The Affinity Platform: Service-Oriented Architecture Based on Abstraction of Connection

Alexandru Ardelean*
aard@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Kuderna-Iulian Benţa*
benta@cs.ubbcluj.ro
Babeş-Bolyai University
Cluj-Napoca, Romania

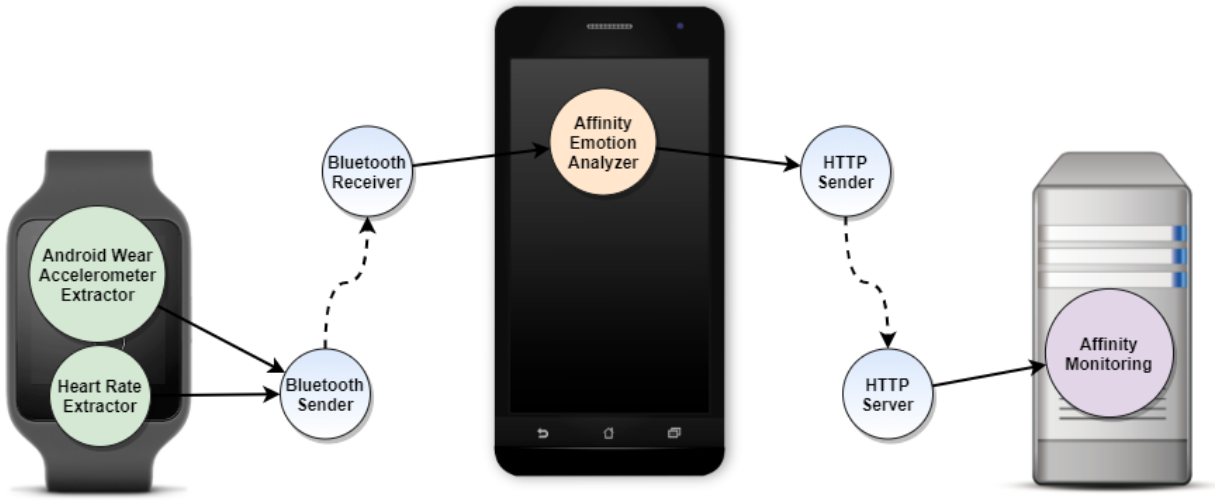

Figure 1: Representation of a distributed software system engineered using the Affinity Platform†

ABSTRACT

The Affinity Platform is an innovative Software as a Service orchestration system designed for building Service-Oriented software solutions. It challenges the modularity of regular software frameworks and microservice architectures by abstracting the connectivity layer between two services.

The main benefit of utilizing this approach is that components designed to communicate through a specific interface can be easily routed to handle this communication over a different medium. A service could extract health data from a smartwatch, use Bluetooth connectivity to send it to a smartphone for pre-processing, and the result can then be transmitted over HTTP to a server for centralization (Figure 1). All this would be possible without requiring services to agree on a specific communication protocol or data format.

The capabilities of the platform were enhanced to assist the development of a multi-modal solution for monitoring patients in a constantly changing environment.

CCS CONCEPTS

• **Computer systems organization** → **Interconnection architectures**; **Peer-to-peer architectures**; • **Software and its engineering** → **Software as a service orchestration system**; **Integrated and visual development environments**; *Search-based software engineering*; *Automatic programming*.

KEYWORDS

Service-Oriented Architecture; Dataflow Programming; Software Platform; Language Agnostic; Visual Development Platform; Automatic Programming

*Both authors contributed equally to this research.
†A git repository with easy to try out examples is available at the following URL: `https://github.com/alexander34ro/affinity-examples`

UbiComp/ISWC '20 Adjunct, September 12–16, 2020, Virtual Event, Mexico

ACM ISBN 978-1-4503-8076-8/20/09…$15.00
https://doi.org/10.1145/3410530.3414371

ACM Reference Format:
Alexandru Ardelean and Kuderna-Iulian Benţa. 2020. The Affinity Platform: Service-Oriented Architecture Based on Abstraction of Connection. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers (UbiComp/ISWC '20 Adjunct), September 12–16, 2020, Virtual Event, Mexico.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3410530.3414371

1 INTRODUCTION

The Affinity Platform is an experiment meant to move forward the idea of constructing software systems by assembling together reusable modules, akin to architecting buildings out of Lego blocks.

The Affinity Platform is a framework for building parallel and distributed software systems composed of independent services [20] organized in the form of data flows and connected through abstract channels of communication. This is similar to the way a message broker would behave, but the Affinity Platform takes it up a notch by introducing multiple brokers for various standards of communication. Not having to implement separate logic to handle the connection with other services means that Affinity components [5] are light, highly reusable, and easily reconfigurable.

1.1 General functionality and reasoning

The Affinity Platform achieves its functionality by capturing the standard input and output of a component and routing them to external services dedicated to sending and receiving data through various interfaces. Services are connected through abstract channels in which additional components can be interpolated to change the medium and the format of communication.

Using the Affinity Platform, developers can construct diagrams of their applications using a visual interface inspired by dataflow programming [17]. This, in turn, enables for facile parallelization of these channels, and a simplified flow through which different services can be easily distributed across a range of hardware devices. Support for any protocol and data format can be added through the Software as a Service [21] component of the platform.

The Affinity Platform can support a plethora of operating systems and hardware devices and can integrate services written in any programming language. The Platform was made possible by recent advancements in the JavaScript programming language [11].

The potential applications of the Affinity Platform are in:

- `Internet of Things`: where services are distributed over various devices with different communication interfaces and operating systems [15].
- `Intelligent Agents`: more specific, multi-agent systems, as they are usually developed using dataflow programming and require strong communication between agents [2].
- `Software Engineering`: since the platform can increase the modularity of software systems, and better encapsulate components and their dependencies [5].

1.2 Identifying common development patterns through dataflow optimization

Abstracting the communication channel between two components brings along with it innovations that were previously much harder to integrate into a software system. Context-aware applications [19] are an emerging field of study concerned with apps that alter their behavior based on factors from the surrounding medium. This can be achieved through the Affinity Platform by interpolating smart components that transmit the information to a different branch of the data flow based on context information. Through smart components and machine learning the Affinity Platform can allow applications to adapt to different scenarios in ways that were never seen before.

The Affinity Platform makes use of dataflow programming and component-based software engineering [6] to facilitate the development process of software systems. A component is abstracted as a data-processing unit with multiple channels for input and output and various configuration options. It only requires the specification of a manifest file that describes the interface of these channels and the possible configuration options. Any software application capable of extracting, processing, or delivering data over a standard output can become a component of the Affinity Platform.

The Affinity Platform can be seen as an additional layer on top of conventional software systems. It doesn't care what language a component was written in (language agnostic), what operating system it runs on (OS agnostic) or what protocol it uses to communicate (protocol agnostic). Offering this level of abstraction means that developers have less to worry about when it comes to the implementation details of their systems. Given a sufficiently large repository of components, Affinity systems could be entirely built from already developed components without requiring hard software development knowledge.

2 RELATED WORK

As a Service-Oriented framework extendable through custom components but with a particular focus on a few key areas, the Affinity Platform can be anchored in multiple fields of study and compared from various points of view.

The first perspective is in terms of its capabilities as a platform, how it compares to other solutions aimed at bringing together components that can be assembled into modular systems.

Other perspectives analyze how the Affinity Platform can produce systems capable of replacing dedicated solutions for specific areas of interest. These include to a certain degree mechanisms for managing and coordinating sensing devices, as well as frameworks for developing intelligent agents and handling communication in multi-agent systems [3].

This paper will discuss only how the Affinity Platform compares to frameworks for managing sensing devices and how it relates to other service-oriented architectures.

2.1 Modular architectures for managing sensing devices

A critical comparison is made to the Sentio framework [10], a middleware for accessing remote sensors that, in its field, achieves functionality more complex than that of the Affinity Platform but in a less modular and user-friendly way. The main capabilities of Sentio include virtualization of sensors and selection and allocating sensors based on contextual information and resource availability.

A significant number of these are functionalities that could be achieved through the Affinity Platform as its repository of components expands with the help of the community. To better understand the differences between the Affinity Platform and Sentio, its main alternatives, RIO [1], Beetle [18], and BraceForce [23], are also included in the comparison from Table 1.

Components that offer support for the Bluetooth Low Energy standard and native sensor selection are currently under development (**–**). Still, this highly-demanded functionality can also be achieved through custom-developed components.

Table 1: Comparison of the Affinity Platform in the field of frameworks for managing sensing devices

	RIO [1]	Beetle [18]	Braceforce [23]	Sentio [10]	This approach [4]
Deployable with unmodified OS	✗	✗	✓	✓	✓
Code offloading	✗	✗	✗	✓	✓
Sensor composition	✗	✗	✗	✓	✓
Sensor selection	✗	✗	✗	✓	–
Support for more than two devices	✗	✗	✓	✓	✓
Concurrent sensor access	✗	✓	✓	✓	✓
BLE peripherals	✗	✓	✗	✗	–
Reconfigurable systems and subsystems	✗	✗	✗	✗	✓
Dynamic sensor switching	✗	✗	✗	✓	✗

Where Affinity systems excel is the partitioning into interconnected subsystems. Individual components can be more easily shared and reused as connectivity with other components is offloaded to the platform. Affinity subsystems can be reconfigured and reconnected to maximize their utility.

2.2 Platforms for software development

In terms of modularity and reusability, the present platform has the same advantages as other plugin architectures, component-based architectures, and service-oriented architectures. The disadvantage that the Affinity Platform tries to address is that of restrictive communication. Web 2.0 [9] was based on rigid APIs and is breaking; the Affinity Platform is trying to solve this problem by abstracting away the communication layer. This philosophy allows applications to delegate the communication responsibility to modular components that can be easily reconfigured to support different protocols and data formats. This approach prolongs the effective life of components and increases their reusability.

The Affinity Platform operates under the Node.js JavaScript runtime [12] and uses terminal emulators such as Termux [22] to port its functionality to a wide range of devices. A custom version of the Chimera framework [14] is used as the underlying mechanism for linking components together.

An in-depth comparison of the systems and tools considered when building the Affinity Platform was previously presented [4].

3 CORE IDEAS

3.1 General

The Affinity Platform is built on the concept of fully independent components. They are considered the building blocks of Affinity-developed software systems but are fully capable of operating individually, not just as part of a parent system. Due to this fact, software systems designed using the Affinity Platform tend to closely follow core software engineering principles such as being low in coupling and high in cohesion.

Engineering and developing distributed software systems from scratch is not the only use case for which the Affinity Platform was designed. Software systems don't have to be created around components from the beginning to make use of the advantages of the Affinity Platform. Since components are independent of the communication layer, developers have the ability to use the platform to extend applications that were not specifically designed to be composed.

More specifically, this is achieved by transforming the application into a black-box component and defining its communication channels in a manifest file. The Affinity platform can then redirect the standard inputs and outputs of the program and connect them to those of other components. This trait enables new functionality to be added even to legacy systems that were not designed with extensibility in mind. The Affinity Platform takes regular software programs and uses them as services, reconnecting their standard communication channels to other components/services.

3.2 Device-agnostic, protocol-agnostic, and format-agnostic

Abstracting the communication layer into separate services means that software systems built with the Affinity Platform are protocol-agnostic and format-agnostic. A system that uses SOAP to connect its services could easily make the transition to HTTP, and from HTTP to a new standard. They are also not limited to a single data format, and, with additional services, can make the transition from XML to JSON or YAML. Moreover, they could even adapt to alternative communication mediums, being able to go from using the Internet to Bluetooth or NFC.

Using independent components also means that the Affinity Platform scores high in terms of compatibility. Any application can be easily integrated into the platform as long as it can be invoked from the command line or is actively listening for data. The manifest file defines a number of input channels and output channels for the component, together with a set of configuration options that can be changed by the user.

The inter-compatibility of input and output channels is not guaranteed as components are not limited by the communication medium or data representation. These are left as choices for the developers, and they are not imposed as a responsibility of the component. Developers can quickly achieve compatibility between different services by defining inputs and outputs that support different data representations or by incorporating additional components to act as translators between common data representations or communication mediums.

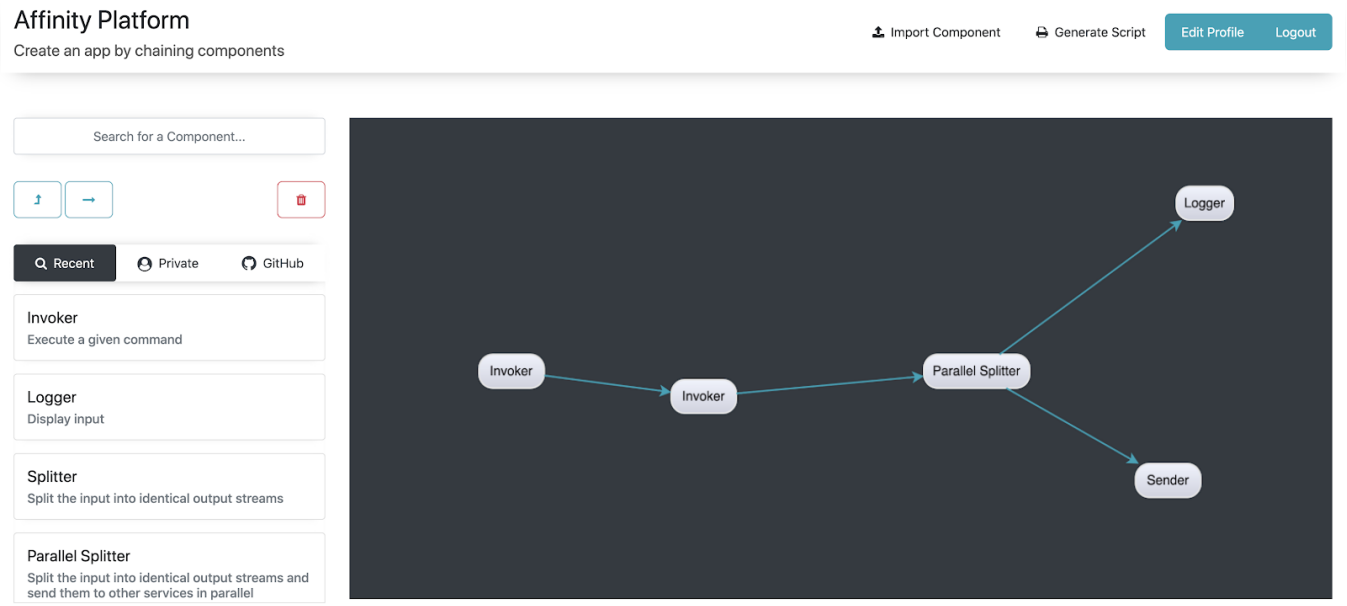

Figure 2: Affinity web user interface, example of building the system from Figure 4

3.3 Dataflow programming and user interface

The way inputs and outputs are connected is described in a CHIML [13] configuration file that is processed using a modified version of the Chimera framework [14]. In order to make writing these files easier, the Affinity Platform employs a graphical user interface (GUI) for generating CHIML configurations out of system diagrams inspired by dataflow programming.

The editor used for designing diagrams is built using the mxGraph [16] JavaScript library, making advanced editing options readily available. The current state of the visual interface used by the Affinity Platform is presented in Figure 2. Developers can edit the auto-generated scripts in case a lower level of control is desired. More details regarding the compilation of diagrams into code are available in previous work [4].

4 RESULTS

The Affinity Platform was envisioned during the development of the main Affinity software application, a system for improving the safety of driving routes in a multi-agent environment using emotion sensing [8]. A simpler way of making the Affinity Emotion Analyzer and its related algorithms available to the public was desired. A high level of reliability and ease of use were also part of the vision, making it possible for other researchers and developers to extend and reuse this work in a way that does not add significant overhead. The Affinity Platform comes as an answer to all these requirements and challenges [7].

Due to using the Affinity Platform, the main components of the Affinity ecosystem are exposed individually. For example, if somebody wants to use just the Affinity Emotion Analyzer, they could easily take its components out of the central system and integrate it into their own system. This, together with other scenarios that were tested, confirm that using the Affinity Platform can significantly improve the reusability of software systems and their components.

4.1 Localized communication between independent components

Figure 3 demonstrates a basic but relatable example of using the Affinity Platform. It consists of three custom components running on the same machine and doing the extraction, pre-processing, and regression parts of an ML algorithm, with the help of built-in Replicator and Logger components.

Custom components for directing the flow of data, like the Replicator, are designed to have minimal impact on performance, using low-level communication standards to pass data. As a result, the Replicator can handle a theoretically unlimited number of output signals. In this example, it is used to send the refined data to both an emotion analyzer and a logger, allowing the direct comparison between extracted features and detected emotions.

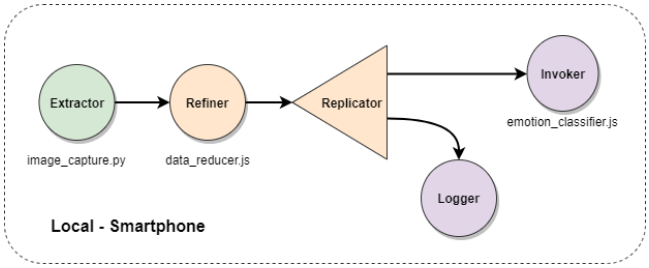

Figure 3: Emotion sensing in a basic setup

4.2 Distributed parallel sensing system running on devices with limited resources

This is a complex but still controlled usage of the Affinity Platform, consisting of a smartphone or a wearable sensing device connected to a server, as represented in Figure 4.

Unlike the previous example, this time, the sensor data extraction phase is happening on a smartphone with limited resources, and the computationally intensive emotion recognition phase is offloaded to a server. This example shows how easy it can be to modify components designed to communicate locally and extend them to work over the network.

Moreover, transmitting the data to a logger is now done in parallel with the help of another component. This is another benefit of using dataflow programming when assembling subsystems. Actions that require multiple streams of data can be trivially parallelized.

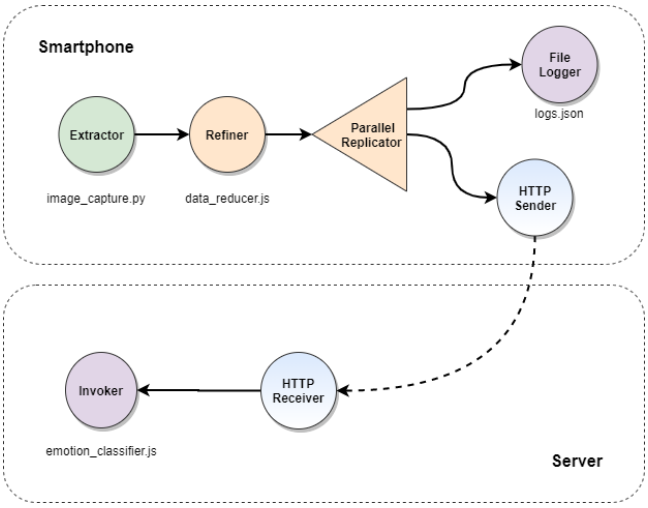

Figure 4: Emotion sensing in a distributed and parallel setup

4.3 Real-life scenario using emotion sensing for monitoring mobile patients

The capabilities of the platform are being showcased during the development of a multi-modal [7] solution for monitoring patients that move through multiple environments during a day. This scenario consists of multiple multi-modal sensing setups that transmit data to a centralization system. The Affinity Platform is being used to tackle challenges like using different sensing devices and adapting the system's structure based on the current environment. To better understand the kind of problems the Affinity Platform was designed to handle, a visual representation of the system design is provided in Figure 5.

Each multi-modal system consists of two data extracting components, one for accelerometer data and another one for heart rate data. They are running on a secondary device (A), a smartwatch. (A) uses Bluetooth to transmit the features it extracted to the primary device (B), a smartphone, where the prevailing emotion of the patient is determined by a machine learning algorithm (the Affinity Emotion Analyzer). The result is sent over via HTTP requests to a remote server (C) for centralization and visualization.

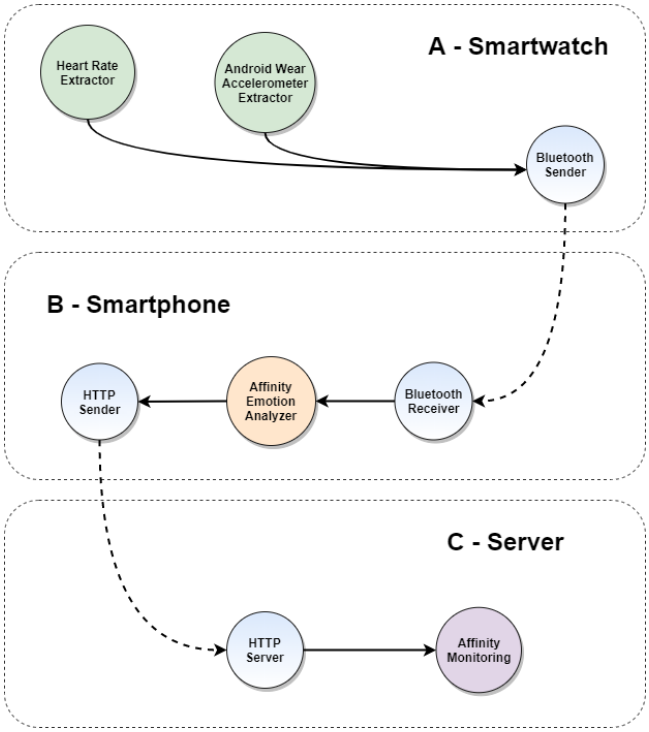

Figure 5: Emotion sensing setup for a challenging distributed multi-modal scenario

4.4 Extending an emotion sensing application developed independently

The Affinity Platform is currently being used to improve the modularity of an already developed facial emotion recognition system (FER). This systems tries to combine traditional emotional recognition methods with advanced machine learning.

In this scenario, the legacy FER software is being minimally altered to allow it to switch between different emotion detection algorithms and multiple machine learning models. This also means that the FER application will be able to easily integrate other algorithms and models in the future.

5 DISCUSSION

The results observed in this experiment are indicative of the potential of the Affinity Platform. Nonetheless, these are only early results and what could be accomplished goes beyond what was exemplified in this experiment. Even though perfectly usable, this tool is more of a proof of concept than a fully-fledged platform. As with any new platform, the number of components available through the Affinity Platform is limited at the moment. There is hope that, with the help of the community and all the effort that went into making it as easy as possible to add new components, the situation will improve in the future.

One of the advantages of the Affinity Platform is its ease of use, not only for regular developers, but also for computer programs. Agents could be programmed to assemble software systems using the Affinity Platform. Artificial Intelligence could be used both inside the system to alter its structure and make smart decisions, but also outside the system to select and connect the primary components.

Stateless components or pure components are another interesting idea evolved during the development of the Affinity Platform. Alike pure functions, they are components that do not depend nor alter the state of the larger system that includes them. Their input consists of only immutable data, and their output is also unchanging. As a result, for a given input, a pure component will always provide the same output. This makes it easier to understand and predict the behavior of a pure component and also to test and debug a system composed of pure components. Together with easier caching for the results, this makes purity an extremely desirable quality for components. This is another good practice emphasized by the Affinity Platform, as most of the proposed components fall in the category of pure components. These include all intermediary components (Refiner, Replicator) marked with orange, and communication components (Sender, Receiver) marked with blue in Figures 3, 4, and 5.

6 CONCLUSION AND FUTURE WORK

This paper discusses the practical and theoretical aspects of using a Software as a Service orchestration system to develop complex sensing setups. It goes over the implementation details of such a system and the benefits observed in challenging real-life scenarios. The ability to reuse parts of the system and the ease of use of the Affinity Platform were both demonstrated in this work.

At this time, the Affinity Platform is built around the concept of independent components. As a better understanding is gained over the capabilities of these components, we start to see a future where smart components can optimize the functionality of the larger system over time. Components with the ability to dynamically reshape the connections of the system that they are part of are an exciting proposition. This development would facilitate the engineering of context-aware sensing setups capable of altering their inner configuration to adapt to their environment [6].

For example, a smart component could monitor the use of a resource-hungry sensor and lower its polling rate when the battery level is getting low. A smart component tracking the battery usage of each sensor of the system could use linear programming algorithms to optimize the quality of the data relative to the power requirements of the system. Similarly, in a multi-modal sensing system, individual sensors could be turned on or off based on their contribution in a specific environment.

REFERENCES

[1] Ardalan Amiri Sani, Kevin Boos, Min Yun, and Lin Zhong. 2013. Rio: A System Solution for Sharing I/O between Mobile Systems. *MobiSys 2014 - Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services* (12 2013). https://doi.org/10.1145/2594368.2594370

[2] Mercedes Amor, Lidia Fuentes, and José M. Troya. 2004. A Component and Aspect-Based Architecture for Rapid Software Agent Development. In *Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, Mexico, November 22-26, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3315)*, Christian Lemaître, Carlos A. Reyes García, and Jesús A. González (Eds.). Springer, 32–42. https://doi.org/10.1007/978-3-540-30498-2_4

[3] Patricia Anthony, Gan Soon, Chin On, Rayner Alfred, and Dickson Lukose. 2014. Agent Architecture: An Overview. *Transactions On Science And Technology* (01 2014), 18–35.

[4] Alexandru Ardelean and Kuderna-Iulian Benţa. 2019. The Affinity Platform: Modular Architecture based on Independent Components (S). 449–452. https://doi.org/10.18293/seke2019-208

[5] Kaur Arvinder and Kulvinder Mann. 2010. Component Selection for Component Based Software Engineering. *International Journal of Computer Applications* 2 (05 2010). https://doi.org/10.5120/604-854

[6] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. 2012. A Survey of Context Data Distribution for Mobile Ubiquitous Systems. *ACM Comput. Surv.* 44, 4, Article 24 (Sept. 2012), 45 pages. https://doi.org/10.1145/2333112.2333119

[7] Kuderna-Iulian Benţa, Marcel Cremene, and Mircea Vaida. 2015. A multimodal affective monitoring tool for mobile learning. https://doi.org/10.1109/RoEduNet.2015.7311824

[8] Kuderna-Iulian Benţa and Mircea Vaida. 2015. Towards Real-Life Facial Expression Recognition Systems. *Advances in Electrical and Computer Engineering* 15 (05 2015), 93–102. https://doi.org/10.4316/aece.2015.02012

[9] Grant Blank and Bianca Reisdorf. 2012. The Participatory Web. *Information* 15 (05 2012). https://doi.org/10.1080/1369118X.2012.665935

[10] H. Debnath, N. Gehani, X. Ding, R. Curtmola, and C. Borcea. 2018. Sentio: Distributed Sensor Virtualization for Mobile Apps. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–9. https://doi.org/10.1109/percom.2018.8444605

[11] ecma. 2015. *ECMAScript Language Specification 6th Edition / June 2015.* Retrieved July 6th, 2020 from http://ecma-international.org/ecma-262/6.0/

[12] OpenJS Foundation. [n.d.]. *Node.js.* Retrieved July 6th, 2020 from https://nodejs.org

[13] G. F. Gunawan. 2019. *Chiml.* Retrieved July 6th, 2020 from https://github.com/goFrendiAsgard/chiml

[14] G. F. Gunawan, M. Amien, and J. F. Palandi. 2017. Chimera — Simple language agnostic framework for stand alone and distributed computing. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*. 1–10.

[15] London Datastore Hendricks Drew. 2015. *The Trouble with the Internet of Things.* Retrieved July 6th, 2020 from https://data.london.gov.uk/blog/the-trouble-with-the-internet-of-things/

[16] JGraph. 2020. *mxGraph.* Retrieved July 6th, 2020 from https://jgraph.github.io/mxgraph

[17] Wesley M.; J.R. Paul Hanna; Richard J. Millar Johnston. 2004. Advances in Dataflow Programming Languages. *Comput. Surveys* 36, 1 (March 2004), 1–34. https://doi.org/10.1145/1013208.1013209

[18] Amit Levy, James Hong, Laurynas Riliskis, Philip Levis, and Keith Winstein. 2016. Beetle: Flexible Communication for Bluetooth Low Energy. In *Proceedings of the 14th International Conference on Mobile Systems, Applications and Services (MobiSys)*.

[19] P. Makris, D. N. Skoutas, and C. Skianis. 2013. A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments' Integration. *IEEE Communications Surveys Tutorials* 15, 1 (2013), 362–386.

[20] Microsoft. 2016. *Chapter 1: Service Oriented Architecture (SOA).* Retrieved July 6th, 2020 from https://web.archive.org/web/20160206132542/https://msdn.microsoft.com/en-us/library/bb833022.aspx

[21] Salesforce. 2018. *What is Software as a Service (SaaS).* Retrieved July 6th, 2020 from https://www.salesforce.com/in/saas/

[22] Termux. [n.d.]. *Termux - Android terminal emulator.* Retrieved July 6th, 2020 from https://termux.com/

[23] Xi Zheng, Dewayne E. Perry, and Christine Julien. 2014. BraceForce: A Middleware to Enable Sensing Integration in Mobile Applications for Novice Programmers. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems* (Hyderabad, India) *(MOBILESoft 2014)*. Association for Computing Machinery, New York, NY, USA, 8–17. https://doi.org/10.1145/2593902.2593907