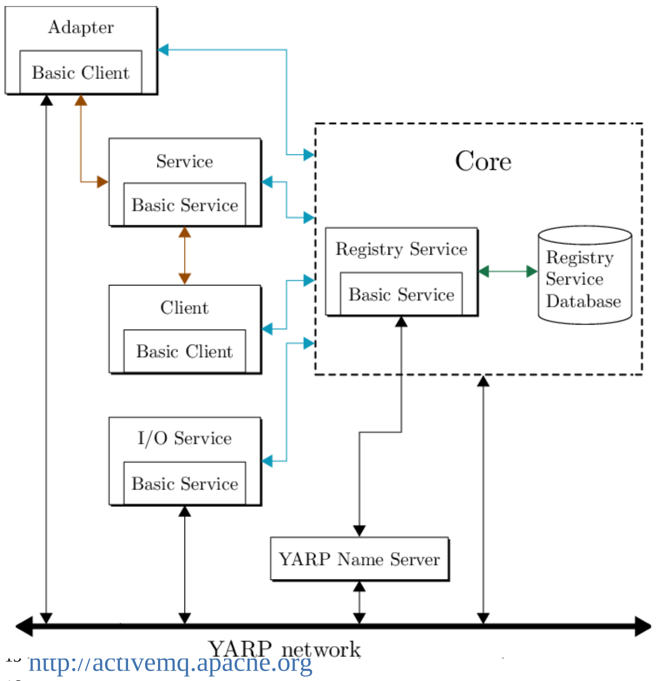

Figure 1: Logical organization of an m+m system. Installation. Brown lines represent client-service communication, the blue lines represent communication with the Registry Service and the black lines represent YARP communication paths.

[15] http://activemq.apache.org

[16] http://ros.org

bandwidth is achieved by two main mechanisms: the middleware itself does not handle any communication, but rather establishes direct point-to-point communication between end-nodes. Secondly, all communication is based directly on native protocols with as little overhead as is possible. Depending on the specific needs and system topology, communication can be done via TCP/IP, UDP, or shared memory. The properties listed above are key advantages over other middleware platforms such as Apache ActiveMQ[15], or ROS[16]. Additionally, m+m, by virtual of being based on YARP, provides bindings for multiple languages (C++, Perl, Python, Java), and comes with a number of basic interfaces to hardware devices such as microphones and cameras.

m+m middleware

All m+m programs utilize YARP to facilitate communication – it provides one-to-many output and many-to-one input mechanisms, as well as a network-based name server (Figure 1). These input and output mechanisms are implemented via “mini-server” code that is a fundamental component of YARP. The “YARP network” represents the aggregated TCP/IP, UDP and shared memory connections that exist when YARP is active – YARP itself does not use any special protocols and can operate over a variety of physical networks. What m+m provides is a standardized client-service mechanism, a set of naming conventions for YARP ports and a centralized database that is used to locate services within the YARP network. The YARP Name Server is used to obtain the physical network address of each m+m channel, given the name of the channel. Once the network address is known, all communication between entities in m+m is via either TCP/IP or UDP packets, using YARP low-level mechanisms to manage the connections. Services perform a sequence of requests and responses with the Registry Service when they start, in order to be accessible from other m+m entities. Once started, they can receive requests from client applications, data streamed via their input channels, external sensors or generated algorithmically, and transmit data via their output channels, external transducers or files. Additionally, they will receive periodic requests from the Registry Service, inquiring as to their “health” and availability.

m+m Components

Sensors The term "Sensor" refers to a wide range of components providing input to the middleware. Technically a “sensor” ranges from a hardware device (e.g. camera) to high-level processing entities that extract semantically meaningful information from a physical sensing device. Currently, the following sensors are supported: all native YARP devices (serial, video, audio, etc.), Microsoft Kinect (version 1 and 2), Leap Motion[17], AnTS Overheard tracking [3], several motion capture systems (organic motion

[17] http://leapmotion.com

OpenStage, OptiTrack NatNet, Vicon DataStream), biosignals acquisition hardware (BITalino, Thought Technology ProComp2), and sensor data from iOS devices.

Effectors An effector is a component that produces output perceivable by users. As with the sensors, the effectors are interfaced at different semantic levels and are equipped with different levels of autonomy. Currently the following effectors are supported: iDanceForms[18], game engines Unity 3D (unity3d.com) and Unreal Engine[19], and SmartBody (via an ActiveMQ adaptor).

Processing Components The role of a processing component is to mediate between inputs into the system and output generated by the system. Examples of functionality implemented in processing components include feedforward and feedback controllers, psychological models, cognitive architectures, artificial neural networks, machine learning modules, and gesture classifiers. Feature extraction modules are a type of processing component that play a central role in the interpretation of meaning from movement information and can be used e.g. for on-line semantic inferences based on the Laban Movement Analysis [12] that has been successfully used to train dancers, animate characters and automatically segment motion capture input. Currently supported processing components include modules implemented directly in C++, perl or python (via SWIG based language bindings to YARP), MathWorks Matlab and Simulink[20], Processing[21], MAX (cycling74.com/products/max/), large-scale neuronal system simulator iqr[22] [4], and openFrameworks.

m+m GUI (Manager Utility)

The strong separation of components into individual executables in m+m can lead to a usability penalty. To mitigate this issue, m+m provides a graphical tool for managing the system components and the connections between them. The m+m Manager Utility application provides a GUI-based view of the state of connections, services and clients within the installation (Figure 2). The m+m manager Utility application displays a single window view of the connections within a YARP network, with features designed to make management of an m+m installation easier. In the diagram of the network topology, standard YARP components, m+m simple clients, m+m services, and m+m adapters are identified by their type (input or output), IP address, and the number and name of their port. Tags e.g. “S” and “C” are used to identify the type of component in the diagram. Connections between ports are shown as lines with one of three thicknesses and one of three colors. From thinnest to thickest lines, the representations indicate: simple YARP network connections; connections between input/output services; and connections between clients and services. Complementary to the thickness of the edges, the colors indicate whether the connection is TCP/IP (teal), UDP (purple), or shared memory (orange). Next to creating and deleting connections, the m+m GUI provides users with numerous ways to manage their m+m system. Using the tool, users can restart and stop running m+m services and adapters, start and restart the Registry services, and launch registered m+m components. Key managerial features are the ability to display information about a service or adapter, enabling and disabling the collection of service

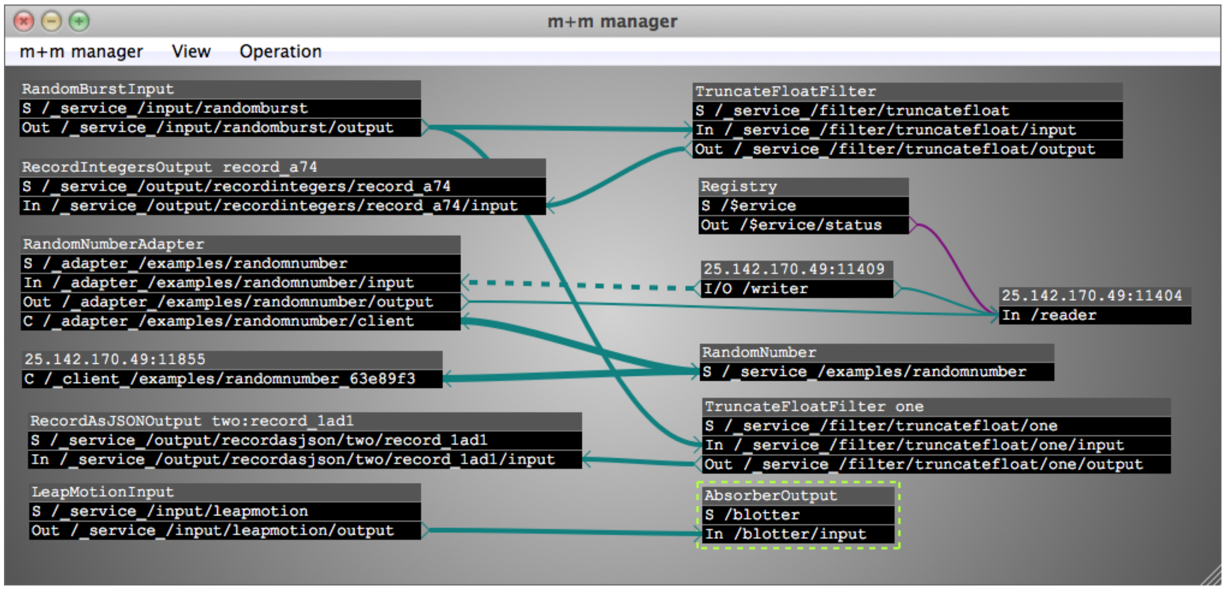
m+m manager
m+m manager
View
Operation
RandomBurstInput
S /_service_/input/randomburst
Out /_service_/input/randomburst/output
RecordIntegersOutput record_a74
S /_service_/output/recordintegers/record_a74
In /_service_/output/recordintegers/record_a74/input
RandomNumberAdapter
S /_adapter_/examples/randomnumber
In /_adapter_/examples/randomnumber/input
Out /_adapter_/examples/randomnumber/output
C /_adapter_/examples/randomnumber/client
25.142.170.49:11855
C /_client_/examples/randomnumber_63e89f3
RecordAsJSONOutput two:record_1ad1
S /_service_/output/recordasjson/two/record_1ad1
In /_service_/output/recordasjson/two/record_1ad1/input
LeapMotionInput
S /_service_/input/leapmotion
Out /_service_/input/leapmotion/output
TruncateFloatFilter
S /_service_/filter/truncatefloat
In /_service_/filter/truncatefloat/input
Out /_service_/filter/truncatefloat/output
Registry
S /$ervice
Out /$ervice/status
25.142.170.49:11409
I/O /writer
25.142.170.49:11404
In /reader
RandomNumber
S /_service_/examples/randomnumber
TruncateFloatFilter one
S /_service_/filter/truncatefloat/one
In /_service_/filter/truncatefloat/one/input
Out /_service_/filter/truncatefloat/one/output
AbsorberOutput
S /blotter
In /blotter/input

Figure 2: The m+m graphical user interface (GUI) is used to create and manage connections between nodes connected to the m+m middleware, to start and stop m+m services, and to display information about static and dynamic properties of nodes.

[18] http://credo-interactive.com
[19] http://unrealengine.com
[20] http://mathworks.com
[21] http://processing.org
[22] http://iqr.sf.net

metrics about the activity on each port of the service (e.g. the number of bytes and number of messages sent to and from the port).

Registry Service

The Registry Service application is a background service that is used to manage other services and their connections. Its primary purpose is to serve as a repository of information on the active services in an m+m system. The Registry Service provides this feature by maintaining searchable descriptions of the active input/output services, hence allowing application to find and connect to those services. Within the m+m system, the central Registry Service plays a key role in enhancing the manageability of complex distributed systems with potentially large numbers of components. Without such a system, the user has to manually keep track of what system is running where, and what services are provided on which port.

m+m Utilities

The utility programs that are part of m+m provide access to the processes that are running in the m+m installation. Although native YARP commands can be used to manage the network connections, it is recommended that the more specialized m+m tools be used to avoid inconsistencies. These m+m utilities include tools for the inspection of activities and components of the m+m system, and to provide displays for: the active services in the m+m installation; the clients for services that have YARP network connections with persistent state; the primary channels belonging to a service matching a given criterion; information on requests for one or more active services; and measurements for the channels of one or more active services. Additionally, m+m provides applications that allow recording streams of YARP values to an external file. These applications respond to the standard Output service requests and can be also be used as standalone data generators.

Integration of the movement database “MoDa"

The Movement Database (MoDa) is used to store motion capture data associated with video, and qualitative annotations at different semantic levels. Database information can be queried by, and streamed to, any node attached to the m+m middleware. Conversely, nodes in the m+m network can request data to be stored in the database. MoDa is built around a Ruby on Rails application that stores info in a MySQL database (mysql.com). Through the web front-end, researchers can both access and upload movement data. Each file or group of files can also be viewed in the accompanying “MoVa” movement visualizer [1]. MoDa provides programmatic access through the use of a standardized RESTful API that allows communication using HTTP message passing. As the middleware server has all the appropriate API requests programmed into it, a middleware client can make requests to the server in an abstract manner. Once a user authenticates through the client, they are able to communicate with MoDa.

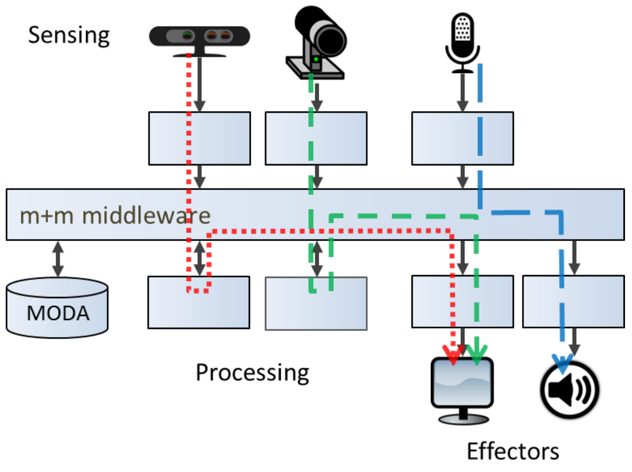

Figure 3: Illustration of the "multipath" capability of an m+m based system.

Standard protocols

Lacking standard protocols for representing messages requires users to define custom data structures. This potentially impairs interoperability and ease of use because the protocols can vary between users and between applications. As mentioned above, the m+m Registry Service allows users to query the syntax and semantics of messages. Complementary to this service, m+m uses a set of standard, interoperable sensor protocols. The basic message packaging in YARP is in the form of “Bottles” that can be containers for primitive types, lists, and “Properties” i.e. associations between tags and values. Bottles are recursive in that they can contain Bottles themselves. Based on the mechanisms provided by YARP, m+m specifies structures of sensor protocols for Kinect, Leap, Vicon, and AnTS tracking. The Extended Backus–Naur Form of these protocols is as follows:

```
x = float; y = float; z = float; w = float; id =
string; tag = string;
quaternion = w, x, y, z;
position3D = x, y, z;
position2D = x, y;
joint = tag, position3D , quaternion;
skeleton = id, joint, {joint};
palm = joint, {joint};
user = id, position2D;
Kinect = skeleton;
Leap = palm;
Vicon = skeleton, {skeleton};
AnTS = user, {user};
```

Prototypical system

One of the main advantages of building systems based on m+m is its "multipath" feature, i.e. the ability to build systems where the information from the same source is concurrently processed by multiple instances without the processing instances interfering with each other or altering the information source (within the constraints of the overall network bandwidth). Figure 3 illustrates such a multipath

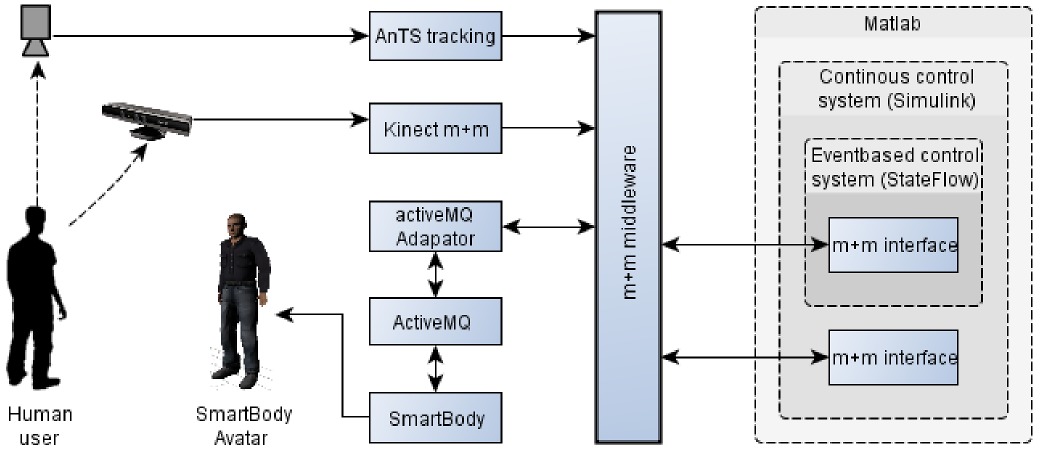

Figure 4: m+m based system for real-time interaction between human and a virtual character

system. Using a Microsoft Kinect, a microphone, and a camera, information from the environment is acquired. The information from the first two sensors is then passed through processing components, e.g. for posture and gesture analysis. Unprocessed, in the case of the microphone, and processed information is then fed to the effectors, such as a monitor and a speaker respectively. It is important to note that all information is acquired, transmitted, processed, and displayed in parallel, without mutual dependencies between the components.

EVALUATION

The subsequent sections give an overview of a number of distributed, real-time interactive systems that have been built using the m+m middleware. Real-time in the current context means that the data is processed and transmitted within the limits of what is perceivable as a delay by an observer. Generally, this ability depends on the processing speed of the nodes (e.g. the motion capture system) combined with the transportation bandwidth and lag. Each of these systems serves to illustrate specific aspects of the m+m middleware.

Distributed real-time mixed-reality dance performance

This example highlights the use of m+m in a performance artistic context where e.g. several dancers co-perform across spatially distributed locations, or a choreographer interacts with performers in real-time over large distances. The concrete system we describe here connects motion acquisition systems at two locations: firstly, in Montreal, at the Computer Research Institute of Montréal (CRIM), hand and finger movement is recorded using a Leap Motion controller. Secondly, motion capture data from a Vicon motion capture system located in Vancouver at the Emily Carr University provides the movements of two dancers: one dancer with a full motion capture suit and a second dancer with wands. Data from the first dancer is mapped onto a humanoid character in the virtual space and the second dancer's movements drive ribbons in the same space. Concurrently information from the Montreal site determines

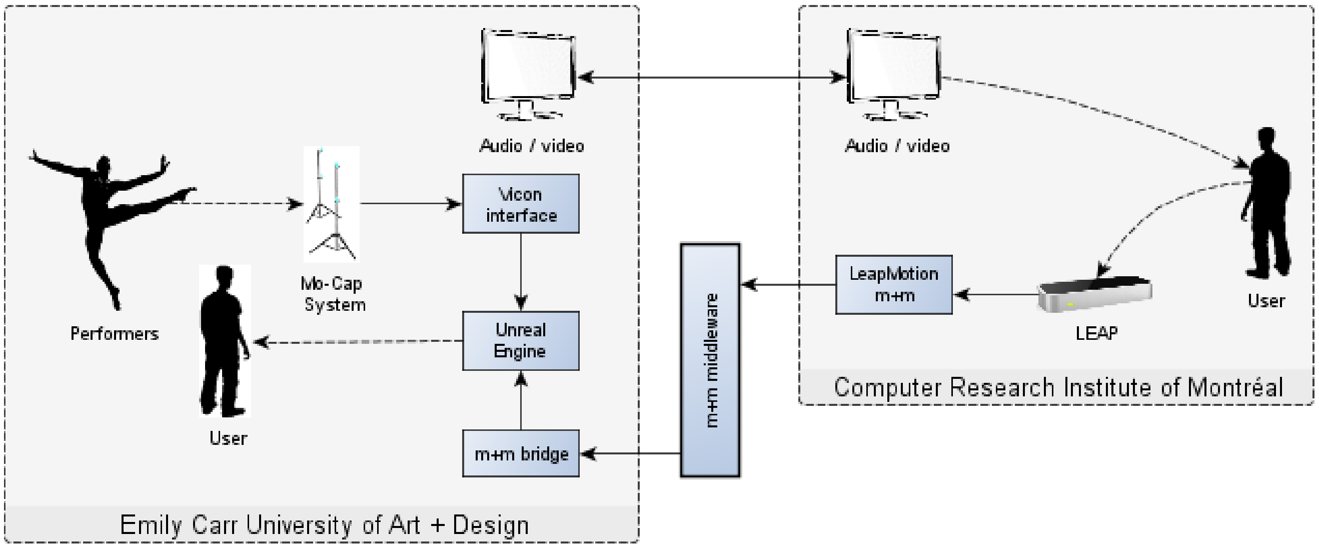

Figure 5: Architecture of the system for a real-time mixed-reality dance performance.

the locations of spotlights in the virtual space. A cluster of computers located in Vancouver provides the logical backbone for the performance: one system acts as the m+m Registry server, one acts as the YARP name registry, another is the motion capture source, one is the m+m system monitor and the last system generates the visual representation of the virtual space. The computer-based communication is via a secure software-based VPN system, that creates a single subnet between the participating computers. The application that generates the virtual space receives its input via a high-speed connection from the local motion capture system and a "bridged" connection from CRIM which is provided by an m+m application that interfaces to the Leap Motion controller. The use of the m+m backbone allows network-address-independent references to the sources and destinations of the messages as well as real-time monitoring and control of the communication paths; the m+m applications dynamically establish their identity and locations, register themselves with the globally-visible m+m Registry server and then are connected at the time of the performance via the m+m system monitor. By using m+m the participants in the performance are able to quickly setup and execute the performance.

System for real-time, real-world interaction between humans and virtual characters

This example illustrates the use of m+m in the construction of a distributed system in which a human is interacting with a virtual character – a realistic 3D representation of a human – in real-time. Such systems can be used e.g. in education and training in performing arts, psychological training and counseling, sports training etc. [8]. In the concrete case elaborated here, the system is used to develop a biologically and psychologically grounded cognitive architecture for the control of nonverbal behavior of a virtual humanoid character during dynamic interactions with human users [21]. Figure 4 illustrates how the scenario integrates heterogeneous sensing and data processing with state-of-the art virtual human technology, and psychology and cognitive science grounded control models. The position of the user is sensed with an overhead tracking camera, and computed using the tracking software AnTS [3]. During the simulation, an m+m plugin for MathWorks Simulink[23] continuously reads the users' location. The hybrid discrete-continuous control system is implemented using MathWorks Simulink and Stateflow[24], and controls the behavior of the 3D character, by sending Behaviour Markup Language (BML) [10] commands to the character animation system SmartBody [23] via the m+m middleware. The interface between m+m and the SmartBody system is realized via a bi-directional adaptor to the ActiveMQ middleware. To accommodate for the high resource needs of components such as the tracking system, and the SmartBody 3D rendering, the system is distributed over three PCs running the Windows operating system. In the future, additional inputs to the system are planned to be integrated such as gesture recognition based on data from a custom-built "data glove", and posture as classified based on information from the Microsoft Kinect sensor[25].

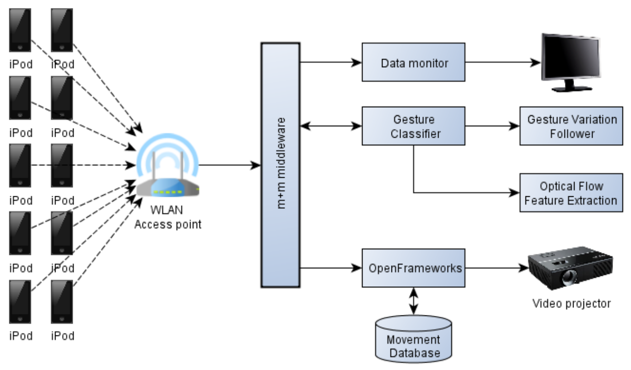

Figure 6 Architecture of the "Longing and Forgetting" multi-user interactive video installation.

Multi-user interactive video installation "Longing and Forgetting"

The "Longing and Forgetting" installation, deployed at the Surrey UrbanScreen venue in British Columbia, Canada, demonstrates the usage of the m+m middleware in constructing an interactive system involving 10 mobile devices and a central server application. The server application models and renders intelligent video agents that respond to user input from the mobile applications, and the result is projected onto an outdoor screen (Figure 6).

In the installation, participants use mobile devices (Apple iPod touch) to select and control agents that are projected onto a wall by pointing the devices at the agents and then moving in the desired direction. The accelerometer and gyroscope sensor data is filtered and combined on the mobile device, and sent to the server via the middleware. The server then performs further processing on the input data to determine the selection of a virtual agent (done via hovering over an agent) and then movement (fast jerking motion of the pointer along a certain direction). Once a movement command is issued to an agent, the internal transition of the agent is computed and an output video is selected from the movement database to execute the movement. In this example, the middleware facilitates the discovery, connection and communication of sensor data between the mobile devices and the server. Sensor processing can be done both on the mobile device, as well as the server application, depending on the computational requirements and desired features. Software bindings for the middleware interface are implemented for both the mobile and desktop platforms, and can be used by any application on supported platforms.

[23] http://mathworks.com/products/simulink/
[24] http://mathworks.com/products/stateflow
[25] http://microsoft.com/en-us/kinectforwindows/

Another feature of the system afforded by the middleware is that the sensor data, if desired for testing, deployment of new features, etc., can be dynamically plugged into other systems on the network, without any modifications either to the code or the operational mode of the application running on the existing devices.

SUMMARY AND CONCLUSION

In this paper we present the m+m middleware, the development of which is motivated by the unprecedented confluence of trends in embodied cognition, affective computing, and quantified self with a surge in the proliferation of affordable sensing devices. With its unique combination of ease of setup and configuration, high performance, and flexibility, m+m facilitates the development and deployment of distributed, real-time interactive systems in artistic, research, and commercial domains. Current limitations of m+m are that not all operations can be done via the graphical user interface, and the lack of a generic data visualization module. Future steps in the development of m+m include built-in support for generic multisensory data fusion and cross-modal mapping, a tighter integration of feature extraction methods, and the addition of further capabilities to the graphical user interface.

ACKNOWLEDGMENTS

This work was partially supported by "Moving Stories" and "m+m: Moving and Meaning" Canadian SSHRC and CANARIE grants respectively.

REFERENCES

1. Omid Alemi, Philippe Pasquier, and Chris Shaw. 2014. Mova: Interactive Movement Analytics Platform. *Proceedings of the 2014 International Workshop on Movement and Computing - MOCO '14*, ACM Press, 37–42. http://doi.org/10.1145/2617995.2618002
2. Pierre-Alain Avouac, Philippe Lalanda, and Laurence Nigay. 2012. Autonomic management of multimodal interaction. *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '12*, ACM Press, 35. http://doi.org/10.1145/2305484.2305493
3. Sergi Bermúdez i Badia, Ulysses Bernardet, Mario Negrello, Markus Knaden, and Paul F.M.J. Verschure. 2005. AnTS: A 3-Dimensional Tracking System for Behavioral Analysis of Flying Insects and Robots. http://doi.org/10.13140/RG.2.1.5008.0806
4. Ulysses Bernardet and Paul F M J Verschure. 2010. iqr: A Tool for the Construction of Multi-level Simulations of Brain and Behaviour. *Neuroinformatics* 8, 2: 113–34. http://doi.org/10.1007/s12021-010-9069-7
5. Philip A. Bernstein. 1996. Middleware: a model for distributed system services. *Communications of the ACM* 39, 2: 86–98. http://doi.org/10.1145/230798.230809
6. Frédéric Bevilacqua, Bruno Zamborlin, Anthony Sypniewski, Norbert Schnell, Fabrice Guédy, and Nicolas Rasamimanana. 2009. Continuous Realtime Gesture Following and Recognition. In *Lecture Notes in Computer Science*, Stefan Kopp and Ipke Wachsmuth (eds.). Springer, Berlin, Heidelberg, 73–84. http://doi.org/10.1007/978-3-642-12553-9_7
7. Chris Branton, Brygg Ullmer, Andre Wiggins, et al. 2013. Toward rapid and iterative development of tangible, collaborative, distributed user interfaces. *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '13*, JUNE: 239. http://doi.org/10.1145/2494603.2480312
8. Steve DiPaola and Caitlin Akai. 2006. Designing an adaptive multimedia interactive to support shared learning experiences. *ACM SIGGRAPH 2006 Educators program on - SIGGRAPH '06*: 14. http://doi.org/10.1145/1179295.1179310
9. Brad Johanson, Armande Fox, and Terry Winograd. 2002. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing 1*, 67–74. http://doi.org/10.1109/MPRV.2002.1012339
10. Stefan Kopp, Brigitte Krenn, Stacy Marsella, et al. Towards a Common Framework for Multimodal Generation : The Behavior Markup Language. 205–217.
11. R. Laban and F. C. Lawrence. 1974. *Effort: Economy of Human Movement*. Macdonald and Evans.
12. R. Laban and L. Ullmann. 1971. *Mastery of Movement*. Macdonald & Evans Ltd.
13. Asa Macwilliams, Christian Sandor, Martin Wagner, Martin Bauer, Gudrun Klinker, and Bernd Bruegge. 2003. Herding Sheep: Live System Development for Distributed Augmented Reality. *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, 123 – 132.
14. Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. *Proceedings of the 24th Annual Symposium on User Interface Software and Technology (UIST'11)*, 315–325. http://doi.org/10.1145/1979742.1979691
15. Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. 2006. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 1. http://doi.org/10.5772/5761
16. R. W. Picard. 2003. Affective computing: challenges. *International Journal of Human-Computer Studies* 59, 1-2: 55–64.
17. Hennadiy Pinus. 2004. Middleware: Past and present a comparison. 1–5. Retrieved February 6, 2014 from http://userpages.umbc.edu/~dgorin1/451/middleware/middleware.pdf
18. Miller Puckette. 1996. Pure Data: another integrated computer music environment. *International Computer Music Conference*, 37–41. Retrieved February 22, 2014 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.3903
19. Daniel Pustka, Manuel Huber, Christian Waechter, et al. 2011. Automatic configuration of pervasive sensor

networks for augmented reality. *IEEE Pervasive Computing* 10, 3: 68–79. http://doi.org/10.1109/MPRV.2010.50

20. Casey Reas and Ben Fry. 2007. *Processing: A Programming Handbook for Visual Designers and Artists*. The MIT Press.
21. Maryam Saberi, Ulysses Bernardet, and Steve DiPaola. 2014. An Architecture for Personality-based, Nonverbal Behavior in Affective Virtual Humanoid Character. *Procedia Computer Science* 41: 204–211. http://doi.org/10.1016/j.procs.2014.11.104
22. Etienne Schneider and F Picioroagă. 2004. Dynamic reconfiguration through OSA+, a real-time middleware. *International Middleware Doctoral Symposium*: 319–323. http://doi.org/10.1145/1028480.1030196
23. Ari Shapiro. 2011. Building a character animation system. *Motion in Games*, 98–109. Retrieved from http://www.springerlink.com/index/L24P125448583571.pdf
24. Emily Singer. 2011. The Measured Life. *MIT Technology Review* July/Augus. Retrieved March 11, 2015 from http://www.technologyreview.com/featuredstory/424390/the-measured-life/
25. Dean Takahashi. 2015. The top 11 tech trends of the Consumer Electronics Show. Retrieved March 11, 2015 from http://venturebeat.com/2015/01/12/the-top-11-tech-trends-of-the-consumer-electronics-show/
26. Andrew D Wilson and Sabrina Golonka. 2013. Embodied Cognition is Not What you Think it is. *Frontiers in psychology* 4, February: 58. http://doi.org/10.3389/fpsyg.2013.00058