TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/09 pp581-597
Volume 21, Number 6, December 2016

Towards a Service-Oriented Architecture for a Mobile Assistive System with Real-time Environmental Sensing

Darpan Triboan, Liming Chen, Feng Chen, and Zumin Wang*

Abstract: With the growing aging population, age-related diseases have increased considerably over the years. In response to these, Ambient Assistive Living (AAL) systems are being developed and are continually evolving to enrich and support independent living. While most researchers investigate robust Activity Recognition (AR) techniques, this paper focuses on some of the architectural challenges of the AAL systems. This work proposes a system architecture that fuses varying software design patterns and integrates readily available hardware devices to create Wireless Sensor Networks (WSNs) for real-time applications. The system architecture brings together the Service-Oriented Architecture (SOA), semantic web technologies, and other methods to address some of the shortcomings of the preceding system implementations using off-the-shelf and open source components. In order to validate the proposed architecture, a prototype is developed and tested positively to recognize basic user activities in real time. The system provides a base that can be further extended in many areas of AAL systems, including composite AR.

Key words: Activities of Daily Living (ADL); Service-Oriented Architecture (SOA); semantic web; ontology modeling; Web Ontology Language (OWL); Activity Recognition (AR); Smart Homes (SH); Wireless Sensor Networks (WSNs)

1 Introduction

The global aging population is estimated to reach 2 billion by 2050[1–3], and such an increase will inevitably create a larger demand on the health care system that is already facing a shortage in resources. The Smart Homes (SH) environment is a technological solution for this modern-day problem, and it works by monitoring and gathering contextual Activity Recognition (AR) date from inhabitants to come up with solutions to provide real-time assistance and care for the patient or elderly person. However, many problems must be resolved so that SH can fully simulate and take the role of a care-provider or health care professional to certain degree[4].

This paper is set within the context of current problems related to the delivery of high-quality of care for the aging population by health care professionals, which focused on addressing the three levels of system architecture challenges in building an assistive system. These levels are (1) selecting appropriate style and design pattern, (2) considering specific technological and technical requirements for activity recognition, and (3) building and integrating appropriate wireless sensor technologies for providing real-time assistance and monitoring. The sections below introduce these three levels, identify the key challenges, and discuss

• Darpan Triboan, Liming Chen, and Feng Chen are with the Context, Intelligence, and Interaction Research Group (CIIRG), De Montfort University, Leicester, LE1 9BH, UK. E-mail: darpan.triboan@my365; {liming.chen, fengchen}@dmu.ac.uk.
• Zumin Wang is with the Department of Information Engineering, Dalian University, Dalian 116622, China. E-mail: wangzumin@163.com.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-07-01; revised: 2016-09-12; accepted: 2016-10-03

their potential opportunities.

1.1 Assistive system architecture style and pattern

One of the main system architectural challenges in building an assistive system is to select appropriate design styles and patterns, that, unfortunately, may be easily misused[5–7]. Engaging with the wider community by having open source components and using popular programming languages can play a key role in coming up with useful, adaptive, and personalised solutions. Other factors influencing the design decisions include: semantica data storage, computation power requirement, low latency communication protocols, and the ability to allow simultaneous access to the users with a convenient Human-Computer Interface (HCI). Some of the exiting assistive systems (explored further in Section 2) are built in a standalone application environment. However, questions have been raised regarding its extensibility, reusability, scalability, maintainability, and/or use of proprietary components, which may have limited community support. In addition, having a poor or an unnatural HCI design poses practical limitations for the key users.

Over the years, the Service-Orientated Architecture (SOA) approach has become popular, because it can address some of the aforementioned issues as well as create a mechanism by which to delegate resource-intensive tasks and storage to powerful sets of computers over a network (cloud computing). Moreover, using the SOA approach also allows low-power devices such as mobile devices or any other gadgets with network capabilities, to utilise the available services. This has not only improved the HCI of the system, but also made it scalable such that it can serve cross-platform clients as well as integrate and reuse third-party services in a creative manner. The approach now drives the concepts of SH, Internet-of-Things (IoT), and ubiquitous or pervasive computing. This is the main approach by which everyday objects can be seamlessly integrated into the interconnected World Wide Web (WWW).

1.2 Activity Recognition (AR)

A key part of an assistive system is to achieve accurate AR. However, AR capabilities within SH pose many challenges. AR involves three main tasks in AR: activity modeling, data collection and monitoring, and data processing and pattern recognition[8]. The first task aims to create computational activity models from which the system infers and performs reasoning. These models can be generated using two different approaches, namely, data-driven and knowledge-driven. The data-driven approach involves processing the predefined data to create a training model by using various machine-learning techniques. In contrast, the knowledge-driven approach takes the conceptualization of the real world axioms (i.e., established or accepted statements)[9, 10] and, from these, formally defines the domain specific knowledge explicitly. The second task aims to monitor and capture the inhabitants' behaviors along with the changes in the environmental conditions. Here a wide range of monitoring technologies and devices can be used, such as vision-based and sensor-based techniques, which depend on various factors, such as the type of information required, granularity level, privacy, and technical availability/feasibility of the devices. The third task aims to process sensor data and map the extracted patterns against the activity model created in the initial stage to determine which activity is performed and, from such information, provide assistance accordingly.

In addition, within the first task of activity modeling, the data-driven approach includes the generative and discriminative methods that employ statistical and probabilistic methods to analyze pre-existing datasets and derive activity models. This approach can handle modeling uncertainties and temporal information. However, this technique suffers from the "cold start" problem because of the need of pre-existing datasets for model learning, which in turn leads to reusability and scalability issues. In comparison, knowledge-based modeling is performed using formal theories (mining- and logical-based theories) and domain expertise to create activity models[11]. In turn, this approach eliminates the need of learning from a pre-existing dataset, hence, no "cold start" problems. However, the knowledge-driven approach also suffers from handling uncertainties and temporal information as a result of the manual pre-defined activity models. A previous study proposed a hybrid approach in this work[12] to address the shortcomings of both approaches by developing an incremental model discovery and activity learning method.

For each of the abovementioned AR tasks, various interdependent underlying technologies also exist[8]. These technologies present further integration challenges, which are mainly due to their differences in programming languages, development environments,

proprietary components, and communication protocols. Therefore, interconnectivity of each stage into a single technology poses challenges for researchers.

Another challenge that arises from this topic is the problem of storing the activity modeling and recognition data using the semantical structure in such a way that the data can later be used in a meaningful way.

The storage options considered here also influence the overall system architectural design decisions. Recently, this has become a much wider issue with the accumulation of large amounts of unstructured or semi-structured data with no clear semantical relations. This has created many problems, such as automating the task of processing and retrieving data efficiently[13]. Currently, machine-learning techniques, such as genetic algorithms, are used to extract and train computers on how to process the data over time. This approach, however, is lengthy and requires a high computation rate. To make this process more efficient, the concept of the semantic web was introduced. This concept was originally envisioned by Tim Berners-Lee and his colleagues to create the Web with linked data, which have semantic meanings, formalisms, and a structure that can be processed by a machine[9, 14]. This is achieved by representing the data in the form of a triplet, subject-predict-object. The most common vocabularies are used and shared to create an expressivity in the data (i.e., using Resource Description Framework (RDF)[15, 16] and Web Ontology Language (OWL)[17]). In addition, various reasoning engines (i.e., Pellet, HermiT, and FaCT++) are used to perform inferencing utilizing the user-specific rules and formal languages. The triple datasets can be stored in the triple-store (database) as a graph, which are specially optimized for handling them. Moreover, just like the Structured Query Language (SQL) in traditional relational databases, the SPARQL protocol and RDF Query language (SPARQL) is used to perform Create, Read, Update, and Delete (CRUD) operations[15, 18]. These capabilities and benefits enable the back end of any applications to achieve greater flexibility within its specific system architecture.

1.3 Wireless Sensor Networks (WSNs)

WSN technology has enabled a large variety of applications to be developed; these have also been applied across many domains, i.e., military[19], healthcare, transport[20], and smart city infrastructure. WSNs play an important role in emerging Network-of-Things (NoT) or IoT paradigms[21]. The capabilities of the WSNs within the assistive systems can be seen as a supporting tool to allow humans or machines to interact with their environment and react to real-world events[22]. Therefore, the key responsibility of WSNs is to acquire environmental data from remote nodes and execute commands instructed by a coordinator, also known as sink or base station. Depending on the application requirements, various communication protocols are available, through which a remote node can send data to the coordinator. These protocols have their own properties, benefits, and limitations but they can be characterized by their range and energy consumptions. Some of the popular protocols are ZigBee[20], Z-Wave, WiFi, 6LoWPAn, 2G/3G/4G/5G, Blue-tooth (+BLE), Radio Frequency Identification (RFID), Near Field Communication (NFC), and infrared.

Owing of the diversity of communication protocols, there exist a large number of vendors who create application-specific off-the-shelf products that are not always open-source. This can create a big challenge as far as integrating them within WSNs of any given size is concerned. However, to address this challenge, many efforts have been exerted by the vendors in recent years. One common practice is to provide Application Program Interfaces (APIs) and Software Development Kits (SDKs) to allow cross-platform third-party service integrations. For instance, Securifi Almond+ router, Amazon Echo[23], and Samsung SmartThings[24] have the ability to interact with each other's devices. Although these services are growing, limited intelligence can be added to the sensor nodes as they are governed by rules, such as "if this, then that" concepts (i.e., IFTTT[25]). Furthermore, they still have limited types of sensors that can support fine-grained sensing capabilities for AR, i.e., capacitive touch sensor on an object for dense sensing. Therefore, bespoke Arduino-based wireless sensing methods are still commonly used[26, 27]. The current paper integrates some of these aforementioned off-the-self and open-source WSN technologies within the system architecture to achieve real time AR, monitoring, and assistance.

The consecutive sections are organized as follows. Section 2 discusses related works and existing systems to find their shortcomings. Sections 3 and 4 present a proposed system architecture and the implementation details of an assistive system, respectively. Section 5 analyzes the experimental results and provides some

discussions. It must be noted here that the nature of this paper is not to propose a new way of modeling or recognizing activities, but rather to assess the feasibility of the proposed system architecture and to prepare for further works in these areas. Finally, Section 6 concludes the paper with recommendations for further work.

2 Related Work

In the past, several assistive systems were implemented. In particular, two prototype assistive systems were implemented to provide activity recognition and assistance features for the elderly or those who have cognitive difficulties in carrying out Activities of Daily Living (ADL), namely, the SMART system[28, 29]. In its initial implementation, the SMART system was built in a standalone environment with a direct interface to the SH environment and featured a rich web-based interface using dotNet programming language. As shown in Fig. 1, the SMART system consists of six main classes: speech core, reasoning core, preferences core, communication core, simulation recording core, and database tools core. The speech core class is used to output pre-recorded audio messages to the user when the assistance is triggered; personalization of the pre-recorded message is also supported. The reasoning and preferences core classes are the core components of this system. The reasoning core class is used to infer the users' activities from their preferences. The user preferences are administered via basic or advance learning methods presented by the system as well as the sensor activation data retrieved from the communication core. The data from the sensor activations (i.e., inferred activities from reasoning) can be recorded using simulation recording core class. Such data can then be exported to the users local disk or stored in a repository database as a history log.

In the latter implementation, the SOA approach was introduced (see Fig. 2) with open-source components. The core system was written in one of the more popular programming languages, Java. The main reasons were to move away from a standalone environment as well as to resolve limited community support and proprietary components. This approach allows many users from multiple devices to communicate simultaneously with platform independency. The system further addresses the monolithic code structure of the source code by logically separating it into three web services. The Enterprise Service Bus (ESB) supporting software is used to bind these services together; thus enabling better maintainability, reuse, and debugging. The system still has a web-based interface that uses JavaScript, Asynchronous JavaScript, and XML (AJAX) features to request and load the data from the ESB. In addition, the Simple Object Access Protocol (SOAP) and Hypertext Transfer Protocol (HTTP) have been used for exchanging data between different devices. Moreover, this service has the potential to be deployed on to the cloud servers that possess superior computational capacity to perform very complex reasoning within a short amount of time[30]. One of the disadvantages of using this system, however, is that it has multiple web services with an ESB, which requires it to be hosted on the network. This can create unnecessary overhead and delays in the system.

A previous study[31] presented a location-based context-aware system architecture, in which a range of stakeholders can work collaboratively. The users do

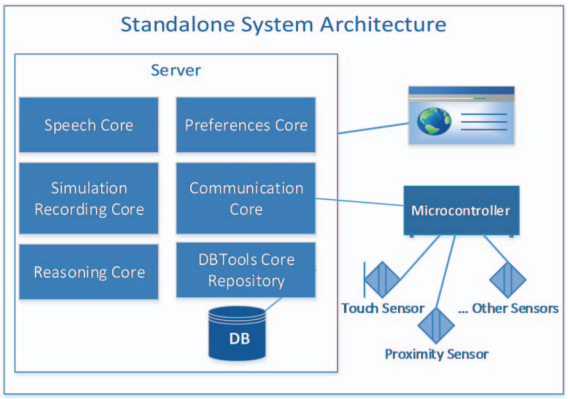


Fig. 1 System architecture overview: Initial implementation of the SMART system (2009).

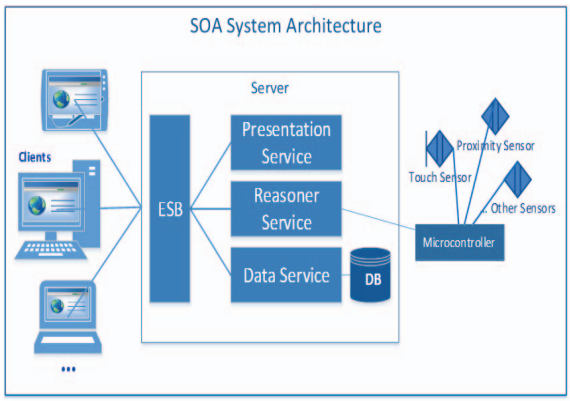


Fig. 2 System architecture overview: Service-oriented implementation of the SMART system (2012).

not require any prior knowledge of programming skills to model, manage rules, infer, and specify actions. The system adapts the SOA style architecture and has a web browser-based interface similar to a SMART SOA system. The results of the study indicate that the system is easy to use; however, the performance of the reasoning degrades with the increase in the number of models and the complexity of the rules. Likewise, Ref. [32] provides a pioneering OPEN framework. The OPEN framework is based on ontology for rapid prototyping, sharing, and personalization of the system for the cooperative use of the developer, and non-expert users.

A number of other related works exist in the literature. For example, the assistive system[33] enables remote assistance and monitoring between the hospital and the clients' SH environment. Another study[34] proposed an SOA-style architecture involving a mobile device and a web service to detect objects in real-time by using image analysis techniques and augmenting the assistance on the user's tablet; here, a data-driven approach is employed through which images in the database are analyzed. Meanwhile, the work in Ref. [35] adapts the knowledge-driven approach to propose a multi-tier architecture for an autonomic Ambient Intelligent system. The system exploits ontology modeling techniques and logical rules [Java Expert System Shell (Jess)] to formally describe the environment as well as to infer and reason the activity. In addition, Ref. [36] fused the data-driven and knowledge-driven techniques to achieve unusual behavior recognition with the help of Decision Support Systems (DSS) and the ontology modeling technique for activity inferencing. The system provides a natural interaction (i.e., speech and gesture) within the smart environment and everything is controlled by the centralized server.

The current paper proposes a new system architecture and presents the system prototype to enhance service-oriented implementation. In addition, further assistive features have been implemented, such as medicine dose management[37], appointment management, and notifications services. The implemented system extends the previous web-based service to the more relevant mobile assistive service. The usage of the mobile phone's sensor capabilities can also play a role in supporting additional application scenarios for the inhabitant and improving the system's usability. Table 1 provides an overview of the two SMART system implementations along with the proposed system.

3 Proposed System Architecture

The proposed system continues with the SOA approach, but develops the web service using Representational State Transfer (REST) protocol instead of the SOAP. In addition, the HCI with the SMART system has been improved by building an Android application instead of a browser-based interface, see Fig. 3. By creating the mobile application, it not only supports patients and caregivers on the move, but potentially enable other stakeholders of the system (e.g., a patient's family members and relatives) to be more connected when the system is developed in the future. In addition, new features that can further assist the inhabitant

Table 1 Comparison between predecessors and the proposed system.

System details	System version		
	SMART		Proposed
	(2009)	(2012)	(2016)
Purpose	Activity recognition using Smart Homes	Reengineered based on the initial version	Reengineered with SOA implementation.
Implementation type	Standalone web application	SOA; SOAP-based; browser-based interface	SOA; REST-based; SSE and mobile application
Language (s)	C#, ASP.NET, dotNet/GWT based	Java, AJAX, JavaScript, HTML/CSS, SQL	Java and SRARQL
Main dependencies	Semantic Web (SemWeb), AJAX, Silverlight, Euler, and Pellet	PELLET (reasoning tool), Apache Jena, Mule ESB, Glassfish, JAX-WS, H2 RDBMS, AJAX	Apache Jena, Fuseki Server, JAX-RS 1.1, Jersey 2, Jersey SSE, XBee Java lib, Tyrus Web sockets, Apache Tomcat Server and Android Studio.
Interface	Browser-based	Browser-based	Mobile-based (Android application)
Portability	Single computer	One-to-many	One-to-many
Licensing	Proprietary	Open-source	Open-source

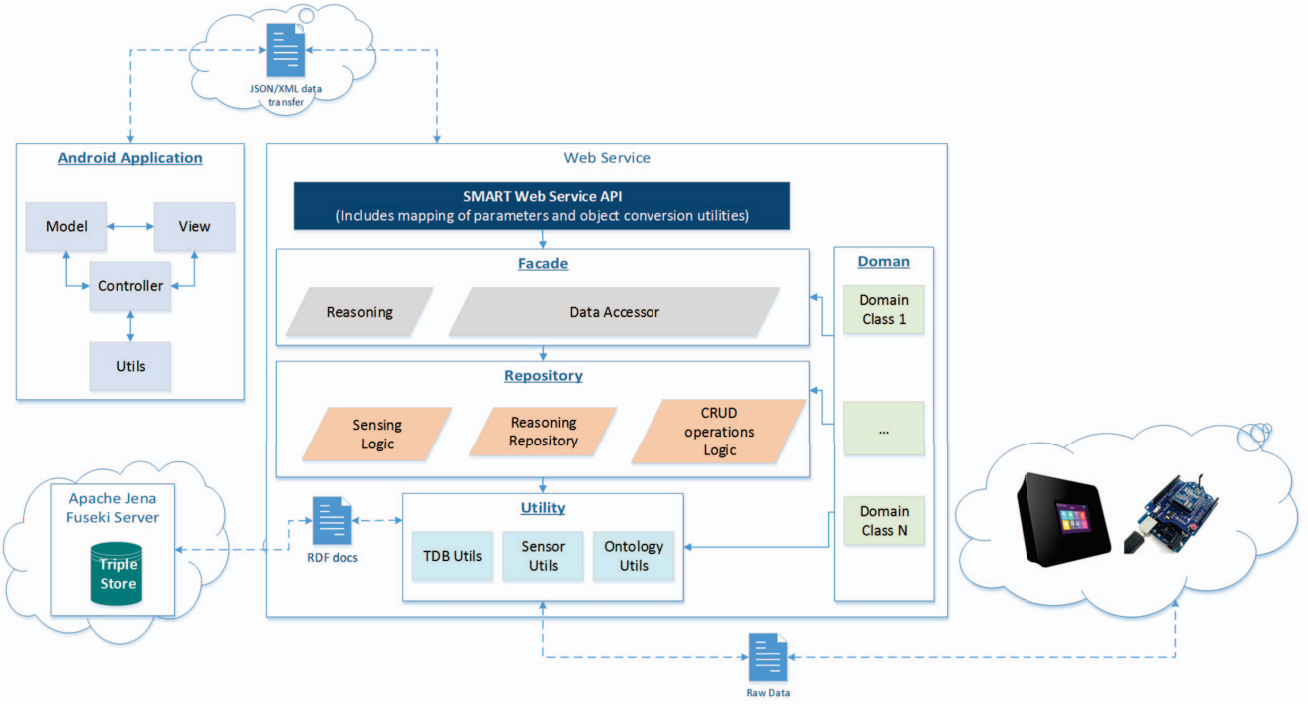


Fig. 3 The proposed mobile SMART system using SOA and semantic web technologies.

in living independently or in the care home have been added. The features are derived from recent inspection reports of various care homes as carried out by Care Quality Commission[38]. Three different web services have been combined into one web service by using suitable software design patterns, such as Facade, Repository, and MVC; hence removing ESB to reduce the communications overhead. Other creational, structural, and behavioral patterns are also considered[7, 39]. Furthermore, the triple-store (Jena Fuseki server[40]) is used to create a distributed system that can be published, reused, and shared, thereby contributing towards the vision of a semantic web and linked data in the future. Overall, the proposed system consists of three main components: REST-based web service (including triple-store), the mobile application, and the sensing network.

3.1 Web service

The REST-based web service has been identified to be better suited for the following reasons. The REST-based protocol is lightweight in nature, and is easy to use and implement compared with the SOAP web service. The SOAP-based protocol supports richer functionalities, but incurs communication overhead[38, 41]. In addition, it poses some restrictions in terms of flexibility, explicit functional parameter requirements, and the data format that it can produce and consume. In comparison, the JAX-RS library[42] in the REST-based service does not require function parameter definitions or publication of their service, i.e., with Universal Description, Discovery, and Integration (UDDI). Another main feature of the REST-based service is that it enables clients to consume and produce data in a variety of data formats, such as XML, JSON, HTML, and encoded text. This makes the system more interoperable compared with others and gives it the ability to support low-powered devices, thus reducing their limited energy consumption resulting from its lightweight nature.

One of the main requirements for the web service is to capture and expose all the sensor data and activity inferencing results to the client devices upon user interactions with the environment. This is achieved by broadcasting the real time sensor data to the clients using the Server-Sent Events (SSE)[43] mechanism instead of a bi-directional WebSockets or pooling method. One of the main reasons for this is to reduce the connection overhead. Although SEE is a bi-directional protocol, other standard requests can still be made by a client outside their SSE connection asynchronously. Another requirement of a web service is to capture and process sensor data that are communicated to the server in various media formats depending on the device vendor. In this proposal, the web service currently supports Almond+ router WebSocket connection, XBee

coordinator connected via comport, and other Arduino-based sensor collection using standard comports (see Sections 1.3 and 3.3 for more details).

The Jena Fuseki server has been used in order to achieve a distributed collection of data for higher scalability, reuse, and performance; however, other triple-stores are also available. Furthermore, this server supports the Java programming language and works well together with the Apache Jena API[44], a supporting library that can be used to perform SPARQL queries and reasoning on the graph models stored on the server. Furthermore, the Jena Fuseki server supports various development tools, such as command line execution of the data (ARQ), and user-friendly web-based interface with which to write, perform queries, and manage multiple datasets.

The web service uses a combination of design patterns, such as facade and repository to layer. In addition, the components are logically separate from the three web services of the SOA SMART implementation, in terms of the task level being performed by the classes. This process created five major layers: Smart Web Service API, Facade, Repository, Domain, and Utility. The Smart Web Service API exposes services as an API to enable client devices to consume their features/data. The Facade layer presents classes that perform high-level commands for complex operations by utilizing multiple repository classes. This layer also enables general CRUD operations to take place, hence serving as the Data Accessor component. The repository layer is where the main logics are defined to perform querying and updating tasks to the Fuseki server, accessing sensor states, and creating the reasoner repository to enable inferencing using rules and variances of reasoner implementations. The domain layer contains classes that enable data to be mapped when communicating among the Fuseki server, Web service, and the Android application. Finally, at the utility layer, low level processes are performed, i.e., communicating with the Fuseki server via HTTP and with sensor devices via serial ports, as well as supporting ontology management for inferring and reasoning.

3.2 Mobile applications

Smartphones have become more ubiquitous and have been integrated to part of the modern lifestyle. Smartphones are continuously becoming more powerful with a diverse number of embedded sensors. In the future these can be used for better contextual data collection as well as better usability of the system. In addition, delegating resource-intensive tasks to cloud-based service approaches can not only further increase the capabilities of smartphones but also open up endless possibilities, such as Mobile Cloud Computing (MCC)[45], Cloud-based Mobile Augmentation (CMA)[46], and Image Recognition processing (i.e., mobile landmark recognition systems)[47]. The old browser-based applications in previous system implementations make a system less accessible to its users. For instance, the patients and carergivers would need to carry a laptop, tablet, or other browser-based devices to interact with the web service in order to receive real-time assistance. Furthermore, a browser-based application may not be able to utilize all services available on the device, whereas built-in hardware components, such as a heart rate sensor, can be used to detect/monitor the users' inactivity. Further hardware devices can be attached to mobile devices using wired or wireless communication protocols, such as Bluetooth, NFC, and Infrared. This capability allows limitless possibilities to collect diverse types of contextual data about the user.

The mobile application in the proposed architecture provides the main User Interface (UI) that makes an asynchronous HTTP request to the REST web services. It uses a simple Model-View-Controller (MVC) design pattern to logically separate the classes. The model package contains all of the domain models that are used to map the data communicating with the web service. The view package can be composed of all the classes that are being used to display views on the screens, i.e., activity classes, fragment classes, and dialog classes. Depending on the user type, the view package may have further sub-packages to separate all the views. The controller package may consist of all the classes that trigger requests to the server with the help of the utility classes, mainly view listeners and adapters. Finally, the utility package holds all the support classes, such as HTTP async requester classes, data parsing classes, data dictionary classes, and date format utility.

The SOA approach essentially follows a client-server pattern, in resolving some of the technical challenges mentioned above in building an assistive system using the SH environment. For instance, a Web Service as a service provider and a Mobile application as a client, can work well together to bridge the communication gaps between the SH environments and mobile device

as well as to make the system more flexible in terms of scalability, performance, and platform independency.

Furthermore, the web service can take advantage of cloud computing technology to increase the ability to perform complex reasoning or computation tasks effortlessly. The main benefits of using the mobile device can be numerous. For example, it would not only allow the inhabitant to have a better HCI, but also allow the utilization of embedded sensors within the device or the attachment of external devices using wireless connectivity (i.e., Bluetooth). Such devices, such as Smartwatch and Shimmer[48] sensing devices can be used to obtain additional contextual information about the inhabitant to increase AR accuracy, which, in turn, can lead to the provision of adequate assistance.

However, despite the advantages of using smartphone-based application, providing every patient in the care home with a smartphone may not be financially feasible and getting the elderly to use it can pose further challenges. Therefore, providing efficient and natural HCI methods for an elderly can reduce those problems to a degree. For instance, the recent introduction of devices, such as Amazon Echo[23] provides voice-based interaction to the system and the ability to interconnect with smartphone and other smart devices using SmartThings[24], can be advantageous.

3.3 Sensing network

As discussed in previous sections, a diverse number of sensors and communication protocols are currently available in the market. The proposed architecture currently uses the Securifi Almond+ router to perform ambient sensing, Arduino boards for dense sensing, and Amazon Echo for voice interaction (see Section 4.1 for configuration details). The Securifi Almond+ router is used as a main "IOT" hub because of its WiFi, ZigBee, and Z-Wave protocol capabilities. Other hubs supporting similar protocols are also available, such as Libelium Waspmote[49], SmartThing Hub, and VeraLite. However, further investigation may be required in order to obtain real-time data from these hubs. The popular Ardunio boards and shield-based approach provides greater capabilities and flexibility with which to perform sensing; however, additional steps are required to configure the individual components. Meanwhile, the Amazon Echo currently supports WiFi and Bluetooth communication protocols, thus allowing voice interaction capabilities with third party services.

In relation to overall system architecture, the "*Utility*" library consists of packages and classes through which to extract, store, and process the data from the sensing hardware devices. In particular, the "*Sensor Utils*" package contains sub-packages and classes that interact with third-party APIs and hardware libraries (i.e., "**.almond*" and "*.ardunio*"). Some of the key Java libraries used are WebSocket API (for Almond+ router), XBee, and comPort (both for Arduinio). Moreover, these classes are used by the parallel thread classes to log the events ("*EventLogThread*"), perform device management ("*DeviceManagementThread*"), and store the data in the triple-store ("*TDBStorageThread*"). Figure 4 illustrates the abovementioned utility library structure.

4 Implementation

The SMART system has been re-engineered to perform ADL assistance within both simulated and real environments. The inferencing is currently performed by using the preference matching technique from the users' pre-defined preferences list. In addition, the care homes inspection reports provided by the Care Quality Commission[50] have been analyzed, and several problems have been identified. Seven application scenarios are considered important; hence, these are partially implemented to support users. The scenarios are as follows: to allow the user to manage daily medication doses, appointments, and shopping checklists, as well as to report issues, make requests for bedwetting assistance, detect inactiveness (i.e., by using the users heart rate values), and faciliate smart bedroom

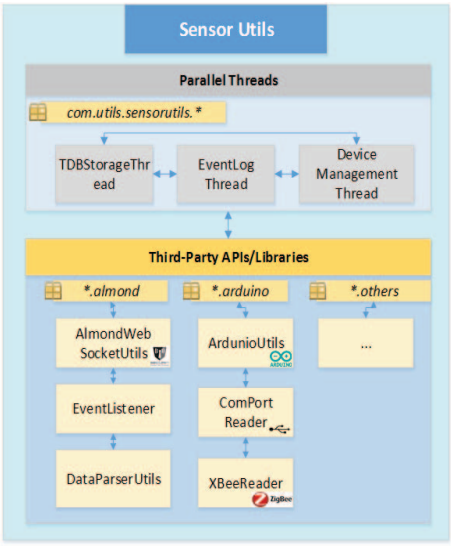


Fig. 4 Software: Breakdown of the "Sensor Utils" package.

cupboard interaction.

Currently, only the web service supports all of the features described above, whereas the Android Operating System (OS) based application is yet to be fully resolved. The real-time ADL inferencing and simulation environments as well as the preference management and medicine dose management interfaces have been implemented to demonstrate this architecture. An Android OS based application has been selected because of its availability, popularity, large community support, and previous experiences of working with Android applications. Other operating systems were considered, however, due to a lack of resources and essential skillsets, it was not considered further.

Apart from the technologies already mentioned in the previous sections, other supporting software components that are used to build the system are Jersey libraries[42] (i.e., Jackson library for JavaScript Object Notation (JSON) strings to object mapping), Jena[16, 44] Pellet (reasoner, see others[51]), Protege[52] (ontology editing tool), and Google API Services[53] (i.e., for Text-To-Speech APIs, and Maps API[53]). The Jersey library plays a key role in developing the RESTful web services for the function and parameter mappings of the incoming requests from the clients, as well as in producing and consuming data in various formats dynamically. In general, Jersey library is used to bind the web services with the Android application and mapping data into various object classes.

4.1 Sensing hardware configuration

Ambient sensing is performed using preconfigured sensors that are compatible with the IOT hub, i.e., door, motion, and multi sensors. Dense sensing is performed using bespoke configurations wherein Arduino Uno boards with XBee shields and modules are used to create a mesh network, see Refs. [54, 55] for more details. The main coordinator that receives data from the remote nodes is directly connected to the web server using comport. However, other options are also available to send the data from the coordinator to server, such as by using WiFi shields or Bluetooth. The remote nodes, which are connected with various multi-modal sensors, send their statuses to the coordinator when an event is triggered. In addition, Android mobile phone, Amazon Echo, and WeMo Sockets are also attached to the IOT router. The Android mobile phone is directly connected to the Amazon Echo via Bluetooth to output activity recognition results. In turn, the Amazon Echo can interact with the Almond+ router and with other popular sensing vendors. The WeMo Sockets and Amazon Echo can be easily integrated within the proposed mobile application using their APIs. Figure 5 presents the possible hardware configuration in order to start collecting the raw data.

4.2 Dataflow among the Android application, web service, and Apache fuseki server

The web service is central to the Android application and Apache Fuseki server. The Android application makes standard HTTP requests (i.e., GET, PUT, POST, and DELETE) to the Web Service to perform several tasks, such as CRUD operations, inferencing, reasoning, and other complex application-based logics. All the RDF data and ontologies are stored in the Apache Fuseki server as a graph. Therefore, the data are retrieved and manipulated by the web service using SPARQL query language with the support of Apache Jena library and the standard HTTP protocol. However, the real-time sensing data are exposed to the clients using a half-duplex, listener-subscription mechanism (i.e., SSE[43]) in comparison to full-duplex WebSocket. One of the key reasons for this decision is that the process intensive tasks of inferencing and reasoning are performed independently of the real-time event logging process.

The web service broadcasts two SSE methods to the clients: one for broadcasting real-time sensor events and another with inferencing results for the clients with a session token. This sequence of events between client device and the key components in the web service is illustrated in Fig. 6. As can be seen, the client Android application can listen to the sensor events in the background asynchronously by making an SSE call to "*EventBroadcaster*" function in the SensorsCall

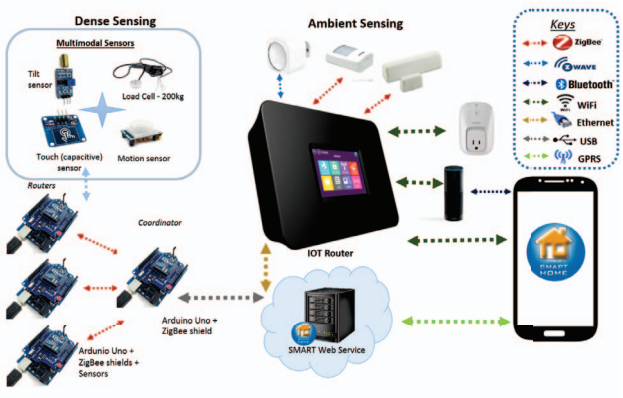


Fig. 5 Hardware: Connectivity diagram of sensing devices.

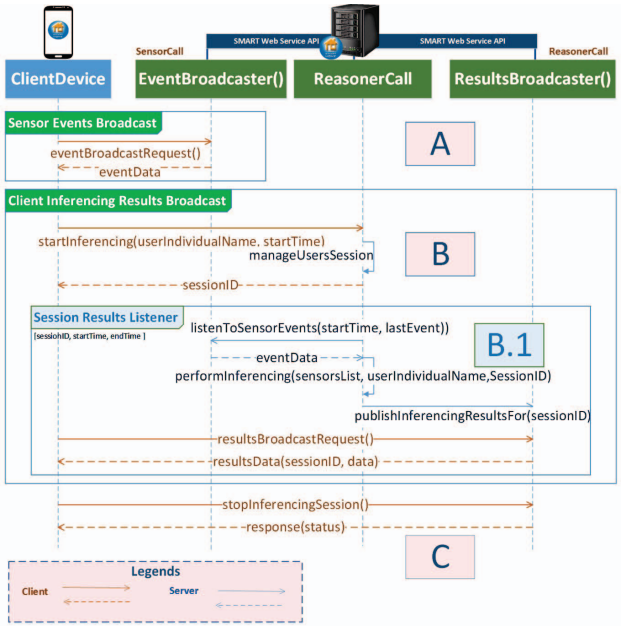


Fig. 6 SSE mechanism for real-time message flow of sensing and inferencing results between client and web service.

class located in "*SmartWebServiceAPI*" (A). To receive client-specific inferencing results, the client must obtain the session identity from the "*ReasonerCall*" first (B). The "*ReasonerCall*" is responsible for the task of listening to the sensor events from the given time, performing inferencing and then broadcasting the result using "*ResultsBroadcaster*" function (B.1). Once the client receives the session token, a request can be made to "*ResultsBroadcaster*", after which the task of listening to the inferencing results associated to their session identity is initiated. Meanwhile, the client device is responsible for closing the session (C) and, if required, storing the session data separately.

The web service performs a query and an update request in three simple steps: (1) building SPQARL query/update string, (2) using Jena classes/standard HTTP post methods to execute the request, and (3) parsing the responses. The pseudocode, shown in Fig. 7, performs a simple SPARQL query on the local Fuseki server end point and parses the result using the ResultSet and QuerySolution method. The standard HTTP post request can be made to perform SPARQL update using the HttpPost, HttpClient, and HttpResponse classes. However, the request content type is set to "*application/sparql-update*", and a static variable already defined in the Jenas WebContent class ("*WebContent.contentTypeSPARQLUpdate*") can be used.

```
// 1) Building Query String (Inc. all the prefixes for the vocabularies)
String queryStr = DataDictionary.PREFIX_DEFAULT
        + "SELECT * "
        + "WHERE { ?class rdfs:subClassOf :Sensor. }"
        + "LIMIT 100";

//2) Using Jena query to execute the SPARQL query
Query query = QueryFactory.create(queryStr);
QueryExecution qexec =
QueryExecutionFactory.sparqlService("http://localhost:3030/ds/query",
query)
ResultSet rs = qexec.execSelect();

//3) Iterate through the result using column names, in this case 'class'
while (rs.hasNext()) {
   QuerySolution soln = rs.nextSolution();
   //populate all the columns
   for (int i = 0; i < columnNames.size(); i++) {
       RDFNode rnode = soln.get("?" + columnNames.get(i));
   }
}
```

Fig. 7 Pseudocode for executing a SPARQL query on the server endpoint using Jena API.

Next, the Android application makes the requests to the web service using the standard HTTP protocols (HttpGet, HttpPost, and HttpPut, HttpDelete), only in a JSON format; hence, the request headers must be set appropriately. The Android application parses the JSON data, and by using the "*org.codehaus.jackson.map.ObjectMapper*" class, the data can be automatically remapped into their respective class instances.

4.3 Ontology modeling and data structuring

An ontology editing tool, such as Protege[52], can be used to build a conceptual model at varying levels of abstraction, leading to encapsulation of a particular set of knowledge. Then, while structuring and adding metadata to the raw data, these ontologies can be used across various domains as a vocabulary. In this way, the dataset is semantically enriched and the reusability of the data is increased, along with the improved ability to infer additional data, and perform reasoning using real world axioms[10]. However, a problem that must be solved at this point is the existence of multiple events or activities associated with one single instance.

A few possible solutions are considered, one of which is by simply linking the activity/event instances directly to the main instance using object properties. This could work, but it would create a large number of instances that would still be unstructured in terms of instance data grouping, ordering, and visualization. In turn, this could increase the querying complexity and create unnecessary computation overhead as the system data grow.

Another approach is a bucket-based one, similar to a table-like structure in any relational database, in which all the data can be associated to the bucket

instance using object properties at various inherent levels (see Fig. 8). For instance, Patient1 individual can have an object property of "*hasAppointment*" and a value as an object instance of "*Patient1_Appointments*" (bucket). This bucket, "*Patient1_Appointments*", can have *N* appointment object instances as a value, such as "*Patient1_Appointments_10_10_15*", which is defined using the sub property of "hasAppointment" object property called "*hasAppointmentItem*" (see Fig. 9(1)). The individual, "*Patient1_Appointments_10_10_15*", will hold all the relevant data required for the appointment, such as date and time of the appointment, location, and notes. This process can be repeated to represent other application scenarios, such as medications lists, notifications, and other user-specific preferences (see Fig. 9(2)).

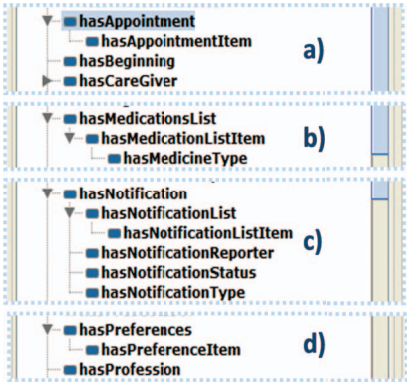


Fig. 8 Layered object properties for bucket-based structure data.

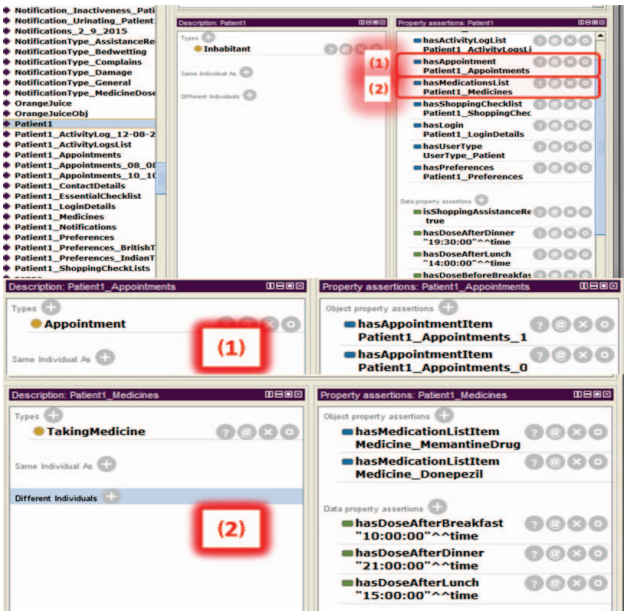


Fig. 9 Bucket-based approach for data structuring using object properties.

4.4 SPARQL-based inferencing

In order to perform activity assistance in ADL, a simple simulated environment is created to enable various sensors and view the activity recognition results (see Fig. 10b and Fig. 11); here, the Text-to-Speech feature is also used for the resulting output. The activity recognition algorithm is performed by the web service using a data-driven approach. Currently, only pre-defined user preferences (shown in Fig. 10a and Fig. 12 for the preference management interface) are used to match against the activated sensors. The aim of the matching process is to find the related user preference(s) and other inactivated sensor object(s) from the matched individual preference(s) to complete the activity. For this to be carried out, the current implementation uses SPARQL queries using the steps defined below. Some examples illustrated in Fig. 13.

Step 1 Find a user preference that has all the activated sensor objects and does not contain additional

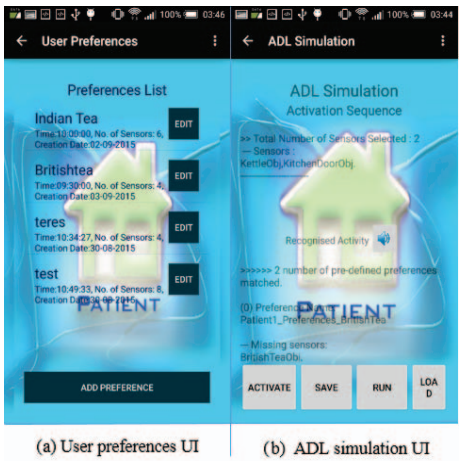
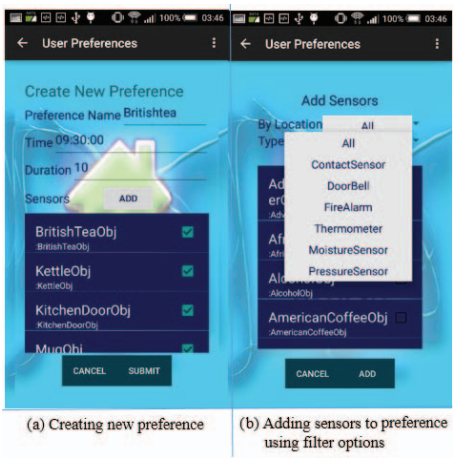


Fig. 10 Managing user preferences and ADL simulation mode interface.

Fig. 11 ADL simulation result of two possible preferences with their missing sensors to complete the activity.



(a) Creating new preference (b) Adding sensors to preference using filter options

Fig. 12 User preference management interface in action.

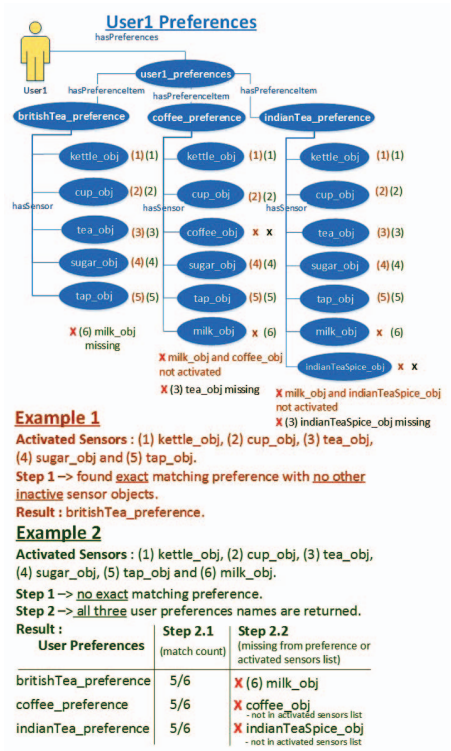


Fig. 13 Illustrating the inferencing steps taken using SPARQL query language.

sensors objects in the same preference.

Step 2 Otherwise, N user preferences are returned, which have all or some activated devices listed in a particular preference and other inactive sensor objects.

(1) The number of activated sensor object(s) exist in each user preference is taken and ordered in a descending order.

(2) Using the results obtained, the search for the missing sensor object(s) is carried out by inspecting the individual user preferences. The matched sensor object from the individual user preference is excluded by using the key functions, such as FILTER, Logical & Comparisons, or Conditional SPARQL operators[18].

One of the advantages of this SPARQL query based approach is that it does not require model loading or reasoning libraries. However, this approach does require explicit relationships to be defined in the dataset. To bridge this gap, the notion of SPARQL Inferencing Notation (SPIN) can be used to create rules, constraints, and functions in SPARQL syntax, which can be executed on the triplestore. SPIN is also known as SPARQL rules; for more information, see Refs. [56, 57].

4.5 Additional application scenarios

The Android application currently provides a simple login mechanism that directs users to different interfaces depending on their user types, i.e., Patient, Carergiver, Administrator, and System Manager. The user-specific interface allows the user to navigate through different activities. Figure 14a shows the patient's UI and Fig. 14b shows one of the features that allows patients to manage their medication and dose timing records. The UI and other features for other users will be further developed in the successive prototypes.

5 Experiment and Discussion

5.1 Experiment details

The proposed system implementation is tested by measuring the time between sensor activation and generation of inferencing results on the client device. The sensor activation time is only taken into consideration once the data are received by the web service. This is to reduce the effort for time synchronization between the sensing devices.

(a) Patient main UI (b) Patient medication and dose management UI

Fig. 14 Patient's main menu and UI of managing medicines doses.

A fixed time window length is defined for six User Activity Preferences (UAPs) that are listed and tested with three different scenarios, see Tables 2 and 3. The first scenario (TP1) activates the exact number of sensors defined in the user preferences, the second scenario (TP2) shows the activation of additional sensors objects, and the third scenario (TP3) shows a simulation of faulty sensors by using some sensor objects that are missing or not activated. The scenarios for the first two activities are illustrated in Table 4. Overall, each of the six activities are executed with three different scenarios by two actors (Exp).

The web service was deployed on the HP Z440 workstation with Intel(R) Xeon(E) v3 3.50 GHz processor with 16 GB RAM. The mobile application was tested on a Samsung S6 edge smartphone running Android 6.0.1 OS. The sensing data were collected using several touch sensors and door contact sensors using different protocols defined in Section 4.1.

5.2 Results

The results in Table 5 indicate that on average, it takes 4477 ms to receive the inferencing result on the mobile phone for all six User Activity Preferences (UAPs) with three different scenarios executed thrice. Overall, the results show little to no correlation between the number of sensors in the UAPs and the average time taken for inferencing and then communicating the results to the user.

5.3 Discussions

Although this paper does not focus on proposing activity recognition approaches, further changes in the system are still required to utilize the full capabilities of OWL and DLs. One of the current limitations of the defined SPARQL-based inferencing approach is that assertions (ABox), or instances, are mainly used rather than the terminology (TBox). The term TBox refers to the concepts and roles that are defined as vocabulary, whereas the ABox are named individually for those instances of a concept[10]. This allows the vocabularies to be generalized, shared, and applied across domains. In addition, the AR process can be enriched by investigating the process of dynamically separating and segmenting using these shared vocabularies and personalized rules/preferences. Another key difference in the proposed system is that all the data are stored in the triplestore and all open-source hardware and software components are utilized.

The HCI with the system also plays a key role gaining further benefits from the system's capabilities. The system implementation uses a mobile application; however, our society is moving towards more natural and ubiquitous HCI. Other systems[34, 36], discussed in Section 2, have already adapted the notion of augmented reality to overlay instructions on the camera or use natural gesture-/voice-based HCI. In comparison to the SMART system implementations and other systems discussed in Section 2, mainly having a web-browser based interface, this may limit the client devices from further utilization, unlike with mobile devices with embedded sensor capabilities to collect meaningful and contextual data. In addition to embedded sensors within the mobile device, instead of configuring additional dense or ambient sensors in the SH enviornment, more external sensors can be directly attached to a mobile device using any standard

Table 2 User activity preferences with the associated total number of sensor objects.

Activity number (#)	UAP	Sensor objects sequence	Total number of sensors
1	MakeIndian Tea	KitchenDoor1, KitchenCupboard1, TeaBagJar, IndianTeaSpiceJar, SugarJar, Kettle1, KitchenWaterTap1, Fridge1, MilkBottle1,	11
2	MakeCappuccino Coffee	KitchenDoor1, KitchenCupboard1, CappuccinoBagJar, SugarJar, Kettle1, KitchenWaterTap1, Fridge1, MilkBottle1, EatingSpoon1, Mug1	10
3	MakeStawberry Juice	KitchenDoor1, KitchenCupboard1, JuicerMixerCup1, SugarJar, KitchenCupboard2, ChoppingBoard1, Knife1, Fridge1, StawberryPacket1, MilkBottle1, KitchenWaterTap1, GlassCup1, JuicerMixer1,	13
4	MakingChips AndBeans	KitchenDoor1, FridgeFreezer1, ChipsBag1, KitehenCupboard2, OvenTray1, HeinzBakedBeansCan1, KitchenWaterTap1, MicrowaveBowl1, OvenDoor1, MicrowaveDoor1, CeramicPlate1	11
5	MakePasta	KitchenDoor1, KitchenCupboard1, PastaBag1, PastaPot1, KitchenWaterTap1, WoodCookingSpoon, PastaSauce, SaltBottle1	8
6	TakingMedicine	KitchenCupboard1, MedicineContainer1, GlassContainer1, KitchenWaterTar1	4

Note: Items in red color, Changes in object(s) from previous activity.

Table 3 AR test scenario types.

Scenario types	Exact no. of sensors	Extra sensors activation	Faulty/ missing
TP1	√	×	×
TP2	×	√	×
TP3	×	×	√

Table 4 Two examples of AR test cases.

Activity number (#)	Examples of tests specifications
1	TP1: #1, TP2: #1, add KitchenCupboard2 and GlasCup1. TP3: #1, swap TeaBagJar and OvenDoor1.
2	TP1: #2, TP2: #2, add KitchenCupboard2 and GlasCup1. TP3: #2, replace Mug1 with GlassCup1.

Table 5 Results showing average activity inferencing duration from the last activities recorded.

Activity number (#)	Test type	Exp1 (ms)	Exp2 (ms)	Exp2 (ms)	Avg. (ms)	Avg. per activity number (ms)
1	TP1	3890	3988	5127	4335.00	4472.33
	TP2	5175	4176	4802	4717.67	
	TP3	4172	4145	4776	4364.33	
2	TP1	4013	3953	4439	4135.00	4287.67
	TP2	4131	4135	4725	4330.33	
	TP3	4275	4288	4630	4397.67	
3	TP1	3926	3923	4353	4067.33	4410.56
	TP2	4303	4316	4571	4396.67	
	TP3	5310	4225	4768	4767.67	
4	TP1	4116	4175	4452	4247.67	4636.33
	TP2	6330	4474	4695	5166.33	
	TP3	4410	4461	4614	4495.00	
5	TP1	4150	4265	4409	4274.67	4584.11
	TP2	4446	4414	5919	4926.33	
	TP3	4497	4533	4624	4551.33	
6	TP1	4166	4801	4271	4412.67	4473.56
	TP2	4532	4556	4563	4550.33	
	TP3	4415	4460	4498	4457.67	
						4477.43

communication protocol[58].

The past system implementations with similar architectural styles and patterns have shown positive results in both functional and non-functional requirements; not only for AAL systems[59, 60]. However, finding suitable design patterns for a given application can be challenging and be easily misused[5, 7]. Nevertheless, several benefits of using a popular styles and pattern exist. One example is system maintainability, which can improve code compensation level and efficient debugging for the developer. Furthermore, the decomposed SOA architecture can enable any application to improve its scalability. In the case of the proposed system, additional sensing devices can be added within the SH so that the server can collect, process, and disseminate data to multiple clients more easily. Moreover, creating an opportunity to interact with other third-party services can help to extend the capabilities of the existing ubiquitous system.

6 Conclusion

This paper investigates some of the system architectural issues when building an AAL system. This was achieved by investigating some of the latest required components that can integrate and complement one another. A generic system architecture is proposed, which integrates and further extends the previous system implementations by introducing a lightweight, REST-based web service with an Android mobile application interface. The web service plays a key role in interacting with the triple-store (Apache Jena Fuseki server) endpoint, SH sensors, and mobile client applications. The web service provides activity inferencing and reasoning capabilities using Jena API; different reasoning engines can also be easily integrated. Moreover, this generic architecture uses simple design patterns (facade, repository, and domain for the web service and MVC for the Android application). The proposed system architecture also has open-source components that can be deployed in a distributed environment, making it scalable, as well as easy to use, maintain, and develop further.

A real-time system was implemented to illustrate the feasibility of the proposed architecture with some additional use case scenarios. The system leverages on the popular hardware components that are off-the-shelf and open-source. The real time testing results show that the average inferencing time taken to display the results to the user is 4477 ms on average. Finally, the implementation shows greater flexibility and potential for further development in terms of usability, ability to support additional application scenarios, and capacity to provide a greater scope of collecting personalized and contextual data (i.e., by paring wearable devices to the mobile phone and integrating other third-party APIs), thus increasing the accuracy of activity recognition.

The future implementations will focus on areas such as improving data modeling techniques, semantically

processing raw sensor data with an efficient timing mechanism[61], inferencing and reasoning activities with Jena API, as well as enhancing the SH sensing capabilities, performance optimization, and HCI methods (i.e., utilizing Amazon's Alexa voice services[62]). In addition, exploring rules (i.e., SPIN[57] and SWRL rules[63]), and Description Logics (DLs) capabilities instead of current SPARQL-based querying approach can be carried out. Finally, the system currently solves the problem of single sequential activity detection. The challenge of recognizing multiple or interweaving activities occurring concurrently in a non-sequential order is still being investigated. In this light, future works will focus on developing a framework or a mechanism that can support the ability to disentangle complex activities, i.e., recognizing that the user is making hot chocolate, taking medicine, or speaking on the phone simultaneously.

Acknowledgment

The authors gratefully acknowledge the contributions of Simon Forest for the implementation, deployment, and testing of the system. This project was partially supported by EU H2020 Marie Sklodowska-Curie Actions, ITN-ETN (ACROSSING Project ID: 676157) and Research Investment Fund, DMU.

References

[1] X. Zhang, H. Wang, and Z. Yu, Toward a smart home environment for elder people based on situation analysis, in *2010 7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*, 2010, pp. 7–12.

[2] R. Sterritt and C. Nugent, Autonomic computing and ambient assisted living - extended abstract, in *Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on*, 2010, pp. 149–151.

[3] D. Triboan, L. Chen, and F. Chen, Towards a mobile assistive system using service-oriented architecture, in *2016 IEEE Symposium on Service-Oriented System Engineering Towards*, 2016, pp. 187–196.

[4] G. Bohme, *Invasive Technification: Critical Essays in the Philosophy of Technology*. Bloomsbury Publishing, 2012.

[5] L. Pavlic, M. Hericko, and V. Podgorelec, Improving design pattern adoption with ontology-based design pattern repository, in *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, 2008, pp. 649–654.

[6] M. Ali and M. O. Elish, A comparative literature survey of design patterns impact on software quality, in *Information Science and Applications (ICISA), 2013 International Conference on*, 2013, pp. 1–7.

[7] C. Zhang, D. Budgen, and S. Drummond, Using a follow-on survey to investigate why use of the visitor, singleton & facade patterns is controversial, in *Proceedings of the ACM—IEEE International Symposium on Empirical Software Engineering and Measurement—ESEM'12*, 2012, pp. 79–88.

[8] L. Chen, J. Hoey, C. D. Nugent, J. D. Cook, and Z. Yu, Sensor-based activity recognition, *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 6, pp. 790–808, 2012.

[9] A. Ameen, K. U. R. Khan, and B. P. Rani, Extracting knowledge from ontology using Jena for semantic web, in *2014 International Conference for Convergence of Technology(I2CT)*, 2014.

[10] S. Staab and S. Rudi, *Handbook on Ontologies*, 2nd Ed. Springer-Verlag, 2009.

[11] R. Culmone, M. Falcioni, R. Giuliodori, E. Merelli, A. Orru, M. Quadrini, P. Ciampolini, F. Grossi, and G. Matrella, AAL domain ontology for event-based human activity recognition, in *Mechatronic and Embedded Systems and Applications (MESA), IEEE/ASME 10th Intl Conf*, 2014.

[12] L. Chen, C. Nugent, and G. Okeyo, An ontology-based hybrid approach to activity modeling for smart homes, *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 1, pp. 92–105, 2014.

[13] D. Gaaevic, D. Djuric, V. Devedzic, and B. Selic, *Model Driven Architecture and Ontology Development*. Springer-Verlag, 2006.

[14] J. Davies, F. Harmelen, and D. Fensel, eds. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, 2002.

[15] S. Powers, *Practical RDF*. O'Reilly & Associates, 2003.

[16] Apache, An introduction to RDF and the Jena RDF API, http://jena.apache.org/ tutorials/rdf_api.html, 2016.

[17] W3C, OWL 2 web ontology language document overview, http://www.w3.org/TR/owl2-overview/, 2012.

[18] B. DuCharme, *Learning SPARQL*, 2nd Ed. O'Reilly Media, 2013.

[19] W. Pawgasame, A survey in adaptive hybrid wireless sensor network for military operations, in *2016 Second Asian Conference on Defence Technology (ACDT)*, 2016, pp. 78–83.

[20] X. Hu, L. Yang, and W. Xiong, A novel wireless sensor network frame for urban transportation, *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 586–595, 2015.

[21] P. Gaikwad, J. P. Gabhane, and S. S. Golait, A survey based on smart homes system using internet-of-things, in *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, 2015, pp. 330–335.

[22] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, Wireless sensor network virtualization: A survey, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 553–576, 2016.

[23] Amazon, Amazon echo, http://www.amazon.com/Amazon-

SK705DI-Echo/dp/B00X4WHP5E, 2016.
[24] Samsung, SmartThings, https://www.smartthings.com/compatible\discretionary{-}{}{}products, 2016.
[25] IFTTT, Recipes on IFTTT are the easy way to automate your world, https://ifttt.com/, 2016.
[26] M. S. Perez and E. Carrera, Time synchronization in Arduino-based wireless sensor networks, *IEEE Latin America Transactions*, vol. 13, no. 2, pp. 455–461, 2015.
[27] Samsung SmartThings, SmartThings shield for Arduino, https://shop.smartthings.com/#!/products/smartthings-shield-arduino, 2016.
[28] L. Chen, C. Nugent, and A. Al-Bashrawi, Semantic data management for situation-aware assistance in ambient assisted living, in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services- IIWAS '09*, 2009.
[29] L. Chen, C. Nugent, and J. Rafferty, Ontology-based activity recognition framework and services, in *Proceedings of International Conference on Information Integration and Web-based Applications & Services - IIWAS '13*, 2013, pp. 463–469.
[30] X. Wang, J. Wang, X. Wang, and X. Chen, Energy and delay tradeoff for application offloading in mobile cloud computing, *IEEE Systems Journal*, 2015. doi: 10.1109/JSYST.2015.2466617.
[31] D. Martn, D. Lpez de Ipia, A. Alzua-Sorzabal, C. Lamsfus, and E. Torres-Manzanera, A methodology and a web platform for the collaborative development of context-aware systems, *Sensors*, vol. 13, no. 5, p. 6032, 2013.
[32] B. Guo, D. Zhang, and M. Imai, Toward a cooperative programming framework for context-aware applications, *Personal and Ubiquitous Computing*, vol. 15, no. 3, pp. 221–233, 2011.
[33] P. N. Borza, M. Romanca, and V. Delgado-Gomes, Embedding patient remote monitoring and assistive facilities on home multimedia systems, in *2014 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 2014, pp. 873–879.
[34] T. Kistel, O. Wendlandt, and R. Vandenhouten, Using distributed feature detection for an assistive work system, in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 1801–1802.
[35] A. D. Paola, P. Ferraro, S. Gaglio, and G. Lo Re, Autonomic behaviors in an ambient intelligence system, in *2014 IEEE Symposium on Computational Intelligence for Human-like Intelligence (IEEE SSCI 2014)*, 2014.
[36] A. Reichman and M. Zwiling, The architecture of ambient assisted living system, in *IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems*, 2011.
[37] A. N. Khan, D. Rodrguez, R. Danielsson-Ojala, H. Pirinen, L. Kauhanen, S. Salanter, J. Majors, S. Bjrklund, K. Rautanen, T. Salakoski, et al., Smart dosing: A mobile application for tracking the medication tray-filling and dispensation processes in hospital wards, in *6th International Workshop on Intelligent Environments Supporting Healthcare and Well-being (WISHWell'14)*, 2014.
[38] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, Web services composition: A decade's overview, *Information Sciences*, vol. 280, pp. 218–238, 2014.
[39] G. He, S. Wu, and J. Yao, Application of design pattern in the JDBC programming, in *the 8th International Conference on Computer Science & Education (ICCSE)*, 2013, pp. 1037–1040.
[40] Apache Jena Fuseki, https://jena.apache.org/documentation/fuseki2/index.html, 2016.
[41] X. Hu, T. Chu, V. Leung, E.C.-H. Ngai, P. Kruchten, and H. Chan, A survey on mobile social networks: Applications, platforms, system architectures, and future research directions, *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1557–1581, 2014.
[42] Jersey, RESTful web services in Java, https://jersey.java.net/, 2016.
[43] Jersey, Server-Sent Events (SSE) support, https://jersey.java.net/documentation/latest/sse.html, 2016.
[44] Apache, Jena ontology API, https://jena.apache.org/documentation/ontology/, 2016.
[45] M. Ayad, M. Taher, and A. Salem, Real-time mobile cloud computing: A case study in face recognition, in *2014 28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2014, pp. 73–78.
[46] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges, *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 337–368, 2014.
[47] Z. Li and K. Yap, Context-aware discriminative vocabulary tree learning for mobile landmark recognition, *Digital Signal Processing*, vol. 24, pp. 124–134, 2014.
[48] Shimmer, Shimmer sensing, http://www.shimmersensing.com/, 2015.
[49] Libelium, Waspmote plug & sense, http://www.libelium.com/products/plug-sense/, 2013.
[50] Care Quality Commission, About us, http://www.cqc.org.uk/content/about-us, 2016.
[51] K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer, Comparison of reasoners for large ontologies in the OWL 2 EL profile, *Semantic Web*, vol. 2, no. 2, pp. 71–87, 2011.
[52] Stanford University, A free, open-source ontology editor and framework for building intelligent systems, http://protege.stanford.edu/, 2016.
[53] Google, Products, https://developers.google.com/products/, 2016.
[54] R. Faludi, *Building Wireless Sensor Networks*, 1st Ed. O'Reilly Media, 2010.
[55] T. Igoe, *Making Things Talk*, 2nd Ed. Maker Media, Inc, 2007.
[56] G. Meditskos, S. Dasiopoulou, E. Vasiliki, and I. Kompatsiaris, Sp-act: A hybrid framework for complex activity recognition combining owl and sparql rules, in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2013, pp. 25–30.
[57] W3C, SPIN — Overview and motivation, http://www.w3.org/Submission/spin-overview/, 2011.

[58] R. K. Lomotey and R. Deters, Sensor data propagation in mobile hosting networks, in 2015 *IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2015, pp. 98–106.

[59] W. Dai and V. Vyatkin, A component-based design pattern for improving reusability of automation programs, in *IECON Proceedings (Industrial Electronics Conference)*, 2013, pp. 4328–4333.

[60] X. Xu, Y. Tao, X. Wang, and X. Ding, Research on architecture of smart home networks and service platform, in 2014 *5th International Conference on Digital Home (ICDH)*, 2014, pp. 232–236.

[61] L. Chen, C. D. Nugent, and H. Wang, A knowledge-driven approach to activity recognition in smart homes, *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 961–974, 2012.

[62] Amazon Developer, Alexa — Build engaging voice experiences for your services and devices. http://developer.amazon.com/public/solutions/alexa, 2016.

[63] W3C, SWRL: A semantic web rule language combining OWL and RuleML, https://www.w3.org/Submission/SWRL/, 2004.

Darpan Triboan is currently a PhD student at De Montfort University after receiving the MSc degree in software engineering in 2015 and BSc degree in 2014 from the same university. His current research interests include semantic and knowledge representation, Wireless Sensor Networks (WSNs), and pervasive computing and Ambient Assisted Living (AAL).

Liming Chen is a professor of computer science and the Head of the Context, Intelligence, and Interaction Research Group (CIIRG) of the School of Computer Science and Informatics at De Montfort University, UK. He received the BEng and MEng degrees from Beijing Institute of Technology (BIT), Beijing, China, in 1985 and 1988, respectively, and the PhD degree in artificial intelligence from De Montfort University, UK, in 2003. He worked as a senior research fellow at the School of Electronics and Computer Science (ECS), University of Southampton, and then a lecturer, a senior lecturer, and a reader at the School of Computing and Mathematics, University of Ulster, before he took the Professorship at De Montfort University. He has extensive research expertise and a wide range of research interests in areas such as artificial intelligence, semantic and knowledge representation, pervasive computing, and Ambient Assisted Living (AAL).

Feng Chen received the BSc, Mphil, and PhD degrees from Nankai University, Dalian University of Technology, and De Montfort University in 1991, 1994, and 2007, respectively. He is now a senior lecturer at De Montfort University. His research interests include software engineering, distributed computing, knowledge engineering, and image processing.

Zumin Wang is a professor of computer science at Dalian University, China, and the Head of Dalian Key Lab. of Smart Medical and Healthcare. He received the MEng degree from North China Institute of Technology, Taiyuan, China, in 2004, and the PhD degree from the Institute of Electronics, Chinese Academy of Science, China, in 2007. He has worked as a lecturer, an associate professor, and a professor at the College of Information Engineering, Dalian University. His current research areas include internet-of-things, software engineering, and wireless sensor networks.