A cloud-based middleware for multi-modal interaction services and applications

Avenoğlu, Bilgin; Koeman, Vincent J.; Hindriks, Koen V.

published in
Journal of Ambient Intelligence and Smart Environments
2022

DOI (link to publisher)
10.3233/AIS-220161

document version
Publisher's PDF, also known as Version of record

Journal of Ambient Intelligence and Smart Environments 14 (2022) 455–481
DOI 10.3233/AIS-220161
IOS Press

A cloud-based middleware for multi-modal interaction services and applications

Bilgin Avenoğlu [a,*], Vincent J. Koeman [b] and Koen V. Hindriks [b]
[a] *Faculty of Engineering, Ankara University, Ankara, Turkey*
E-mail: bavenoglu@gmail.com
[b] *Faculty of Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands*
E-mails: v.j.koeman@vu.nl, k.v.hindriks@vu.nl

Received 22 April 2022
Accepted 25 October 2022

Abstract. Smart devices, such as smart phones, voice assistants and social robots, provide users with a range of input modalities, e.g., speech, touch, gestures, and vision. In recent years, advancements in processing of these input channels enable more natural interaction (e.g., automated speech, face, and gesture recognition, dialog generation, emotion expression etc.) experiences for users. However, there are several important challenges that need to be addressed to create these user experiences. One challenge is that most smart devices do not have sufficient computing resources to execute the Artificial Intelligence (AI) techniques locally. Another challenge is that users expect responses in near real-time when they interact with these devices. Moreover, users also want to be able to seamlessly switch between devices and services any time and from anywhere and expect personalized and privacy-aware services. To address these challenges, we design and develop a cloud-based middleware (CMI) which helps to develop multi-modal interaction applications and easily integrate applications to AI services. In this middleware, services developed by different producers with different protocols and smart devices with different capabilities and protocols can be integrated easily. In CMI, applications stream data from devices to cloud services for processing and consume the results. It supports data streaming from multiple devices to multiple services (and vice versa). CMI provides an integration framework for decoupling the services and devices and enabling application developers to concentrate on "interaction" instead of AI techniques. We provide simple examples to illustrate the conceptual ideas incorporated in CMI.

Keywords: Software architectures for AI, multi-modal interaction, smart devices, cloud computing, integration framework

1. Introduction

People use a variety of smart devices in their daily lives and most people are familiar these days with laptops, smartphones, tablets, smart TVs, smart speakers, wearable devices, and also to some extent social robots. These smart devices have different types of sensors including camera, microphone, touch screen or touchable surface and display. The common trend in all of these devices is that they support multimodal-multisensor interfaces which have become the dominant device interface [37]. Research has also shown that people prefer using multimodal interfaces over other interfaces [12,24]. Moreover, recent developments on AI techniques (e.g. speech recognition, natural language understanding, face and gesture recognition) have matured on multimodal interfaces and it has become feasible to build multimodal interactive applications. However, users, service providers and application developers have different expectations related to multimodal interaction.

*Corresponding author. E-mail: bavenoglu@gmail.com.

First of all, users use different smart devices with different capabilities in terms of processing, battery, screen size etc. Multimodal interaction requires near or semi-real-time processing of the data generated by sensors of smart devices to be able to produce immediate responses to user input. [13] finds that user satisfaction decreases when response delays are more than about half a second. To process both speech as well as visual data, AI techniques, e.g., automated speech, face, and gesture recognition, need to be applied. These techniques, however, require powerful computing resources to be able to produce instant replies. The resources to do this are typically not available on smart devices, especially not when we would like to apply multiple techniques on a single data stream (e.g., for detecting face, emotion, lip movement, etc. from a video stream at the same time). Several studies [1,26,47], have shown that the resources available on social robot platforms are also not sufficient for the high levels of computation and storage that are required. Besides the resource capabilities, users expect services to be accessible anytime and anywhere on different types of smart devices. Users also want to be able to seamlessly switch between devices and services any time and from anywhere and expect personalized and privacy-aware services. Secondly, service providers support variety of interaction services, from processing single data type such as audio processing for speech-to-text conversion to processing multiple data types, such as audio and video processing for dialog management. It is not easy for service providers to provide programming libraries for all different protocols, data formats and programming languages that the user applications may need. Moreover, it is better for them to be a part of an ecosystem to increase the usage of their services. Lastly, application developers expect to use simple and complex AI services without delving into the details of these AI techniques. They want to integrate services easily and concentrate on the interaction related issues. The integration of various interaction services and making these available on a range of different smart devices is still a major challenge.

To address these challenges and facilitate the development of multimodal interactions with smart devices, we propose a cloud-based middleware. This is an open architecture which enables service providers to easily integrate their services and application developers to easily integrate different types of devices, their sensors and actuators. It allows (re)using of various interaction modalities. Reusing modalities for particular types of data streams has been an important goal of our work in order to decrease development time and to allow developers to focus most of their attention on development of the application itself. To this end, we have tried to abstract as much as possible from device- or sensor-specific details and propose a middleware that supports reuse for multimodal-multisensor interaction. Although developers should have an understanding of multimodal-multisensor interfaces, development with our architecture does not require expert knowledge of the technicalities of these devices, enabling developers to focus more on the design and application development. Because the potential for such applications is in principle unlimited, we believe it is important that this infrastructure is an open architecture that enables developers to engineer applications for the potentially unlimited variation and combination of types of users, interaction styles, and application contexts. An open architecture is also required to be able to integrate the many different AI services that are provided by different companies through the Internet. As these services assume different data types, communication protocols, security requirements, etc., our architecture needs to be able to support a range of different APIs of these services.

In the reminder of the paper, we first introduce a conceptual model that specifies the type of interaction between user, smart device and our cloud-based middleware that we have in mind. Building on this, we then provide a more detailed design of CMI. We also provide a preliminary evaluation of CMI by means of a simple example use case. The main contribution of this work is the design of a cloud-based middleware that enables:

- different parties (end users, smart devices, developers and AI service providers) to be loosely integrated through integration patterns.
- smart devices to stream sensor data and consume AI services for processing these streams, possibly provided by third party service providers;
- developers to engineer applications and composite services for multimodal interaction with smart devices by simply streaming and receiving data without the need to implement an API;
- fast and acceptable response delays for the processing of data streams by AI services (we provide some initial indications of end-to-end delays that support this in the paper).

2. Related work

In this section, we briefly discuss and compare some of the work that has been done to make it easier to engineer multimodal applications. We classify these works as frameworks for multimodal interaction and cloud architectures for smart devices.

The first framework is a standard produced by W3C for Multi-Modal Interaction (MMI). This is a standard proposal for MMI architecture and communications protocol to facilitate the integration of a wide variety of modalities into multimodal applications. According to [17], this standard proposes an open architecture and standard way of application development that supports more natural interaction by means of, for example, speech, gestures, and touch. The architecture consists of modality components (MC) for processing input modalities such as speech, fusion components to fuse input modalities and fission and presentation components to generate a response to a user, and interaction managers that determine the next step in the interaction based on available user, task, and context information. CMI has some commonalities and differences with the W3C standard [17]. CMI uses the black box approach of the W3C standard for services and aims at creating an ecosystem for processing data from different modalities by means of services developed by different companies with different expertise. One of the main differences is the philosophy of the application layer. The Interaction Manager in the W3C standard works as an application layer software and manages the interaction. For example, it can run as a dialog manager between users and devices. However, in CMI, a dialog is an application or an interaction service that can be provided by composite services or the applications on the user devices (or on other servers that are consumed by users' devices). CMI does not directly provide interaction applications and it just works as data stream mediator and leaves the development of interaction applications to application developers. CMI works as mediator between service providers and application developers. Currently, we have not implemented a protocol in CMI that enables communication between services and applications. Life cycle events such as start, stop, resume for requests and responses should be defined in a standard way and W3C standard can also be used in CMI for this purpose.

The second framework is the "Platform for Situated Intelligence" produced by Microsoft. Platform for Situated Intelligence provides AI processing (e.g., acoustic feature extraction, voice activity detection, speech recognition, face tracking, etc.) on the data collected from sensors (e.g., camera, microphone, Kinect, etc.), and visualizing the results (e.g., speech synthesis, avatar rendering, etc.) [6]. The main aim of this platform is to enable quick development of interactive systems in the real world [3]. It is an end-to-end platform which means that it has support for collecting data, extracting features, training the models and deployment of these models. Developers have to use the libraries of the platform to develop interaction applications. For instance, to enable Google ASR (automatic speech recognition) support, a developer has to develop a component with the libraries of the platform. The platform does not provide a service-based architecture but offers instead a component-based approach, where the platform already provides several components. Compared with CMI, Microsoft's platform integrates services as well as applications whereas CMI focuses on service provision and leaves the choice of application development framework to the developer. Besides these, for integrating new services CMI does not require a client library.

The third framework is AM4I. The AM4I framework discussed in [2] focuses on providing a distributed, ubiquitous, loosely coupled solution for developing and deploying multimodal interaction for smart environments. As in our approach, AM4I is aimed at keeping the demands on local computational resources low, the integration of devices simple, offering off-the-shelf technology for, e.g., speech interaction, and seamless interaction across devices. The proposed architecture follows and refines the proposed W3C standard and promotes an application-oriented point of view that is event driven. Key components of the architecture are input, output, and what the authors call generic modalities. Input modalities are related to the data streams from sensors which are the key building blocks in our architecture but they differ as they represent abstracted events in line with the W3C standard [17] rather than streams of sensor data. Generic modalities, which enable complex interaction options such as multimodal affect recognition, are somewhat similar to the composite services that our architecture offers. [20] is an example for such generic modalities for emotion-aware interaction. AM4I also includes passive or implicit modalities. Data such as temperature or user location can be given as interaction data. In CMI, these are not modalities. They are data streams which are processed by services in the cloud or applications located on devices or other servers. Applications may or may not stream these data or they may stream pre-processed information. In AM4I, applications are built on top of so-called interaction managers (IM) that transform events from input modalities to messages for output modalities.

IM includes state machines for determining the interaction event of the user and the total architecture runs with HTTP protocol. IM is similar to the integration framework of CMI, but all of these roles are delegated to services in CMI and CMI supports different protocols.

The fourth framework that shares many of the goals of our work is Social Signal Interpretation (SSI) which aims to "equip machines with tools that are able to recognize and interpret diverse types of social signals carried by voice, gestures, mimics, etc." and "detect and react to user behavior in real-time" [45]. The paper identifies several challenges to achieve this, including how to synchronize and handle raw signal streams from a variety of sensors, how to create robust recogniser for social signals, how to fuse information at different levels (i.e., data, feature, and decision level), and how to process sensor information in pipelines on the fly in real-time. In the SSI framework, everything read from a sensor is transformed into a stream and handled as a continuous flow of samples at a fixed sample rate and size. However, the SSI framework only provides support for specific devices/sensors that have been integrated into the framework. Developers have access to a C++ API and end users can specify recognition pipelines in XML from components available in the framework. These steps are embedded in the SDK of SSI. It is not an open architecture like CMI which clearly separates services, streams, and applications and facilitates the integration of services of different parties into the architecture.

Our proposal differs from the frameworks that we discussed by committing to a service-based middleware that is open and treats (data) streams as first-class citizens. Our middleware facilitates the integration and use of external services by means of a registration service and can automatically route streams by means of a broker without the need for a developer to manually write code to perform such registration tasks. We also focus more on smart devices than the environment and advanced services using AI to improve user experience by offering more natural interaction.

There are also several cloud architectures for smart devices in the literature. Cloud architectures have already been used in cloud robotics, computation and storage offloading, fog and cloud computing, and IoT. In cloud robotics, the main aim has been to transfer compute-intensive tasks such as face recognition and speech-to-text conversion to the cloud [23]. RoboEarth [46] and Rapyuta [26] are two studies which use cloud technologies to execute robotics tasks and share and re-use semantic knowledge between robots. The main aim of these studies has been to enable robots to accomplish tasks such as grasping and path planning and re-use knowledge obtained in previous applications in new unstructured environments.

In the literature, we also find works that proposes service-oriented architectures for robots. Some of these allow robots to expose their services as web services [14,32], also known as Robot as a Service [31]. One of the aims of this work is to make resources available to other clients, such as end users who want to control these robots through a web page. These systems use standard web protocols such as SOAP or REST to integrate these services into the cloud. For example, MyBot [32] makes various services of the Robot Operating System (ROS) available as web services for client applications. Client applications must implement a special ROSLink protocol to control robots remotely. Other work has aimed to enable robots to be controlled by using cloud services, also known as offloading the computation to the cloud [31]. Chaari et al. [11] develop a cloud architecture for robots to offload their computation and storage requirements. They implement a three layered distributed architecture including cloud, communication and robot layers. In Chaari et al.'s architecture, ROS-based robots expose their messages to communication middleware. This middleware includes clustered Apache Kafka for publish-subscribe messaging and Apache Storm for stream processing. Apache Zookeeper is used for coordination within the cluster system. The aim is to provide a transparent, scalable and reliable architecture for robots by using publish-subscribe messaging and stream processing in a clustered architecture. Offloading studies are also developed for offloading the computation and storage load of the Internet of Things (IoT) devices or other smart devices used for smart homes, smart health care, intelligent vehicles etc. [1]. Aazam et al., lists some criteria for determining when to use offloading. Excessive computation or storage needs, latency, load balancing, permanent storage, data management, privacy and security, accessibility and feasibility issues are listed for deciding whether to offload or not.

IoT related studies have proposed architectures for streaming IoT data and processing tasks by using Fog or Edge Computing. Cheng et al. [15] proposes an architecture based on fog and cloud computing named FogFlow. In FogFlow, compute intensive tasks, such as big data analytics, can be executed on the cloud and others can be executed on the edges. In another study, Belli et al. [5] argue that current Big Data architectures are developed for batch processing; they propose a so-called Big Stream Oriented architecture for applications which require low latency

and real-time processing. In their architecture, they use a classic data processing approach which includes dispatching incoming data, processing, storing it, and responding in case of a request, and they propose a listener-based approach which decreases the in-line processing for the dispatching, processing, and notifying steps. Moreover, they use Graph Framework that they developed based on graphs with nodes and edges for processing data flows between nodes in the processing step.

3. Conceptual model and requirements

Our aim is to create a middleware for AI service providers to publish their AI services and for application developers to consume these services to create more interactive applications for end users. According to the model that we use in this middleware, end users use interactive applications on their smart devices or other interactive technology such as social robots to stream data generated by sensors on these devices to the services in the cloud. The services process these data streams and results are returned to the applications which then use these results to provide more natural interaction. The middleware works as a mediator and allow application developers to search, consume, and change services and adjust streaming parameters even at real interaction time. To develop such a middleware, we first introduce a conceptual model to identify relevant requirements of the user from such a middleware. In Fig. 1, we give this conceptual model.

Users have several expectations of their smart devices and interactive technology more generally. In Table 1 we identify several expectations which we derived from the literature that users of interactive technology have. We focus in particular on those expectations that are relevant for the design of our middleware. Moreover, we treat these expectations as generic requirements that we should address in the design. That does not mean that we do not recognize that users can significantly vary in their expectations (based on characteristics such as sex, age and background knowledge [22]) but we believe that the expectations in Table 1 give rise to overall requirements that our middleware should meet.

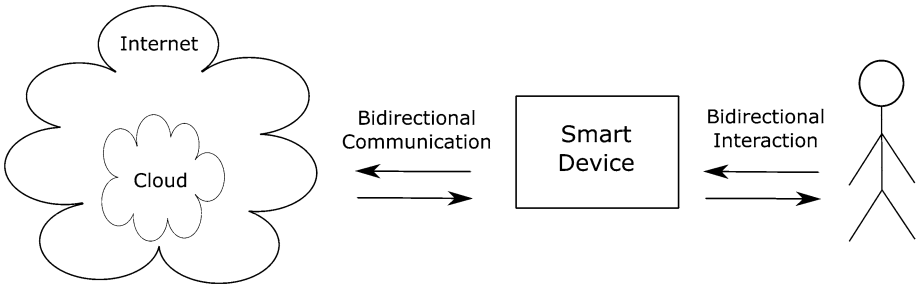

Fig. 1. Conceptual model.

Table 1
Expectations of users of smart devices

A user expects:	
UE1	A fast response to her or his input.
UE2	A personalized experience.
UE3	Her or his privacy to be protected.
UE4	An always available and robust system.
A user expects to be able to:	
UE5	Use different services on her or his device(s).
UE6	Easily switch between and use different types of smart devices.
UE7	Access services anytime and anywhere.
UE8	Easily connect a new smart device.

- **UE1 – A fast response to her or his input**: Users expect *a fast response to their input*. Chen et al. [13] performed a study on response delays with automated speech recognition and found that users are satisfied with response times of less than 431 ms. and still find response delays between 431 ms. and 1.673 ms. acceptable. [44] reports that there is a broad consensus in Human-Computer Interaction (HCI) that a user who has to wait more than a second for a response becomes unhappy and productivity decreases. Satisfactory interaction thus requires semi-real-time responses which ranges from about 100 ms. to 500 ms. and in certain cases up to 1.000 ms.
- **UE2 – A personalized experience**: It is generally recognized that services need to be tailored to various audiences and users expect *services to provide experiences that are adapted or personalized to their needs*. Green et al. [22] report that different users expect different functionalities in their smart homes. Coskun et al. [16] report that early adopters (visionaries) and the early majority (pragmatists) who represent 50% of the consumer market have different expectations of smart household appliances than innovators (technology enthusiasts), the late majority (conservatives), and laggards (skeptics).
- **UE3 – Her or his privacy to be protected**: There are also many studies which show that users expect *their privacy is protected* [10,22,40]. According to [40] privacy is one of the foremost concerns of people in smart homes.
- **UE4 – An always available and robust system**: Users expect *an always available and robust system*. Users' past experiences about computers increase the need for a reliable smart home system [22]. [43] shows that users loose interest when the system is unavailable and quick support is very crucial in case of failures. Moreover, the middleware, services and applications should remain stable over the years and they should not disappear without informing and providing replacements to the users.
- **UE5 – Use different services on her or his device(s)**: Users expect that their *devices are able to provide a range of different services*. [16] reports that people expect to see more services on smart devices to compensate for the price of these devices. Smart devices thus should not only increase user comfort but also provide new services. Moreover, different consumer groups expect different functions to be made available, especially in smart homes [49].
- **UE6 – Easily switch between and use different types of smart devices**: Users do not only expect multiple services but also expect to be able to *use these services on different types of smart devices and easily switch between these devices*. The use of multiple devices allows users to handle more complex tasks [35]. [28] finds that access to the same content on multiple devices is very important to users. The use and seamless integration of multiple devices for handling tasks has been a vision of ubiquitous computing [18,48] for along time. According to Jokela et al. using multiple devices to access content is one of the most commonly requested features [28]. Moreover, Brudy et al. suggest that boundaries between devices should be hidden and ecology of devices should be provided to allow seamless cross-device computing [8]. The satisfaction of this user expectation depends on the interoperability of the services and devices which allow them to work in conjunction.
- **UE7 – Access services anytime and anywhere**: Users expect to be able to *access services anytime and anywhere*, also known as nomadic computing. The vision of nomadic computing is that users should be provided with transparent technologies without location, device, bandwidth, or platform limitations [30]. Using interactive technology anywhere and anytime is one of the aims of ubiquitous computing [41].
- **UE8 – Easily connect a new smart device**: Ease of use has always been important to users. One aspect of this is that users expect to be able to easily *connect their (new) smart devices to services*. [9] find that connecting and integrating different devices is one of the barriers of adoption of home automation systems. Based on this study, Kim et al. propose a method for overcoming this barrier and suggest that plug-and-play and semantic service discovery are necessary for flexibility in smart homes [29]. Newman et al. think that, with the introduction of new devices and technologies in the market, users have to use different applications and user interface styles to connect and control devices and services which produces a problem of "piecemeal interaction" [34].

A user interacts with one or more of her or his *smart devices* by means of the interaction modalities that are made available by the device. A smart device interacts with its users, for example, by means of speech, by showing a video or other content on a display. Today's smart devices support user input by means of natural interaction modalities such as the touching of a screen, the use of voice to ask a question or issue a command, or gaze, gestures, and

body movements. User input is recorded by means of *sensors* which generate data streams of a corresponding type when the sensor is recording. A touch screen produces a stream of touch events, a microphone produces an audio stream, a camera produces a video stream, etc. We therefore view smart devices and more specifically their sensors as producers of *data streams*. A device with multiple sensors can produce multiple data streams. In our work, we target in particular smart devices and interactive technology that provides the means for *bidirectional interaction* between a user and the technology (e.g., multi-modal interaction with a smart device). Both user and technology can interact at the same time. This includes technology such as laptops (which nowadays also have touch screens and interfaces), kiosk systems, and tablets, but also social robots designed for user interaction, and smart devices such as smart displays, glasses, phones, speakers, TVs, and watches. It excludes, for example, a simple Bluetooth speaker which does not have a two-way interaction capability and a smart thermostat which does not require user interaction to control the heater.

As we argued above, a cloud infrastructure can compensate for the limited local hardware resources for computation and storage of most interactive technology. Of course, these resources can still be used to process events which require quick responses but no heavy computation. However, the resources on a robot, for example, are typically insufficient for the heavy computation required by the natural language understanding tasks required for speech-based user interaction. In our model, the *cloud* can provide the resources for handling big data and data persistence and execute the services to provide more interaction with smart devices to users. As our cloud-based middleware is aimed at supporting multi-modal interaction, the services our cloud should be able to support include basic services such as emotion detection, face detection, tracking, and recognition, gaze detection and tracking, gesture recognition, natural language understanding, people detection and tracking, speech recognition, and voice detection and recognition. We also envisage the support of more complex, composite services which are built on top of these basic services such as activity recognition and engagement and user modelling. The cloud should also support basic services that facilitate its operation such as authentication and identity determination, audio and video streaming, and information processing by means of information retrieval. Finally, we assume that the cloud-based middleware can be connected to the *Internet* and thus access available third party services.

The communication between the smart device (or, more precisely, the application) and the cloud-based middleware is also bidirectional. In principle, any network connection type (Bluetooth, Wi-Fi etc.) can be used for connecting the device to the Internet. Data streams generated by sensors on the devices can be send to the cloud in almost any known format that is suitable. Services in the cloud generate event streams that are delivered to the smart device (or application). These event streams can consist of discrete results (e.g., names associated with faces), control commands (e.g., what a speaker or robot should say), or multimedia data (e.g., a video teaching to fix something or an interactive map).

We conclude this section by deriving the main requirements that guided the design of our cloud-based middleware. We have started with identifying several important user expectations (see Table 1) and use these as a starting point to specify our requirements. Each of the user expectations that we identified should be "covered" by at least one requirement on the middleware. These requirements are consistent with the quality attributes of software architectures [38]. In the list we derive, we only select the attributes directly related with the user expectations that we identified. Figure 2 shows how each expectation is matched with at least one requirement.

- **RQ1 – Fast/quick execution of tasks**: Users need quick responses when they interact with their smart devices. Moreover, the tasks such as understanding users' language and face detection need high computation resources to quickly produce result.
- **RQ2 – Fast/quick data streaming and downloading**: Enabling fast/quick execution of tasks will only be possible when the data is sent to remote resources fast enough. Similarly, the results of the tasks should be made available as quickly as possible.
- **RQ3 – Multiple task execution at the same time**: Smart devices have different sensors to collect data. All of these data should be sent and processed at the same time to facilitate and ensure fast multi-modal interaction with users.
- **RQ4 – Multiple and diverse data streaming and downloading**: Users want different types of interaction. This can be enabled by using different sensors on smart devices. Multiple sensors should be able to produce

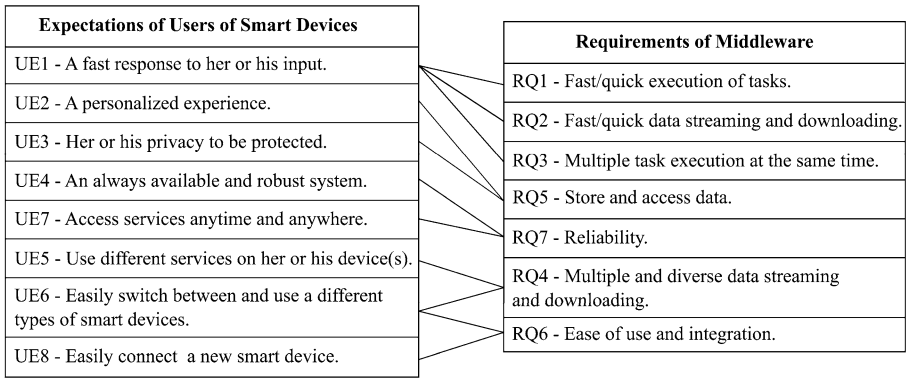

Fig. 2. Deriving requirements for the middleware from user expectations.

and send data at the same time. The data type can be different according to the type of sensors such as, picture, video or audio.

- **RQ5 – Store and access data**: The data collected from sensors and also the processing results should be stored permanently, securely and privately. The middleware should provide necessary security controls to protect the data from getting into the wrong hands. The smart devices and users should be able to access the data whenever they need. The privacy of the data transferring between the smart device and cloud is also preserved.
- **RQ6 – Ease of use and integration**: Smart devices must provide seamless interaction experiences to the users. The new devices of the user must be easily integrated to the middleware. Executing smart devices and services in an integrated manner is also important.
- **RQ7 – Reliability**: The middleware must be available all the time. It must provide consistent speed without any faults. Smart devices can gather data whenever they need and they can restart or resume data gathering in case of a connection problem. The middleware accepts messages in the order they came and correctly identifies the order. The middleware guarantees the delivery of messages to the consumers.

4. Design of a cloud-based middleware

We propose an open architecture to realize our cloud-based middleware with two different layers: a cloud layer and an application layer (see Fig. 3 for a detailed design of the generic architecture). The cloud layer provides stream processing capabilities as services and includes the following components: a core CMI module, a persistent data storage, and cloud services (internal, abstract, and composite) that are all part of the middleware, and external services offered by third parties. All of these services can be provided through distributed servers. *The external services and the application layer are not part of the CMI itself.* External services, such as, e.g., the Google Speech-to-Text API, can be integrated into CMI by registering the service and providing protocol information for data exchange. The open architecture enables developers to integrate external services and to develop applications using the services provided by CMI. CMI facilitates the routing of sensor data streams from an application layer, the processing of these streams by services, and the routing of streams with results from these services back to the application layer.

4.1. Application layer

The application layer consists of the applications developed using the services provided by CMI or other cloud services. Applications can use services to provide their users with more interactive applications with their devices. CMI provides an open architecture for application developers. The basic idea is that applications stream sensor data from smart devices to the cloud layer and consume services that are made available by the cloud layer. *Applications can be developed on smart devices themselves or on other resources (e.g., another cloud server).* If applications

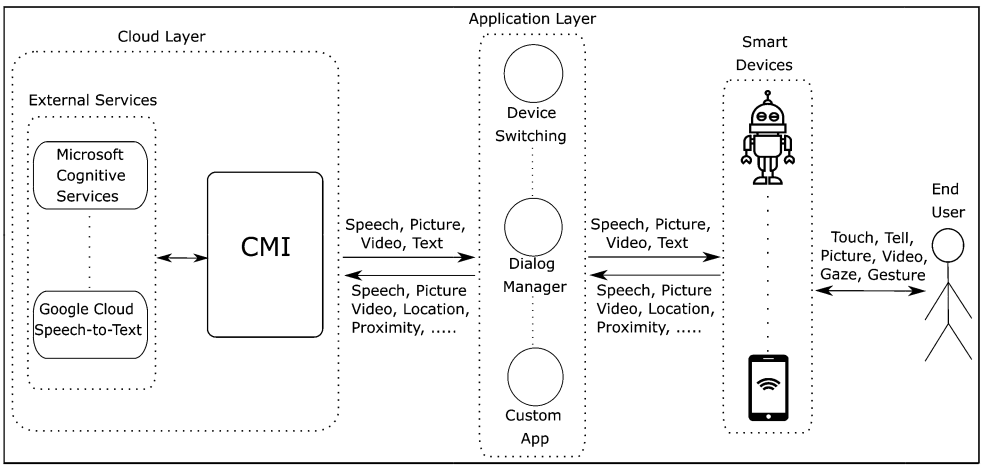

Fig. 3. CMI open architecture.

reside on other servers, devices must at least include a separate small module to stream sensor data and receive results from applications. We also classify these small modules as part of applications. Without the application layer, no device can send/receive data. The application does not have to be an interaction-rich app. It can be a simple app using composite services for interaction features on the cloud. The main tasks of applications are to create data streams that collect the data that users generate when using their smart devices and interactive technology and to receive results from the requested services. Applications enable users to register their devices to CMI (e.g., by calling web services provided by CMI) and enter protocol information of all of the sensors of devices (see a sample prototype interface from Fig. 4 for entering device information of a user). In principle, text, audio and video data types can be sent and received as byte streams by the application layer. Different communication protocols can be used to exchange data. Since we target different smart devices with diverse capabilities ranging from e.g. smart watches to social robots, supporting different types of protocols and data formatting languages provides prevalent access to our middleware. *Applications do not need to implement a client API to access the cloud layer*. Instead they only need to register devices to CMI and indicate which protocols they want to use for sending and receiving data streams of sensors. Applications can send data and receive results in different data format languages since CMI supports major known formatting languages such as XML, JSON and Protobuf. Applications send sensor data as messages in one of these formatting languages to CMI and CMI forwards messages to services by putting necessary identifiers to inform services about the message owner. Applications therefore only need to understand the simple message structure which they receive from services. These messages include identifiers of sensors of the devices of the user, identifiers of the requested services and data or result packet number. There are no restrictions on how streams with the results produced by CMI services can be organized by application developers. *Application developers do not have to implement complex client-side code, which is a key feature of our cloud-based middleware and makes it easier for developers to create applications*.

4.2. Cloud layer

The cloud layer consists of external services provided by third parties and by CMI. CMI provides various services to enable the development of applications for interactive technology. Figure 5 shows these services and modules of CMI core including “Registration”, “Connection”, “Integration Framework” and an optional “Streaming” module.

4.2.1. Basic, abstract and composite services

A CMI service assumes that a data stream is provided by an application layer. CMI services process these streams and returns a stream with results back to the application layer. We distinguish *basic, abstract, and composite* services. A *basic service* provides a basic capability such as emotion, face, or gesture recognition in a video stream. Some basic services (e.g. emotion recognition [4]) that are part of CMI have been built using existing open source

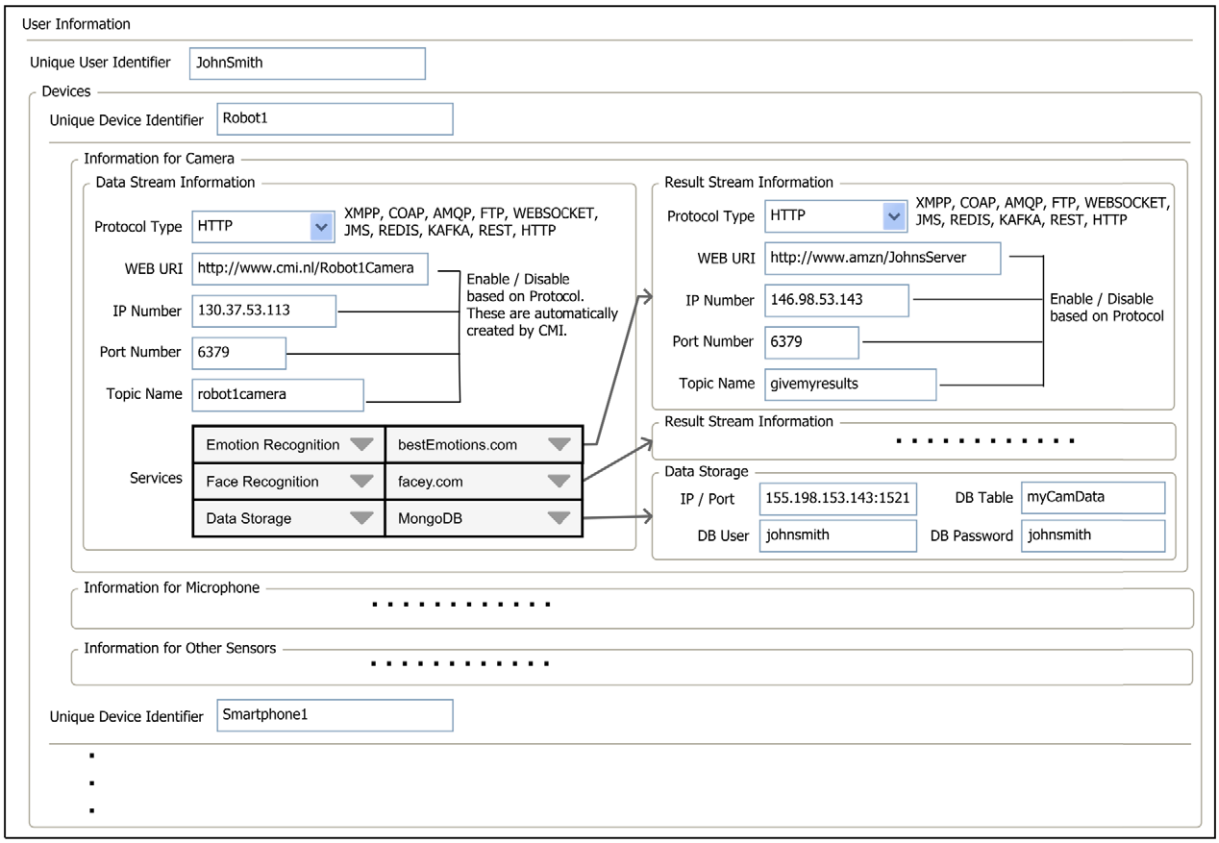

Fig. 4. Settings for the devices of a user (user defines her devices and the sensors of her devices. Each sensor has a data sending and result receiving endpoint. User can select many different services for streaming data registered beforehand to CMI.)

code. An *abstract service* can be used to create an 'interface' for basic and external services. Abstract services hide the implementation details of a service and can be used to provide several different implementations of a type of service to the application layer. This allows applications to switch to another service implementation without the need to change any application code. Switching can be automatic based on some situations such as temporary shutdown of a service or manual by users according to personal choices such as service satisfaction. Finally, *a composite service* is a service composed of two or more basic or abstract services. Such a service, for example, fuses the results of various services that process speech and visual modalities to detect emotion. Other examples of composite services (see Fig. 5) are services that return the level of engagement of a user or return the current state derived from the continuous modelling of a user's state. The benefit of a composite service in CMI is that it knows which (registered) device(s) is(are) used for, e.g., engagement detection, and the service can take this into account to adapt the service dynamically (e.g., using only a single modality to compute engagement when a smart speaker is used but using both speech and visual modalities when a social robot is interacting with a user).

4.2.2. External services

External services are the services provided by third party companies, organizations or developers. There are two ways of integrating an external service provided by a third party into CMI. First, an external service is registered with CMI. This can be done through a web site or web service interface. An external service must give details about the protocols for receiving data and sending results (see a sample prototype interface from Fig. 6 for entering service information of a service provider). Second, an external service can be integrated into CMI by creating a corresponding abstract service that is part of CMI (see also previous section). Moreover, some AI services may require the implementation of a client library or API for accessing the service. This is not easy or sometimes impossible to

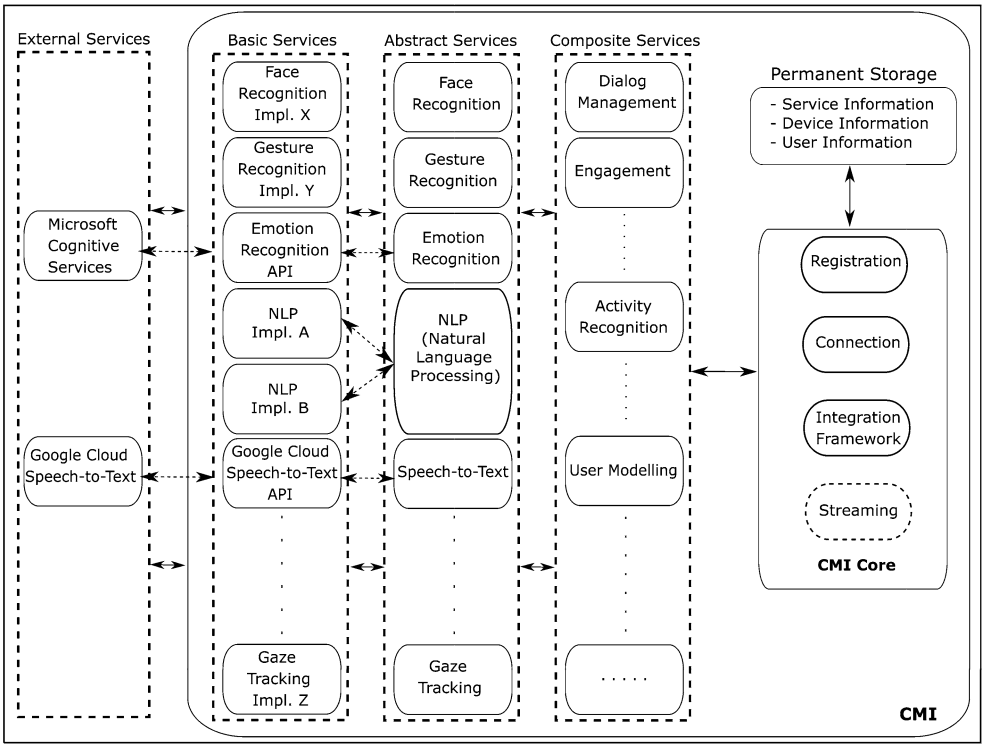

Fig. 5. CMI architecture.

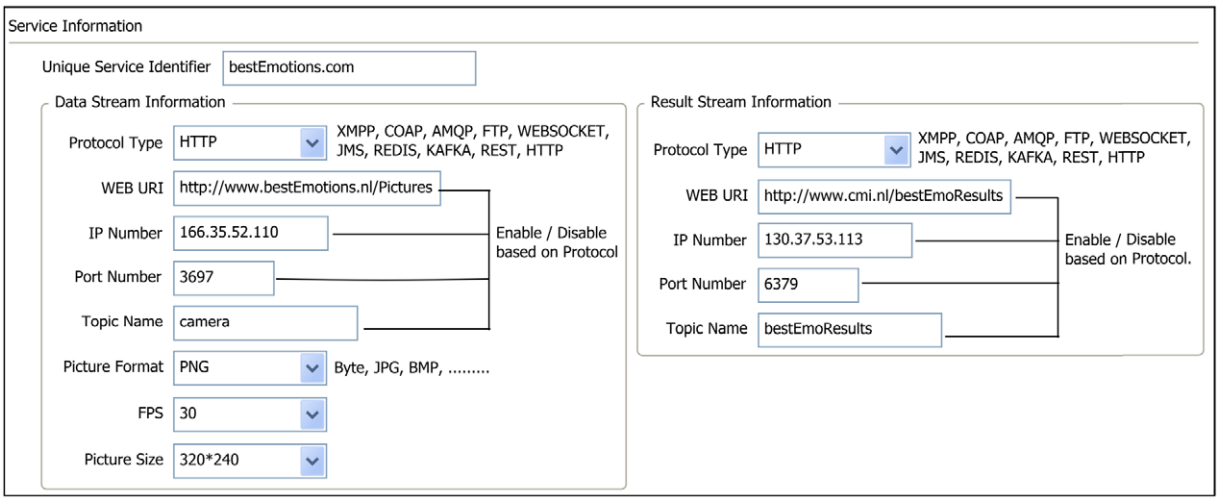

Fig. 6. Settings for an external service (service provider defines its services. Each service has a data sending and result receiving endpoint. After registering, end user applications can search and select this service.

implement in some architectures, especially if the service does not provide client libraries for different programming languages or operating systems. To solve this problem and make the life of application programmers easier, a basic service or another external service by other third party developers can be developed to enable application developers to send and receive data without using an API. This service handles the requirements of the service API on behalf of the application developers (see also Section 5 for example implementation).

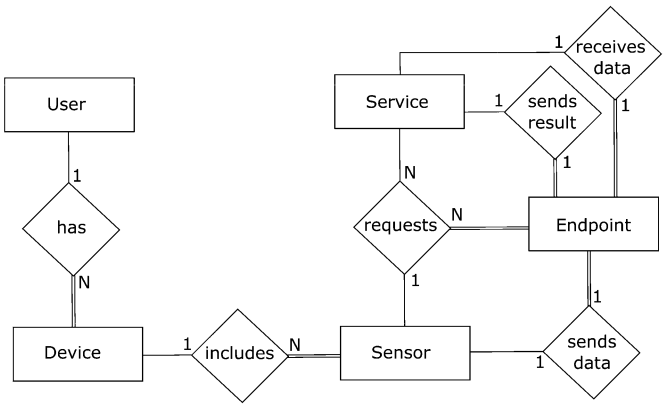

Fig. 7. CMI entity-relationship diagram for sensors, devices, users and services.

4.2.3. Permanent storage

Information about services, devices and their sensors, and users are stored in a permanent storage that is part of CMI. The requested services for each sensor are also stored permanently and they are allowed to be changed. Figure 7 shows the database entity-relationship (ER) diagram. The following statements also describe the structure of the database:

- A user may have several devices (Laptop, smartphone, robot etc.).
- A device may have several sensors (Two cameras, three microphones etc.)
- A sensor may request several services (A camera of a device can request different services. (e.g. face recognition and emotion recognition))
- A sensor may request same services from different service providers (e.g. two 'Face Recognition' services provided by different service providers).
- Results of all services requested by a sensor may be sent to the same endpoint or different endpoints of a sensor.

Permanent storage can also be used for storing information about users that are being serviced at runtime. For example, photos of users for face recognition and sample audio files for voice recognition can be stored. Besides these, user preferences can be stored. These preferences may include user preferences such as data types, e.g., always use JPEG pictures for services to decrease incoming/outgoing data size, device requests, e.g. always send the results of text-to-speech service to the same speaker etc. Stored data can be used for offering personalized services to users. A basic service can be developed to open user data to other services (see the example case in Section 5). Personalization can also be provided by applications and services developed for this purpose and these applications and services can access to permanent storage.

4.2.4. CMI core module

The CMI core module provides three essential services and one optional service. The essential services include a registration service, a connection service, and an integration framework. The optional service is a streaming service. The streaming service adds some capabilities on demand such as asynchronous messaging or storing data in a fault-tolerant and durable way.

Registration service Services, users, and their devices need to be registered with CMI before they can be used or get access to the services that CMI offers. Registration information is stored in a permanent storage that is part of CMI.

CMI stores unique identifiers of users, which can include images of their faces and sample voice messages for identification. Images and voice messages are used as parameters for face and voice recognition services. Users can register all their devices that have sensors. Each device and sensor are also stored with unique identifiers. Protocol information for all sensors of the devices of users must be entered. Applications or the CMI web interface can be used for entering this information. Applications may automatically detect these protocols from devices. Sensors are

registered with their sending and receiving protocol information. Sensor data can be sent by different protocols. Different protocols can also be used for data sending and result receiving. Information is stored according to the protocol used for the sensor. For example, if the HTTP protocol is used for getting the results of a service for a sensor, the URL information is stored. However, if the application of the user prefers using a streaming software for the sensor, IP address, port number and topic name are stored.

Services are registered to CMI and all services have unique identifiers. Services may include some additional data dependent on the type of service they provide. E.g. type, size, and resolution of pictures for processing. Sending and receiving protocol information are also stored similarly with sensors.

Connection service CMI has a web address for connecting services and devices by a post message. Services must execute a connection service with their service unique identifiers. The connection service creates the necessary routing components inside the integration framework of CMI for the service. Services have the option of ending the connection which releases the routing components for the service. The connection service can be executed by services or the operators of CMI on behalf of services. Moreover, different connection options, e.g. calling a web service, can be provided. The connection service also indicates that the service is active and available.

Users connect their devices to CMI after signing into CMI by their user unique identifiers. Users use their device unique identifiers for connecting them to CMI. Users execute the connection service for each of their devices. They can use applications, e.g. an Android application, provided by CMI or developed by third parties to make this operation easier. When a user connects her device, CMI creates routes inside the integration framework of CMI for each requested service of each sensor of this device. Beside routes, CMI may create some software components inside the integration framework of CMI dependent on the protocol that the device uses. For example, if a device prefers sending data through HTTP for one of its sensors, CMI creates a web server component on itself for this sensor. A route is then created from this component to the requested service. If the application of user prefers to use a streaming server such as Redis or Apache Kafka for a sensor of a device, two topics are created per sensor in streaming server. User's device uses one of these topics to send data and another topic to receive results. Routes are created between the requested service's sending and receiving endpoints and these topics. CMI has the option of providing different streaming servers. Both services and devices can also use their own streaming servers. CMI, is not bound to any particular streaming server. CMI can provide different streaming servers which Apache Camel has component support.

Integration framework The main support that CMI offers is to facilitate a variety of smart devices to connect to and use a variety of services for enabling application developers to offer natural interaction capabilities to their users. Smart devices provide different capabilities and may use a range of different communication protocols. Similarly, services may use different protocols and may expect data streams to be of a variety of particular formats. To facilitate this, CMI provides an integration framework to enable the communication between smart devices and services even if they use different protocols or data formats. Integration frameworks are used to consolidate data and synchronize different systems [36]. We use Apache Camel as the integration framework of choice for CMI. Apache Camel integrates almost any kind of system and allows developers to process and send data between systems [27]. It routes data coming from one source to one or many sources by using a route-engine. It also allows different filtering, aggregation and mediation capabilities also known as integration patterns [25]. Data transformation between different formatting languages are also possible. A route in Apache Camel is a component which takes data from the input and based on some conditions, sends it to one or more outputs.

In CMI, we use two concepts with the integration framework: endpoint and route. Endpoint is the receiving and sending points of sensors of devices and services. An application has an endpoint for sending data and another endpoint for receiving results for each sensor. This is similar for a service and it has an endpoint for receiving data and another endpoint for sending results. The endpoint is dependent on the protocol that the sensor and service are used. A service or sensor can use the same or different protocols for sending and receiving. They can use protocols such as HTTP, RESTful, WebSocket, TCP, FTP, XMPP, RESP, etc. We support all the protocols for which Apache Camel has component support. If an application uses HTTP protocol for sending data of a sensor, the endpoint URI is created on CMI with a small web server and necessary parameters. The application sends data streams as HTTP POST messages to this URI. If an application managing a sensor prefers using a messaging system such as Apache ActiveMQ, IP number and port number of the message broker represents the endpoint. Receiving and sending

endpoint protocols can be different for both sensors and services. In CMI, a route is used to link endpoints and it can be created for two different purposes. Routes are used to forward data from devices to services and to forward results from services to client applications. *First, a route is created for each sensor of a device of a user to send data to services.* When the device is connected, through client applications, a route is created for each of its sensors. The route gets data from the sensor and forwards the data to one or more receiving endpoints of the services based on the requests for the sensor of the device of the user. We give unique identifiers to this type of routes by combining unique identifiers of users, their devices and sensors of these devices. Only one route is created for a sensor to send data. We can ensure this way that data needs to be sent only once to several services. A device can have several routes to send data according to its total number of sensors. A client application managing the device can get many service results. These results can come from the services requested by its sensors or they come from the requested services of the sensors of the devices of other users. A user can select receiving users by applications which define other users' receiving endpoints as receiving endpoints of her device sensors. The number of receiving addresses (endpoints) depends on the user or the application that the user uses. The user may prefer creating endpoints (result receiving addresses) as much as the total number of sensors she has or she may prefer only one endpoint (result receiving address). If there is only one endpoint, the application that the user uses can distinguish the results from incoming messages. Incoming message includes the identifiers of the requested service and sensor of the device. *Second, a route is created for each service to send results.* The results of all users are delivered to this route and they are distributed to receiving endpoints of users. A result of sensor of the device of the user may be sent to many receiving endpoints of different users. We give unique identifiers to this type of routes by using service identifiers. This type of route is created when the first device requested this service is connected to CMI. This means that if there is no request for the service, it does not have a route for sending results. However, if a request exists and another device requesting this service is connected, the route is stopped, modified by adding this new request and started again. By this way, we only create one route per service result and separate the devices and services to enable loose coupling between these. At runtime, many other requests can come to this service from different users.

Streaming We use an integration framework in CMI to forward data and results between different devices and services and to enable devices to change requested services at runtime. This enables loosely coupled integration of devices and services. However, for interaction applications, users may need other capabilities. First, data sending and result receiving must be fast enough to support fluent interaction. Second, a sensor's data may be used by several services at the same time which indicates parallelism. Third, sensor data should be persistent for a specific time period to allow consumers to apply a rewind operation in case of a failure. Besides this, if an application developer wants to provide a dialog system, she may need to store data persistently for historical reasons [7]. Such requirements call for special technologies known as "Streaming". One type of usage of streaming is related with "video" (and audio) streaming in web sites such as YouTube and Netflix. RTSP and HTTP progressive downloading mechanisms and their variations are used for this purpose. The main aim of this type of streaming is to provide a fluent experience to end users which have different devices and different bit rates [39]. These protocols allow fast-forward, seek/play and rewind operations on video streams. This type of streaming also provides opportunities for CMI. In CMI, mostly, data are streamed from sensors of smart devices. Services can also stream results after processing the data coming from the sensors. Video, audio and text data are streamed to the services for applying AI operations on the fly. To provide continuous interaction, unbounded video, audio and text streams should be processed while the data is still being streamed. The use of streaming is optional in CMI. If a device or service has the capability of sending and receiving data to a streaming software, they can use this service. Moreover, if an application or service already has a streaming software on themselves, they can continue to use this but still can be integrated into CMI. This also means that, if one side of the communication uses its own streaming software and the other side does not use it, they can still send and receive data through CMI. For example, an application on a user device, which uses HTTP for communication, can send data to a service which has its own streaming software without using the streaming software provided by CMI.

To understand the maximum amount of data that a regular smart device can send and receive and determine whether streaming software are suitable, we collect some information from the literature. We determine the maximum amount of data that can be streamed in a second from some of the smart devices. NAOqi and Pepper robots can send audio data up to 3 MB and support a sampling rate of up to 48 kHz. This sampling rate is more than enough

Table 2
Data sizes of smart devices for audio and video streaming, (for NAOqi 2.8 see [33], for Pepper see [33], for Samsung Galaxy S8 see [42])

Smart Device	Audio				Video			
	Rate (kHz)	Bits	Channels	Total	Size	Fps	Color	Total
NAOqi 2.8	48	16	4	∼3 MB	2560*1920	1	1 B	∼5 MB
					640*480	30	1 B	∼9.2 MB
					1280*960	30	1 B	∼37 MB
Pepper	same with NAOqi 2.8				2D – 1280*960	30	1 B	∼37 MB
					Stereo – 2560x720	15		∼27.7 MB
Samsung Galaxy S8	384	32	1	∼12.3 MB	4K (3840*2160)	30	1 B	∼249 MB
					1080p HD (1920*1080)	60	1 B	∼125 MB

as sampling rates above 16 kHz do not contribute much to speech recognition [21]. NAOqi and Pepper can also send video data up to 37 MB per second. Additionally, we examined the Samsung Galaxy S8 which includes up to date technologies for audio and video. According to specifications, Galaxy S8 can produce 12.3 MB audio data and 250 MB video data per second. Even though there is no need to send 250 MB data for operations such as face recognition, we want to determine approximate maximum values for selecting the best streaming software. A study [19] for comparing data streaming frameworks shows that, Redis and RabbitMQ software are very successful for data sizes more than 1 MB and frequencies from 30 Hz to 100 Hz. These results and the values in Table 2 show that Redis and RabbitMQ are more appropriate solutions than the other frameworks (Apache Kafka and NATS) given in [19] for high-speed data delivery which is very important for interaction. We tested CMI with Redis to show how the architecture works with a streaming software (see Section 5 for details).

5. Case-based evaluation

In this section, we illustrate some of the details of how CMI middleware operates by means of two example use cases. In the first use case, we show how users and their devices are registered to CMI, start using some services and how they can use the results of services. In this use case, a user will say something to her smartphone and the text conversion of this message will be sent to her friends. In the second use case we provide additional statistics about the inline execution of the middleware.

5.1. Evaluation of CMI from the user and service perspective

In this first example use case, we have one service and three devices:

- **Service**: We use Google speech-to-text external service through a basic CMI service as for converting the speech of a user to text. Google speech-to-text service requires to use a client API. This means that, if the user wants to use the Google speech-to-text service, an application which uses client API is needed. In CMI, we want to allow user applications to just send and receive data. Because of this, we provide a special basic service to users. This service, only gets the user data and the unique identifier of the user determined by CMI. Users only need a Google account. In this example our basic service uses Redis streaming for receiving data and sending results back. The basic service runs a thread for each user request. All the user requests come to Redis data receiving topic of the basic service. The basic service sends these requests to Google speech-to-text service in different threads and when it gets the results, it sends these results to its Redis result sending topic. The distribution of results to applications are handled by Apache Camel route created for the service when the first device requested this service is connected to CMI. If another device requests this basic service for its sensor, the route is updated and the receiving endpoints of the device's sensor are included to the route.
- **Devices**: We assume that there are three users with smartphones and these smartphones have a mobile application which send data to and get results from CMI. As we described before, applications do not have to be on

smartphones. But, in the end, a smart device must have a small app to send data. Results can be received by apps located anywhere. The first user, the sender, sends microphone data through HTTP and a request is made to the Google speech-to-text service. This information is entered to CMI permanent storage when the device is registered through application. The first user selects the result receiving endpoint addresses of other users as result receivers of the service that she is requested by using the mobile application. We assume that these addresses are provided through a mobile application. A second user's application has a web server and receives results through HTTP. Finally, a third user's application uses Redis for receiving results.

We now detail the step-by step execution of the use case. The first seven steps are the registration and connection steps for services and devices. These steps are prerequisite for sending and getting data for services and applications. Step-by-step execution:

1. Basic service with Google speech-to-text API support is registered. Service indicates that it uses Redis for getting data and sending results. It also wants to use Redis server provided by CMI. CMI's Redis server IP address and port number are assigned to the endpoints of the service.
2. Basic service is connected to CMI. Because the basic service wants to use Redis server of CMI, the connection module in CMI Core creates two topics in Redis: The first one, "GoogleSpeechToTextRcv", for getting data and the second one, "GoogleSpeechToTextSnd", for sending results. We use the service unique identifier for determining topic names.
3. Users register their devices to CMI. We assume that all the users created their user accounts before. The second user indicates through her application that she wants to use a streaming application for getting results. Because the application has the CMI Redis server settings, a topic name "User2SmartPhone1Display1Rcv" is established by CMI. This information in permanent storage is updated with IP number and port number of CMI's Redis server and topic name. We omit data sending information for this user's device for simplicity since we don't use data sending capabilities of these sensors for this simple example case.
4. A third user registers her device to CMI. The third user's device has the capability of using HTTP for getting results. This information may be automatically entered by the application that the user uses. Since, the user device uses HTTP, a valid web address needs to be provided. CMI creates a web server for only data sending, not for receiving results. For this reason, we assume that her application has the capability to create a small web server and provided "http://user3smartphone/results" as the web address for listening results.
5. The first user registers her device and its sensors thorough her application. She registers her smartphone's microphone and indicates that microphone data will be sent by using HTTP by the application. She also selects the basic service which provides Google speech-to-text API support as requested service for microphone data. For receiving results, she selects their friends through the application and the application selects second and third users result receiving addresses which are recorded to permanent storage at previous steps. The application should authorize the user to see the receiving addresses of her friends.
6. The second and third users use their application to connect their devices to CMI. Because they don't request any service for data sending, no route is created for this purpose for them.
7. First user connects her device to CMI. Microphone sensor uses HTTP for sending data and its requested basic service uses Redis for receiving data. Because of this, first, a web server is created for microphone sensor and a route is created from this web component to the receiving topic of the basic service. Microphone sensor has two result receiving endpoints. First, a route is created from result sending endpoint of the basic service to receiving endpoint of the second user. This means that a route is created from basic service's Redis topic to Redis topic of second user's display sensor. Second, already created route is updated by adding new receiving endpoint. A new HTTP endpoint for the third user's application is added. Hence, the route gets results from basic service's Redis topic and transfers the results to Redis topic of the second user's display sensor and third user's HTTP address on the application. The application of the second user should be subscribed to the Redis topic of its display sensor to get results.
8. The application of the first user starts sending microphone data. Data is sent as byte streams to the web server created by CMI. The route gets the data from web server and make a package by including a unique identifier representing the sensor of the device of the user and microphone data. This package structure is constructed by Google Protocol Buffers serialization language. The package is sent to the basic service's Redis topic.

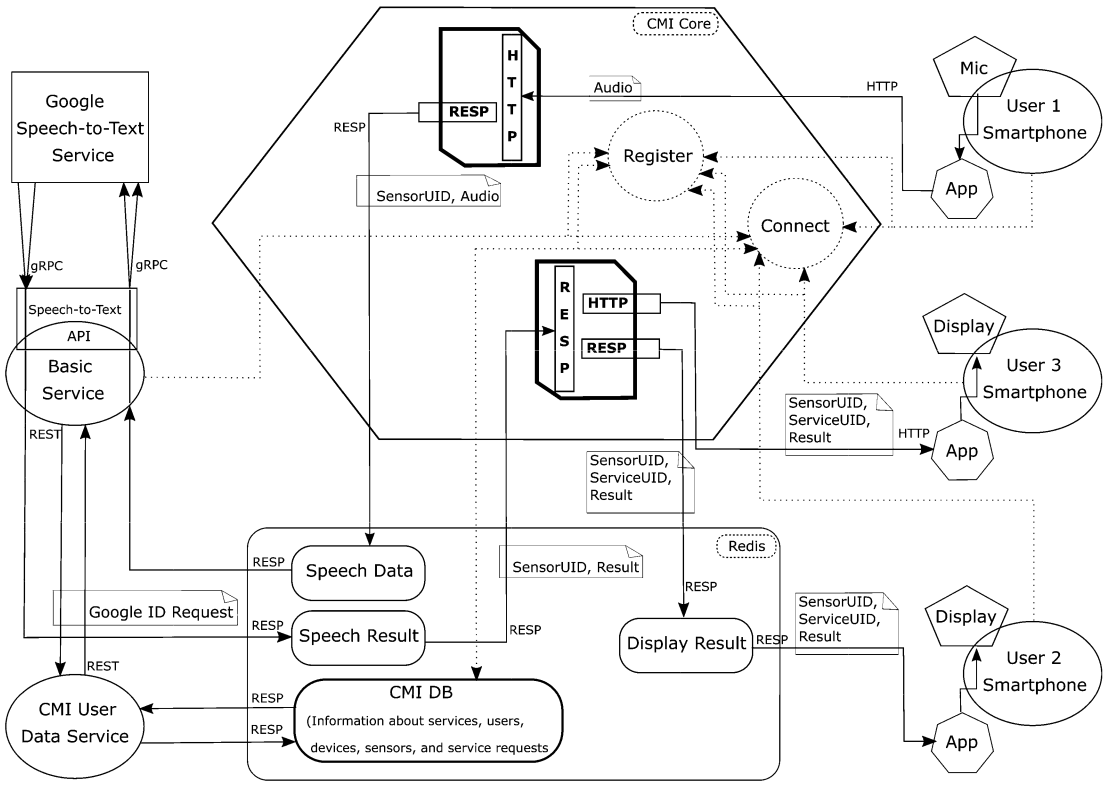

Fig. 8. Information flow of the use case in CMI (dotted lines and circles show the registration and connection operations for devices and services. These operations are prerequisite to send/receive data and results. Basic service can get data from different smart devices, stream them to Google speech-to-text service with their identifiers and collects all results from the same address. It distinguishes them by their Google account identifiers.)

9. The basic service listens to its Redis topic and gets the package. It parses the package, requests the user's Google account information from CMI persistent storage (indicated as "CMI User Data Service" in Fig. 8) by using a unique identifier. We have used Redis for storing all of user data permanently. In this example we use a Redisson Java client for accessing the data at Redis. All data is stored as live objects, a capability provided by Redisson. Then, the basic service creates a connection to Google speech-to-text service within a new thread and starts sending data. The basic service uses Google speech-to-text client API to create connection to Google's service. It uses streaming API of Google which works with gRPC protocol.
10. The basic service thread gets results from Google speech-to-text. It creates a package with unique identifier and the text result and sends it to service's result sending Redis topic. The thread also uses protocol buffers to serialize the package. The route created for this service gets the result and deserializes the package. The route component creates a package including the result text, unique identifier for the sensor of the device of the user and service's unique identifier. The route component, sends the package to second user's Redis topic of the display by RESP protocol and third user web address by HTTP POST message.

Finally, as we discussed, response time in interaction affects user satisfaction. Some studies [13,44] show that response times below one second is acceptable by the users. In the study of Chen et al. [13] users are satisfied with automatic speech recognition responses which are below 431 ms. and they found acceptable when the response time is below 1.673 ms. To determine whether CMI enables responses within the limits of acceptable times we conduct a measurement test. In this test, we use the example that we describe above and measure the time between the applications of User 1 and User 2. We implement the test by sending 500 messages from User 1's device to the Google's Speech-to-Text service through the basic service. We create an almost 1,6 seconds ".wav" file including only one word which is "hello" and stream this file by using the code provided by Google for transcribing audio from

Table 3
Statistics related with the measurements

	Speech-to-text processing time (ms.)	Architecture overhead (end-to-end – speech-to-text) (ms.)	End-to-end delay (ms.)
Avg.	475	181	656
Std. Dev.	229.34	97.84	267.65

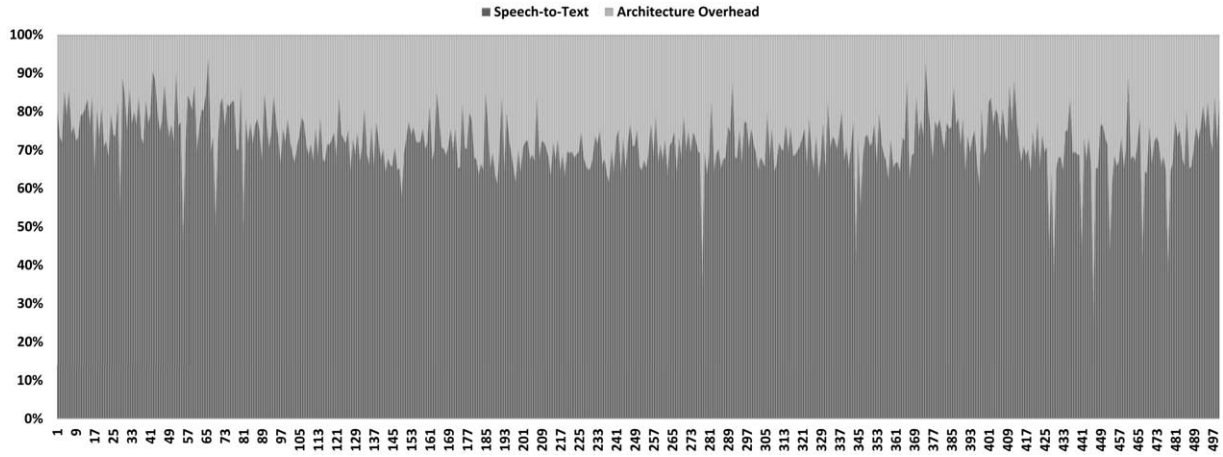

Fig. 9. Graph of messages sent for speech-to-text service.

streaming input. We record elapsed time for end-to-end and for Google speech-to-text service. For simplicity, we put these applications on the same computer for time synchronization. The average duration of these 500 messages is 656 ms for sending the file from the device and receiving the transcription by the application on the other device. On average, 475 ms. are used by the speech-to-text service. Only 8 of these messages are returned above 1.673 ms. 461 messages are below 1.000 ms. and 51 of them are below 431 ms. Table 3 shows these statistics. Moreover, Fig. 9 includes the distribution of these messages. This figure, is a %100 stacked area chart. Dark grey bars represents the time for speech processing. Light grey bars represents the time used by CMI. According to these results, on average %71.93 of the time is consumed by speech processing. We can say that the average end-to-end duration is acceptable. However, the performance of the AI service is very important to get acceptable results. In our case, Google Speech-to-Text AI service performs well for developing interactive applications when we compare the results with the acceptable response times found in the studies that we discussed. AI service's processing time and CMI's overhead should be lower than the indicated limits. We should indicate that this measurement only includes specific data type, audio data. The results may be different for other data types, e.g., video. However, the main aim of this part is to show the architecture overhead.

5.2. Evaluation of the performance of CMI

In the second example case, we show the performance of different parts of the middleware. This example case includes a user device and a service. We assume that the user device, such as a humanoid robot, sends video data from its camera to a face detection and emotion classification service. We use the source code for emotion classification that we get from GitHub[1] and it is written in Python and uses OpenCV. We modify the code so that it gets a video frame in ".png" picture format as a byte stream by using RESP protocol and send the string result back by using HTTP protocol. Instead of sending a video stream, for simplicity, we send one picture 1.000 times sequentially and measure the times from different points of the architecture. We use two different pictures with two different emotion

[1] https://github.com/oarriaga/face_classification

Table 4
Properties of test servers

	Server 1	Server 2	Server 3
Installed Component	CMI Core	Redis	Service and Device Software
Operating System	Windows Server 2019 Standard 64 Bit	Windows Server 2016 Datacenter 64 Bit	Windows Server 2016 Standard 64 Bit
CPU	Intel Xeon CPU E5-2630 v4 @ 2.20 GHz 2.10 GHz	Intel Xeon CPU E5-2630 v4 @ 2.20 GHz 2.10 GHz	Intel Xeon Gold 6130 CPU @ 2.10 GHz 2.10 GHz
RAM	8 GB	6 GB	6 GB
Network Speed	200 Mbps	200 Mbps	200 Mbps

modes. This means that, we send a happy person picture 1.000 times and we send an angry person picture 1.000 times. We calculate statistics for each picture. The file size for both pictures is 53 KB. At the device side we write Python code for sending pictures and getting string results. We simulate a device which only uses HTTP protocol for streaming data and receiving results. We develop a small web server by using Flask for getting HTTP results at the user device. We use three different servers to implement the architecture. Table 4 show the properties of the servers that we use. We install the service and device software in the same server and apply measurements on this server to overcome the time discrepancy problem that can occur between different servers. We install Redis and CMI Core on two different servers. The source code of CMI Core, service and device software including the measurement parts are uploaded to GitHub.[2] Step-by-step execution of the example case is given below:

1. We assume that the service and the user device register to the CMI as described in Section 5.1.
2. The service connects to the CMI. Connection process includes creating some components in CMI. Additionally, depending on the service, some components may be created in the streaming software (Redis or Apache Kafka). In this example case, the service uses Redis and its RESP protocol for receiving byte streams. For this reason, at the service connection process, a Redis topic is created and the server address and topic name are inserted to the CMI persistent database and returned to the service. The service starts listening to this topic for receiving picture streams. Then, CMI prepares the necessary configuration parameters for this service to send the results of emotion classification. The parameters are stored to the persistent database. These parameters will be used for creating a web server for the service, when the first user device requesting this service is connected to CMI. The service will send HTTP POST messages to this web server. Web servers are created by using the Jetty component of Apache Camel. *We measure the time to complete "Service Connection" process and the mean time for 10 service connection operations is 375 ms.*
3. The user selects services for her devices and their sensors. A web site or an API that is accessible by applications on the user device can be used for this purpose. We assume that, in this sample case, the user selects the emotion classification service that we develop for her robot's camera. When the robot connects to CMI, two operations are applied. The first operation is to create necessary components for device data streams. A Jetty based web server is created and this web server will be used by robot for sending pictures through HTTP protocol. For sending pictures by HTTP POST messages, we use Python httplib2 library. All the messages are constructed by Google Protocol Buffers serialization language. In CMI, a route which is Route 1 is created between the device's web server and service's Redis topic. The second operation is to create some components for the robot to receive results. Because this is the first device requesting the emotion classification service, a web server is created for the service by using the parameters determined at the previous step. This operation will be skipped for other device connection operations requesting this service. After than, Route 2 is created which will get results from the service's web server and transfers it to the "direct" endpoint. This endpoint is used by Apache Camel as an intermediate component to connect different routes. We use this component to provide dynamic routes. By this way, results can be forwarded to more than one receiving endpoint and the

[2] https://github.com/bavenoglu/cmi

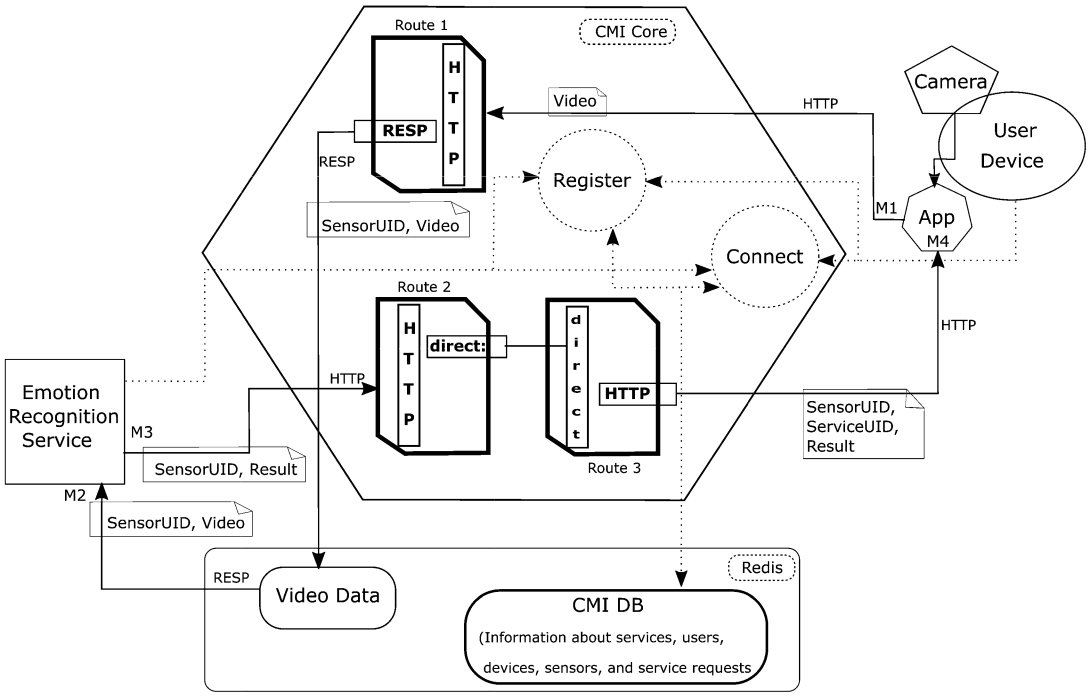

Fig. 10. Information flow of the use case in CMI (dotted lines and circles show the registration and connection operations for devices and services. These operations are prerequisite to send/receive data and results. M1, M2, M3 and M4 represent the time measurement points. Times are recorded in each of these points for each picture.

user can later add other receiving endpoints. Moreover, Route 2 can also be used by sending results of other devices of the same user or other users. Another route, Route 3 is created from the "direct" endpoint to the web server of the user device. By this way, in our example case, the results of the service will be received from the web server in CMI, it will be transferred to "direct" and from direct to the web server of the robot. In other words, we can say that Route 2 is owned by the service and it is a general route for all other users and their devices. However, Route 3 is owned by our user and it will be used to forward results to all devices of our user. *We measure the time to complete "Device Connection" process and the mean time for 10 device connection operations is 1,289 ms.* This time will be smaller for other devices connection operations since no web server for the service will be created.

4. The user device sends a picture to the web server. Route 1 transfers the picture from the web server endpoint to the topic in Redis. Route 1 adds identifier of the user, her device and the device's sensor to the picture data stream and transfers the message coming from HTTP protocol to RESP protocol. The point "M1" in Fig. 10 is measured just before sending the picture at the device side.
5. The service gets the picture from Redis. The point M2 in Fig. 10 is measured just after receiving the picture at the service side. The service applies the emotion classification algorithm and we measure the time consumed by this process. The point M3 is measured when the service completes its operation.
6. The service sends the result message including the identifier of the user, device and sensor to the web server in CMI. Route 2 gets the result from HTTP protocol and sends it to the "direct" endpoint. Route 3 gets the result from the "direct" endpoint and adds service identifier to the result message. This identifier is added at this step, because, the user may have requested many different services for the same data stream. Route 3 transfers the result by sending HTTP POST message to the web server on the device. The point M4 is measured when the result is received at the device side.

We measure the time consumed by four different parts of the middleware. The time difference between M2 and M1 (M2-M1) represents the time difference in which the data is sent from the device and it is received by the service.

Table 5
Performance measurements (M1, M2, M3 and M4 represent the time measurement points showed in Fig. 10)

		M2-M1 (ms.)	M3-M2 (ms.)	M4-M3 (ms.)	End-to-End Delay (ms.)
"Happy" Person Picture	Avg.	61.51	295.78	18.30	375.60
	Std. Dev.	18.11	127.63	21.77	151.07
"Angry" Person Picture	Avg.	61.02	295.50	17.57	374.09
	Std. Dev.	23.31	49.53	8.28	61.28

Table 6
Performance measurements of routing components (routes process 1.000 pictures for each emotion type and they are shown in Fig. 10)

	Route Name	Processing Times (ms.)		
		Minimum	Maximum	Mean
"Happy" Person Picture	Route 1	29	528	35
	Route 2	9	567	15
	Route 3	6	548	11
"Angry" Person Picture	Route 1	29	683	34
	Route 2	8	200	14
	Route 3	5	117	10

This difference is about 61 ms. on average. The service (M3-M2) consumes 295.5 ms. on average which is the main time consuming part of this example. The difference between M4 and M3 which represents the time difference of sending the result from the service to the device is about 18 ms. on average. The total end-to-end delay is about 375 ms. on average. These results are given in Table 5.

We measure the time consumed by each route and we give the statistics in Table 6. According to the results, Route 1 operates through 34–35 ms on average. However, this does not mean that, 35 ms. of the time difference between M2 and M1, which is 61 ms., are consumed by Route 1. Routes transfer the data as much as possible but they continue to operate for other jobs. According to documentation of Apache Camel, logging, copying of messages for error handling and enabling some components for monitoring and management increase the processing time of the routes. The time consumed by Route 1 is greater than other routes. This occurs due to sending 53 KBs data by HTTP protocol. Route 2 takes 14–15 ms. which is almost half of the Route 1. Route 3 consumes 10–11 ms. on average. There is no significant difference between different emotion types. The difference between minimum and maximum processing times of the routes is high. This generally occurs due to initialization of the components when the first picture is sent.

We apply the measurements in the servers connected within the same local area network. The network speed is 200 Mbps and all three servers are virtual servers which are connected to this network. The time difference of M2-M1 and M4-M3 shows that with this network speed, users will not have any latency problems. The difference of M2-M1 is 61 ms. including the network delay. Only 12 pictures of the 1.000 pictures have more than 100 ms. delay between M2 and M1. On the other hand, the time difference between M4-M3 is smaller and it is 18 ms. including the network delay. Only 5 pictures of the 1.000 pictures have more than 100 ms. delay between M4 and M3. In the measurements, we use ".png" pictures which have small sizes. If the applications send big pictures (or audio files) an automatic conversion may be applied at the routes. Even though Apache Camel does not support such conversions, there are third party tools developed for this purpose. Nevertheless, this can also be implemented as a service (internal or external) in CMI. This manipulation can also be handled at the routes since Apache Camel allows accessing the message content and make some operations.

6. Discussion

We proposed and discussed the design and implementation of our cloud-based middleware that integrates AI services to support the development of more interactive applications for end users with their smart devices. This will facilitate application developers to develop interactive applications that provide end users with the benefits of multimodal-multisensor interfaces. We have proposed a cloud-based middleware to enable the use of computationally heavy AI services by applications for smart devices that only have limited computational resources. Conceptually, CMI is an open architecture which creates an ecosystem for connecting devices and integrating AI services. Developers can implement applications by consuming various available AI services in CMI and by using different protocols. They can manipulate the results coming from the AI services. If they want to access an AI service which requires an API and if there is a service in CMI providing this API service, they can do this without attaching and consuming the API. Developers can also develop external services for giving support for such API-enabled services and integrate the external service to CMI. They have the option of developing applications which communicate by many different protocols and data formatting languages. The choice for an open architecture has various advantages. First, it enables independent developers to develop interaction applications for users without any constraints (e.g. use REST API to stream data or use specific protocol and data format to send data to a special service) forced by the architecture. These applications are the major components which provide usable interactions to the users. Interactions can be developed by combining the results of several AI services. Second, it allows service providers to make their services available to many different developers, hence many users. Currently, there are many AI services deployed on the Internet. These services provide different AI services such as face recognition, activity recognition etc. Combining these services to get a composite service is not a trivial programming skill and the result may not be usable for different types of devices. Third, devices and services can use many different protocols which may ease the job of the developers and also, may allow different types of smart devices in different living environments to be included into the architecture. Fourth, thanks to the integration framework, data that are sent by devices and results that are sent by services can be in different data formats. Fifth, developers can develop applications on different devices without using APIs of services. This has two important implications, First, many services may not have support of client libraries for different programming languages. Because of this, a developer cannot develop an application for a specific device. For example, a developer can develop an application on a smart watch without consuming the API client library of a speech recognition service. Second, implementing client libraries is not easy. Developers have to spent some time to learn and implement client libraries. In CMI, developers only have to manage data sending and result receiving operations.

The integration framework that we utilize in our cloud-based middleware enables to create a loosely coupled relationship between end users, smart devices, developers and AI service providers. Integration framework enables to provide interaction services based on AI operations by implementing service-based architecture. The orchestration of services is handled by applications or special composite services which consume other services. An application may also have the choice of orchestrating composite services and other services. Different granularities (e.g. a face recognition service or a dialog service) and types (e.g. SOAP-based web service or microservices) of services are supported. CMI does not have a centralized database for storing data like typical service-oriented architectures. However, data storage is also provided as a service. The only persistent database is used for storing meta-data about users, services and devices. The components provided by the integration framework allow services and applications to use different protocols and understand each other. Message routing capabilities of the integration framework based on different patterns allow consuming services by many applications. Additional to this, message transforming also allows to use different types of smart devices with different capabilities.

We evaluate CMI by using two example cases. In the first case we show how the middleware works. The case includes a basic service and Google Speech-to-Text service. We develop the basic service and it abstracts the Google Speech-to-Text API. By this way, application developers do not have to know the details of the Google's API. Measurements of this example case show that, end-to-end delay of processing an audio stream takes 656 ms. 475 ms. are consumed by the service operations. In the second example case, we give the detailed time measurements from different points of the middleware. In the second example case, we use pictures as data streams and we implement the service in our servers. The time usage of service operations dropped to 295.5 ms. in this second example. This

also decreased the end-to-end delay to 375 ms. Moreover, we measured the times consumed by the routes inside CMI. The results show that, routes use 10 to 35 ms. which are very small according to service operations. The time consumed by Route 1 is higher according to other routes. The reason for this is that, Route 1 takes audio or video data streams and transfers them to other protocols. Route 2 and Route 3 only transfers small text messages such as 'angry' or 'happy'. However, in different cases, the service may send a picture or an audio file as a result of the service. In such cases, routes may show similar performances. The results show that audio or video data types do not add significant difference to the results.

Conceptually, CMI is a middleware for supporting the development of interactive applications for smart devices'. With the implementation of CMI, we aimed at realizing the requirements that we identified in Section 3. Our measurements in Table 3 show some promising results that it is possible to meet RQ1 with a cloud-based middleware. We transfer 1.5 seconds audio data, process it and get the results in 0.656 seconds on average. The cloud-based middleware that we propose, enables many different processes to execute independently on different servers. Even though the performance of the service and devices effect the speed of the total operation, the cloud-based middleware is ready for multi-device and multi-service operation. RQ2 is related with sending data and getting results in almost real-time. We think that two issues highly affect the fast/quick streaming and downloading. The first one is the network speed that the service and smart device use. Even though we cannot intervene to the network speed, enabling developers to use different protocols may give them different options to access high-speed communication. The second one is the overhead of the architecture. The measurements in Section 5 show that CMI, does not put much overhead to the round-trip times of messages between two parties. Besides this, CMI provides streaming option to enable faster communication. RQ3 is the multiple execution of tasks at the same time. CMI is designed for enabling many devices to consume many services. Technically there is no limit for the components inside the CMI. This means that, a data stream can be routed to many services and the results of services can be forwarded to many receiving points. However, several issues may have effects on this requirement. First, the smart device should be ready to send data from several of its sensors. Sending different types (e.g. audio and video) of data may overload the smart device and slows down its communication speed. Second, the performance of the service should be enough to process data coming from different devices. Third, if CMI is used by many devices scalability techniques should be implemented. Even considering these issues, the service-based approach of CMI allows execution of multiple tasks at the same time. RQ4 is related with multiple and diverse data streaming and downloading. We have already tested CMI with text, audio and video data. Any data type can be used with CMI. Moreover, CMI supports multiple-to-multiple relationship between sensors of devices and services. RQ5 is related to storing and accessing data. We provide two different options for application developers to store and access data. First, they have the option of using a streaming software for fast and secure data transfer and access historical data. Second, developers can use a database as a service similar with using an AI service to store data. These two options implement/realize RQ5. We think that RQ6 is one of the most powerful properties of CMI. It is easier for users and developers to be integrated with CMI. They don't have to use any client API to access to CMI and if there are some special services exist, like the one in Section 5, they can use third party services without client APIs even they are forced to be used by client API. Moreover, RQ4 and RQ6 and correspondingly UE6 are satisfied by the integration framework which allows both data streaming and result receiving operations to be handled by different applications and different devices at run-time. RQ7 concerns the reliability of the middleware. We design CMI as a distributed architecture. This allows different components of CMI to be executed concurrently. This allows implementing horizontal scalability easily. Besides this, if a service consumed by a device, does not provide necessary performance, user applications can easily switch to other services without any overhead such as changing the communication protocol. CMI can automatically creates new routes based on user changes. However, currently, we don't test CMI by multiple CMI Core modules. Deploying CMI Core as a microservice may improve the availability and performance of CMI since the middleware is designed for such dynamic changes. We think that the inherent characteristics of cloud-based distributed architectures provide better reliability. CMI and the interaction applications developed based on it can be used to provide better interaction experiences to users and application developers are inspired to use CMI for different context-sensitive fluent interaction applications.

7. Conclusion and future work

In this study, we identify the expectations of users for multimodal interaction, define the architectural software requirements based on these expectations and design and develop an open cloud-based middleware based on these requirements. We established that, users, first and foremost, want to get quick responses from multimodal interactive technology. Users have several devices with different hardware resources and physical properties such as screen size. They want to use all of their devices from anywhere and anytime. Privacy is another important issue for users. The middleware we designed aims to allow users to stream data in parallel from different multiple devices and get results in an acceptable period. The aim has been to facilitate the use of multimodal interaction easily by downloadable and highly available services. Even though scalability and privacy issues are not developed yet, the design of the middleware take these into account for the possible implementation updates at the future. We evaluate the middleware by a use case and show the in-line details. We also performed some measurements of the speed/overhead of the middleware by sending audio messages compatible with the use case.

We design CMI to support multiple devices and multiple services running in parallel. Developers can consume many different services at the same time to provide fluent and context sensitive interaction. This support can also be provided as composite services. Applications and composite services can also be developed for different types of smart devices, e.g. a tablet and smart watch, which shows the flexibility of CMI. Moreover, these applications best suitable when they are context-sensitive and customized for different user preferences. Services for modeling users according to their preferences and adapting applications that they use are highly required. Services or composite services can merge different streams or data from different modalities and the results can also be distributed to different devices in different formats. Developing such applications and composite services need further research.

Currently, CMI does not implement a communication protocol at the semantic level between devices and services. In current implementation, devices directly send data and services process the data and send the results. However, there must be at least a high-level protocol between these to understand each other. For example, a device should send a message indicating that it will start sending data. Furthermore, both devices and services just cut sending and receiving when there is a network problem. Currently, they don't store the state of the communication. When they reconnect again, the previous state of the communication is unknown. However, persistent storage can easily be used for this purpose. Users enter the details of devices and sensors at the registration process. These details include picture format, size, resolution or number of audio channels, recording frequencies etc. The protocol should also include these details based on the sensor at the start of the communication. The protocol should allow indicating these properties. Moreover, these details may be changed dynamically based on the network load of the user when the communication continues. Such a protocol may have some disadvantages. First, the complexity of binding services and devices can increase, since they have to implement the protocol. Second, it may increase the response times because of packaging and parsing of messages written in this protocol.

Another topic that needs more future work is scalability, which we mentioned but did not discuss extensively in this paper. The CMI open architecture should offer scalability to handle connections coming from large amounts of users. In CMI, only the integration framework, Apache Camel, and CMI core module are tightly integrated. Other modules are developed independently. Based on this design, we can offer different scalability solutions. First, basic services, can be developed as microservices to be scalable. Of course, the scalability of external services depends on the third parties that make them available. It would be useful to explore how a semantic-level communication protocol can be used to provide feedback to applications about the load of the external service. Moreover, if applications use abstract services, automatic switching can be supported between similar services when there is a congestion on the preferred service. We envisage that the CMI core module within the integration framework can be scaled by load balancing techniques or again microservices. On the other hand, in case of network congestion, edge/fog computing-based solutions can also be integrated. However, CMI currently does not support this property. To achieve this at least partly, the locations of the services and users may be stored in permanent storage module and application developers can facilitate from this information.

Finally, data security and privacy is a very important topic that should be explored further in future work. One topic to explore in this area also is how external services can prove they meet security standards that are required by CMI. We believe the semantic-level communication protocol should also integrate security commands to services and devices.

Acknowledgement

Bilgin Avenoğlu is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the name of 2219 – International Postdoctoral Research Fellowship Program for Turkish Citizens.

Conflict of interest

None to report.

References

[1] M. Aazam, S. Zeadally and K.A. Harras, Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities, *Future Generation Computer Systems* **87** (2018), 278–289, https://www.sciencedirect.com/science/article/pii/S0167739X18301973. doi:10.1016/j.future.2018.04.057.

[2] N. Almeida, A. Teixeira, S. Silva and M. Ketsmur, The am4i architecture and framework for multimodal interaction and its application to smart environments, *Sensors (Switzerland)* **19**(11) (2019). doi:10.3390/s19112587.

[3] S. Andrist, D. Bohus and A. Feniello, Demonstrating a framework for rapid development of physically situated interactive systems, in: *ACM/IEEE International Conference on Human-Robot Interaction, Vol. 2019-March*, 2019, p. 668, ISSN 21672148. ISBN 9781538685556.

[4] O. Arriaga, Emotion recognition Python source code, https://github.com/oarriaga/paz/tree/master/examples/face_classification.

[5] L. Belli, S. Cirani, L. Davoli, G. Ferrari, L. Melegari, M. Montón and M. Picone, A scalable big stream cloud architecture for the Internet of Things, *International Journal of Systems and Service-Oriented Engineering* **5**(4) (2015), 26–53. doi:10.4018/IJSSOE.2015100102.

[6] D. Bohus, S. Andrist and M. Jalobeanu, Rapid development of multimodal interactive systems: A demonstration of platform for situated intelligence, in: *ICMI 2017 – Proceedings of the 19th ACM International Conference on Multimodal Interaction, Vol. 2017-Janua*, 2017, pp. 493–494. ISBN 9781450355438. doi:10.1145/3136755.3143021.

[7] D. Bohus and E. Horvitz, Situated interaction, in: *The Handbook of Multimodal-Multisensor Interfaces: Language Processing, Software, Commercialization, and Emerging Directions*, Association for Computing Machinery and Morgan & Claypool, 2019, pp. 105–143. ISBN 9781970001754. doi:10.1145/3233795.3233800.

[8] F. Brudy, C. Holz, R. Rädle, C.-J. Wu, S. Houben, C.N. Klokmose and N. Marquardt, Cross-device taxonomy: Survey, opportunities and challenges of interactions spanning across multiple devices, in: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–28. ISBN 9781450359702. doi:10.1145/3290605.3300792.

[9] A.J.B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu and C. Dixon, Home automation in the wild: Challenges and opportunities, in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11*, Association for Computing Machinery, New York, NY, USA, 2011, pp. 2115–2124. ISBN 9781450302289. doi:10.1145/1978942.1979249.

[10] A. Cesta, G. Cortellessa, F. Fracasso, A. Orlandini and M. Turno, User needs and preferences on AAL systems that support older adults and their carers, *Journal of Ambient Intelligence and Smart Environments* **10**(1) (2018), 49–70. doi:10.3233/AIS-170471.

[11] R. Chaari, O. Cheikhrouhou, A. Koubâa, H. Youssef and H. Hmam, Towards a distributed computation offloading architecture for cloud robotics, in: *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, 2019, pp. 434–441. doi:10.1109/IWCMC.2019.8766504.

[12] J. Chen, Y. She, M. Zheng, Y. Shu, Y. Wang and Y. Xu, A multimodal affective computing approach for children companion robots, in: *Proceedings of the Seventh International Symposium of Chinese CHI, Chinese CHI '19*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 57–64. ISBN 9781450372473. doi:10.1145/3332169.3333569.

[13] X. Chen, M. Zhou, R. Wang, Y. Pan, J. Mi, H. Tong and D. Guan, Evaluating response delay of multimodal interface in smart device, in: *Design, User Experience, and Usability. Practice and Case Studies*, A. Marcus and W. Wang, eds, Springer International Publishing, Cham, 2019, pp. 408–419. ISBN 978-3-030-23535-2. doi:10.1007/978-3-030-23535-2_30.

[14] Y. Chen, Z. Du and M. García-Acosta, Robot as a service in cloud computing, in: *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, 2010, pp. 151–158. doi:10.1109/SOSE.2010.44.

[15] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa and A. Kitazawa, FogFlow: Easy programming of IoT services over cloud and edges for smart cities, *IEEE Internet of Things Journal* **5**(2) (2018), 696–707. doi:10.1109/JIOT.2017.2747214.

[16] A. Coskun, G. Kaner and I. Bostan, Is smart home a necessity or a fantasy for the mainstream user? A study on users' expectations of smart household appliances, *International Journal of Design* **12**(1) (2018), 7–20.

[17] D.A. Dahl, The W3C multimodal architecture and interfaces standard, *Journal on Multimodal User Interfaces* **7**(3) (2013), 171–182. doi:10.1007/s12193-013-0120-5.

[18] T. Dong, E.F. Churchill and J. Nichols, Understanding the challenges of designing and developing multi-device experiences, in: *Proceedings of the 2016 ACM Conference on Designing Interactive Systems, DIS '16*, Association for Computing Machinery, New York, NY, USA, 2016, pp. 62–72. ISBN 9781450340311. doi:10.1145/2901790.2901851.

[19] M. Dunne, G. Gracioli and S. Fischmeister, A comparison of data streaming frameworks for anomaly detection in embedded systems, in: *International Workshop on Security and Privacy for the Internet-of-Things (IoTSec)*, Orlando, USA, 2018.
[20] D. Ferreira, N. Almeida, S. Brâs, S. Soares, A. Teixeira and S. Silva, Enabling multimodal emotionally-aware ecosystems through a W3C-aligned generic interaction modality, 2020. doi:10.1007/978-3-030-49289-2_11.
[21] Google cloud speech-to-text, speech-to-text basics | cloud speech-to-text documentation. https://cloud.google.com/speech-to-text/docs/basics.
[22] W. Green, D. Gyi, R. Kalawsky and D. Atkins, Capturing user requirements for an integrated home environment, in: *Proceedings of the Third Nordic Conference on Human–Computer Interaction, NordiCHI '04*, Association for Computing Machinery, New York, NY, USA, 2004, pp. 255–258. ISBN 1581138571. doi:10.1145/1028014.1028053.
[23] E. Guizzo, Robots with their heads in the clouds, *IEEE Spectrum* **48**(3) (2011), 16–18. doi:10.1109/MSPEC.2011.5719709.
[24] J.T. Hansberger, C. Peng, V. Blakely, S. Meacham, L. Cao and N. Diliberti, A multimodal interface for virtual information environments, in: *Virtual, Augmented and Mixed Reality. Multimodal Interaction*, J.Y.C. Chen and G. Fragomeni, eds, Springer International Publishing, Cham, 2019, pp. 59–70. ISBN 978-3-030-21607-8.
[25] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Longman Publishing Co., Inc., USA, 2003. ISBN 0321200683.
[26] D. Hunziker, M. Gajamohan, M. Waibel and R. D'Andrea, Rapyuta: The RoboEarth cloud engine, in: *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 438–444. doi:10.1109/ICRA.2013.6630612.
[27] C. Ibsen and J. Anstey, *Camel in Action*, 2nd edn, Manning Publications Co., USA, 2018. ISBN 1617292931.
[28] T. Jokela, J. Ojala and T. Olsson, A diary study on combining multiple information devices in everyday activities and tasks, in: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15*, Association for Computing Machinery, New York, NY, USA, 2015, pp. 3903–3912. ISBN 9781450331456. doi:10.1145/2702123.2702211.
[29] J.E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel and D. Mosse, Seamless integration of heterogeneous devices and access control in smart homes, in: *Proceedings of the 2012 Eighth International Conference on Intelligent Environments, IE '12*, IEEE Computer Society, USA, 2012, pp. 206–213. ISBN 9780769547411. doi:10.1109/IE.2012.57.
[30] L. Kleinrock, Nomadicity: Anytime, anywhere in a disconnected world, *Mob. Netw. Appl.* **1**(4) (1996), 351–357.
[31] A. Koubaa, A service-oriented architecture for virtualizing robots in robot-as-a-service clouds, in: *Architecture of Computing Systems – ARCS 2014*, E. Maehle, K. Römer, W. Karl and E. Tovar, eds, Springer International Publishing, Cham, 2014, pp. 196–208. ISBN 978-3-319-04891-8. doi:10.1007/978-3-319-04891-8_17.
[32] A. Koubâa, M.-F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze and A. Mahmoud, Mybot: Cloud-based service robot using service-oriented architecture, *robótica* **107** (2017), 8–13.
[33] NAOqi 2.8 – developer guide, Softbank Robotics, http://doc.aldebaran.com/2-8/index_dev_guide.html.
[34] M.W. Newman, A. Elliott and T.F. Smith, Providing an integrated user experience of networked media, devices, and services through end-user composition, in: *Pervasive Computing*, J. Indulska, D.J. Patterson, T. Rodden and M. Ott, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 213–227. ISBN 978-3-540-79576-6. doi:10.1007/978-3-540-79576-6_13.
[35] E. Nouri, A. Fourney, R. Sim and R.W. White, Supporting complex tasks using multiple devices, in: *Task Intelligence Workshop @ WSDM 2019 Conference*, ACM, 2019, https://www.microsoft.com/en-us/research/publication/supporting-complex-tasks-using-multiple-devices/.
[36] J.-B. Onofre, *Mastering Apache Camel*, Packt Publishing, 2015. ISBN 1782173153.
[37] S. Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos and A. Kruger, Introduction: Scope, trends, and paradigm shift in the field of computer interfaces, in: *The Handbook of Multimodal-Multisensor Interfaces, Volume 1*, S. Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos and A. Kruger, eds, Association for Computing Machinery (ACM), United States of America, 2017, pp. 1–15. ISBN 9781970001679. doi:10.1145/3015783.3015785.
[38] M. Patterns, *Microsoft Application Architecture Guide*, 2nd edn, Microsoft Press, USA, 2009. ISBN 073562710X.
[39] R. Pereira and E.G. Pereira, Dynamic adaptive streaming over HTTP and progressive download: Comparative considerations, in: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014, pp. 905–909. doi:10.1109/WAINA.2014.139.
[40] C. Röcker, M.D. Janse, N. Portolan and N. Streitz, User requirements for intelligent home environments: A scenario-driven approach and empirical cross-cultural study, in: *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-Aware Services: Usages and Technologies, sOc-EUSAI '05*, Association for Computing Machinery, New York, NY, USA, 2005, pp. 111–116. ISBN 1595933042. doi:10.1145/1107548.1107581.
[41] M.N.O. Sadiku, Y. Wang, S. Cui and S.M. Musa, Ubiquitous computing: A primer, *International Journal of Advances in Scientific Research and Engineering (IJASRE)* **4**(2) (2018), 28–31, https://ijasre.net/index.php/ijasre/article/view/852. doi:10.7324/IJASRE.2018.32611.
[42] Samsung, Specifications | Samsung Galaxy S8 and S8+ – The official Samsung Galaxy site. https://www.samsung.com/global/galaxy/galaxy-s8/specs/.
[43] G. Sandström, Smart homes and user values: Long-term evaluation of IT-services in residential and single family dwellings, PhD thesis, KTH, Architecture, 2009, QC 20100809. ISSN 1402-7461. ISBN 978-91-7415-479-5.
[44] N. Tolia, D.G. Andersen and M. Satyanarayanan, Quantifying interactive user experience on thin clients, *Computer* **39**(3) (2006), 46–52. doi:10.1109/MC.2006.101.
[45] J. Wagner, F. Lingenfelser, T. Baur, I. Damian, F. Kistler and E. André, The social signal interpretation (SSI) framework: Multimodal signal processing and recognition in real-time, in: *Proceedings of the 21st ACM International Conference on Multimedia, MM '13*, Association for Computing Machinery, New York, NY, USA, 2013, pp. 831–834. ISBN 9781450324045. doi:10.1145/2502081.2502223.

[46] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle and M.J.G. Molengraft, RoboEarth – a world wide web for robots, *Robotics & Automation Magazine, IEEE* **18** (2011), 69–82. doi:10.1109/MRA.2011.941632.

[47] J. Wan, S. Tang, H. Yan, D. Li, S. Wang and A.V. Vasilakos, Cloud robotics: Current status and open issues, *IEEE Access* **4** (2016), 2797–2807. doi:10.1109/ACCESS.2016.2574979.

[48] M. Weiser, The computer for the 21st century, *SIGMOBILE Mob. Comput. Commun. Rev.* **3**(3) (1999), 3–11. doi:10.1145/329124.329126.

[49] H. Yang, W. Lee and H. Lee, IoT smart home adoption: The importance of proper level automation, *Journal of Sensors* **2018** (2018), 1–11. doi:10.1155/2018/6464036.