Service identification guideline for developing distributed embedded real-time systems

R. Mohamad  M.W. Aziz  D.N.A. Jawawi  M. Ghazali  M.Z. Arbaie  N. Ibrahim

Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia UTM, Skudai, 81310 Johor, Malaysia
E-mail: radziahm@utm.my

Abstract: Analysis and design are the key phases in any software development life cycle. Service identification is one of the most important steps performed in the analysis phase of service-oriented software development life cycle. Service-orientation has been applied successfully in the domain of distributed embedded real-time system because of the advantages it provides such as reusability and interoperability. However, a systematic service-oriented development methodology for distributed embedded real-time system is still missing. A step-by-step guideline for service identification is presented in this study as part of an attempt to define the analysis phase of service-oriented software development life cycle for distributed embedded real-time system. The Smart Home case study is applied to verify the guidelines. This study provides profound descriptions on how to identify services for distributed embedded real-time system. Moreover, this work can be used as a first step towards a systematic service-oriented software development life cycle for distributed embedded real-time systems.

1 Introduction

Service-oriented computing (SOC) is becoming an established paradigm [1] for application development. In SOC, different services are combined together to develop a new application, where a service (set of tasks) is an autonomous platform-independent unit of functionality. A service has defined interfaces, through which the calling application can call and use them, without knowing their internal details. The service itself does not know about the calling applications [2].

Although the distributed embedded real-time systems (DERTS) are used everywhere in the society ranging from defence to industrial and home automation, their development is very complex as compared to enterprise software. This is because of their unique characteristics, resource constraints, temporal guarantees and an increasing shift of functionality and complexity from hardware to software. The lack of sound methodology and software engineering practices further increases the complexity.

SOC provides more advantages compared to previous development paradigms (like object-orientation and component-based software engineering), for example, commonality of functionality among several clients, Publish/Discover paradigm, dynamic composition and exchange of documents between services etc. This makes SOC more favourable for the development of DERTS. Thus, it has been applied in embedded real-time system development [3–6], at the device level for integrating the devices, especially the embedded devices [4], for embedded computing systems [7] and in distributed embedded systems field [8].

However, a systematic development methodology for DERTS based on services is still lacking. Systematic software engineering approaches that can manage the complexity of the system and the development are required for distributed embedded systems, as mentioned in [9]. Furthermore, the state-of-the-art methodologies, techniques and tools available for the embedded systems development are unsatisfactory and are many years behind the traditional software development [10]. Also the need for a complete service-oriented architecture (SOA) methodology is stated in the literature [11]. Thus, more research is needed on embedded software engineering methods.

Service identification is a vital part of the service-oriented software development life cycle (SO-SDLIC), because a service needs to be identified first before it can be designed. Regardless of the amount of information provided, almost all SOA methodologies discuss the service identification process. Among the different methods proposed for service identification some methods follow the traditional procedures of analysing the business requirements and identifying the services from these requirements (Top-Down approach); while others identify the services from the existing information systems (Bottom-Up approach). However, in almost all the methods, the services are derived through identifying the business process or system functionality. For example, the service identification in [12–16] is based on business process modelling.

In this paper, a systematic guideline to identify services is proposed that is based on device-centric and task-centric operations. Moreover, it also supports business process models in case the devices and the task they perform are part of a larger business process for example in industry. The proposed service identification guideline is applied in a Smart Home case study, in which there is no proper business process model to follow. It is important to note
that the focus of this paper is to identify the functionality provided by devices, therefore the way how a device provides its functionality, whether based on binary logic or fuzzy logic, is not a concern and hence not considered. The paper is organised as follows: Section 2 contains the methodology for developing the guideline and the introduction of the Smart Home case study used. The service identification guideline for DERTS is presented in Section 3. Section 4 provides the results of implementing the guideline on the Smart Home case study and discussion is provided in Section 5. Finally, Section 6 concludes the paper.

2 Methodology

Research has been done on service identification during enterprise service-oriented software development. Although some of the existing service identification guidelines, such as the guidelines in [14, 15] are closest to the proposed guideline, they are not suitable for DERTS as they lack device considerations and are based on business processes. Furthermore, these service identification guidelines are for enterprise software development and the identification of services are done with business process rather than from devices. In order to have a step-by-step guideline for service identification in DERTS development, the existing service identification guidelines for enterprise software development were used. These guidelines were identified from an extensive study of the literature and were selected for investigation based on the following criteria:

- The chosen guideline has been applied or tested on case studies.
- The chosen guideline has been referenced by other researchers in the field.
- The chosen guideline satisfies the definition of a ‘software engineering methodology’, as it assists in the process of software development.

In this work, a service identification guideline for DERTS is proposed, while keeping the device considerations in mind. Thus, the proposed guideline is most suitable for service identification during service-oriented DERTS development. Finally, the proposed guideline was refined by applying it in the case study of a Smart Home. As mentioned before, the entire methodology can be seen in Fig. 1.

2.1 Smart Home case study

A Smart Home is a home in which all the electrical devices are connected together, can work automatically and serve its inhabitants [17–20]. The Smart Home case study was selected for this work because it portrays a DERTS: a Smart Home consists of several embedded devices distributed across a home and communicating in real-time.

One of the fundamental problems of Smart Home is heterogeneity in terms of devices, vendors, communication protocols and technologies [17, 21]. Furthermore, the architecture of a Smart Home is traditionally server-centric. The home appliances are connected and controlled by the home gateway (server) [17, 22]. However, the problem with server-centric architecture is its limited interoperability (owing to the incompatibility of appliances using different technologies), scalability (owing to updating of server for every new appliance), extensibility (owing to tight-coupling of appliances and server) [17], reliability (in the case of central server failure) and load concentration (workload of the server becomes heavier as the number of appliances increases) [23]. A better architecture like SOA can solve these problems [17, 22, 24].

The devices used in the case study are limited to a certain number in order to have a better understanding of SOA concepts in general and the service identification process in particular. Thus, the case study is limited to six devices: telephone, television (TV), light, oven, fridge and an air conditioner (AC). The interaction between the devices, which shows the complexity of the functionality, is defined as follows: when a landline call is received while the TV is ON, the TV will automatically reduce the volume. The volume will be increased back to normal when the call is ended [17]. Similarly, lights become dim when the TV is ON and brightened to the normal level otherwise. Radiofrequency identification (RFID) tags are used to identify and cook the food in the microwave oven [25]. An appropriate message is sent to the user’s cell phone and TV when the food is ready. The fridge can send a message to the user and order the item to the retail store if the weight of an item is less than a specified value or if any food item has expired. Similarly, the AC is controlled by a temperature sensor, which gives messages to the AC either ‘Slow’ (to decrease the temperature) or ‘Fast’ (to increase the temperature).

As for this case study it is assumed that each appliance has device control interfaces (that can be software accessible), some storage (to store device control software application), a processor (to execute the application) and a network interface. The design complexity of the software increases by embedding it into the resource constrained devices. Furthermore, the case study used has its own timing complexities like any other real-time system, which can be divided into Hard and Soft real-time requirements. For example, the cooking of food in the oven has a hard real-time requirement of sending the SMS to the user within 1 min to avoid the food getting burned. On the other hand, monitoring of the environment by the temperature sensor and checking of the food item by the Fridge after every 1000 ms are soft real-time requirements.

3 Proposed guideline

Service modelling is the process of identifying services and their operations [14]. Unfortunately, the systematic service development approaches and modelling notations are largely unavailable [26]. Although there has been a lot of discussions on the concept of SOA in the last few years, a unified methodological approach for identifying services has still not been presented [15]. In this work, services were categorised into three types to make service identification easier: Application services, Functional services and
Fig. 2  Three layers proposed architecture

Control services. The Application service represents the application logic provided or derived from a technology platform. The Functional service represents the functional logic and was further divided into two types: Task services and Device services. The Task service is process-specific and represents an atomic process logic, whereas the Device service represents the physical devices involved and acts as a service provider. The Control Service defines and controls the process of service execution (Orchestration).

Based on this classification, the proposed architecture was divided into three layers: Application Layer, Service Layer and Device Layer as shown in Fig. 2. The layered approach was followed to abstract physical devices from the application software. These three layers were developed in order to separate the physical hardware devices on one layer (Device Layer) and the application software service on the other layer (Application Layer). The intermediate Service Layer hides the physical hardware devices from the application layer by providing software services. This is to enable application software to view hardware devices as a set of services. Furthermore, the services in this layer also allow the hardware devices to interact with each other.

A service communicates with a device by using proprietary protocols. The communication between services, on the other hand, uses generic SOA protocols (XML, SOAP etc.). Thus, a common and standard way of communication between all devices was made possible and the device-specific details were hidden from the application software. Furthermore, the service orchestration can be performed at the device level so that the need for a central server was eliminated.

Physical device interfaces, for example a button with OnOff interfaces, have corresponding methods in the services at the Service Layer. These methods controlled the device interfaces through the use of proprietary protocols. The services at the Service Layer provides interfaces (software interfaces – SOA concept), which can be accessed by other services through generic SOA protocols.

Table 1 shows a step-by-step guideline for identifying services within a DERTS environment. The complete process is illustrated by the OMG standard Software Process Engineering Meta model approach as shown in Fig. 3, which is then followed by the detailed explanation of the steps.

Step 1: Develop an initial architecture. In the first step, all the communicating devices of the system under study need to be identified. An initial architecture would be developed by separating the identified devices of the system under study in the Device Layer from the software application(s) in the Application Layer. The Service Layer would be empty in this initial architecture as no services are identified yet. However, at the end of the application of this guideline the Service Layer would contain all the identified services. Another advantage of developing an initial architecture is to achieve separation of concern.

Step 2: Identify candidate operations of the entire system. After developing an initial architecture, the next step is to identify the candidate activities and tasks (units of work) present in the system under study. A system sequence diagram can be used to identify the operations involved in completing the process. The system sequence diagram would not only help in identifying the communication and dependencies among the devices, but would also highlight the devices that are only using the services of other devices without providing their own services. In order to show their participation such devices are also represented as a service and are referred to as DeviceService. The name of a DeviceService would be the same as the name of the device. Moreover, the business requirements can be extracted from the system sequence diagram that would help in following the ‘Top-Down’ service identification approach.

Step 3: Group the operations into logical units (candidate services). In order to group the candidate operations identified in Step 2, one or more logical context needs to be determined. The candidate operations can be grouped together according to the action they perform, which is based on any of the following operation categories:
1. LowHigh
2. Send
3. Check
4. Order
5. Read
6. OnOff

The grouping can also be based on other actions like receive, print etc., which can be added to the list based on the domain in which this guideline is applied. In complex situations where the above categories cannot be used, new categories can be introduced. The atomic operations which cannot fit into any of the group can be represented as a separate service. All operations related to any of the categories are grouped together in one logical group. The group categories are designed in such a way that for most of the time, it is not possible for any operation to be part of more than one group, for example, an operation cannot be in ‘OnOff’ and ‘LowHigh’ groups.

<p>| Table 1  Guidelines for service identification for DERTS |</p>
<table>
<thead>
<tr>
<th>No.</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>develop an initial architecture</td>
</tr>
<tr>
<td>2.</td>
<td>identify candidate operations of the entire system under study</td>
</tr>
<tr>
<td>3.</td>
<td>group the operations into logical units (candidate services)</td>
</tr>
<tr>
<td>4.</td>
<td>make an initial list (inventory) of candidate services</td>
</tr>
<tr>
<td>5.</td>
<td>check existing assets for providing the identified operations</td>
</tr>
<tr>
<td>6.</td>
<td>refine the list of candidate services</td>
</tr>
<tr>
<td>7.</td>
<td>separate and group application operations and repeat Step 6</td>
</tr>
<tr>
<td>8.</td>
<td>separate and group device-centric operations and repeat Step 6</td>
</tr>
<tr>
<td>9.</td>
<td>separate and group task-centric operations and repeat Step 6</td>
</tr>
<tr>
<td>10.</td>
<td>introduce control service for composite services (for orchestration) and other supporting services and repeat Step 6</td>
</tr>
<tr>
<td>11.</td>
<td>identify dependencies of operations and/or services on any external service, separate and group them together and repeat Step 6</td>
</tr>
</tbody>
</table>
simultaneously. However, in complex cases, if one operation can be included in more than one group then it should be included in the group that is semantically closer to the operation. Nevertheless, this grouping of operations is not finalised as the process is iterative and operations would be regrouped in later steps again and again. Each of these logical units represents a candidate service. The collection of these candidate services along with DeviceServices should represent the system’s complete functionality. The grouping of operations into logical units allows these service areas to be looked at more generically. Thus, the identification of the relationship between services and the extraction of common support services is made possible.

Step 4: Make an initial list (inventory) of candidate services. An initial list of all the candidate services identified in Step 3 is made. This list would be modified as new services are identified and/or the operations are regrouped.

Step 5: Check existing assets for providing the identified operations. It is important to ensure that there exist some assets, which can fully or partially provide the functionality of the identified candidate service. The existing assets can be a middleware present on a device or any other legacy system component. If there are some assets present, then that asset should be analysed in detail to find which candidate services or its operations can be fulfilled by reusing the existing assets. The ‘Bottom-Up’ approach of service identification is followed by utilising these existing assets. Only the candidate services or their operations which are not provided by the existing assets need to be developed. The operations can be regrouped into candidate services if needed.

Step 6: Refine the list of candidate services. This step is to refine the list of candidate services developed in Step 4. This iterative step is repeated after any of the later steps.

Step 7: Separate and group application operations and repeat Step 6. The processing requirements of all the services are studied in detail in this step and the services depending on some software application are separated from the others. It is assumed that the software applications already exist and are provided by third party. Most of these applications provide technological facilities. In the case of more than one application providing the same facility, their selection is beyond the scope of this work. This step is only concerned with separating the application-dependent operations from the others. The list of services can be refined and if needed, the operations can be regrouped.

Step 8: Separate and group device-centric operations and repeat Step 6. Although the functionality provided by the devices is represented with the help of candidate services, the operations and/or services related to a particular device need to be separated. In this step, the operations related to a particular device are separated and rearranged into new groups, if required. Each device may have different properties (characteristics) to perform operations. These
properties can have clear (binary) values or they are vague (fuzzy). Indeed these properties may affect the way a device performs the operation. As mentioned earlier, explanation of how a device performs an operation is beyond the scope of this paper. However, it is worth mentioning here that by separating the operations into a device-centric group, the properties of the device can be represented for each group. Some new services can also be created to represent a device action. If two or more devices are performing the same operations, the operations of both the devices would be represented as separate services. The devices, which only use the functionality of other devices without providing their own, are exempted from this step because they are already represented as DeviceServices. Finally the service inventory is updated.

Step 9: Separate and group task-centric operations and/or services and repeat Step 6. All the operations are checked in this step in order to find the workflow between them, which would help in identifying the tasks. After identifying the tasks, the operations and/or services which are contributing in completing a task are grouped together replacing the existing groups. In order to complete a task it is possible for services provided by different devices to be combined. As many services may be involved in completing a single task, the result of this step could be one or more composite services. It is also possible that one service may be involved in completing several tasks; in that case the service would be modelled with its provider and not as part of a composite service. All composite services involving such a service would have a link to that service. The list of services is then updated after the existing services are rearranged and grouped into new composite services.

Step 10: Introduce control service for composite services (for orchestration) and other supporting services and repeat Step 6. In this step, all the identified services are inspected to identify the supporting services they need. If not already grouped, the related services are grouped together into composite services. Furthermore, the control services for orchestration (Orchestrator service) are also introduced into the service inventory. The Orchestrator service is responsible for handling the overall orchestration of atomic services within the composite service. It includes the list of all participating atomic services, the work flow of the activities and different types of Handlers, for example, exception-handler. During a service composition process, if a device (service provider) is not available (switch OFF or malfunctioned), the same service provided by other devices would be searched. If there is no other device providing the same service then the exception-handler would be used to cope with the erroneous condition.

Step 11: Identify dependencies of operations and/or services on any external service, separate and group them together and repeat Step 6. The candidate services are checked in this step to find out their dependencies on any service external to the system. If a dependency is found, the external service is also included in the candidate service list and the service inventory is updated accordingly. The selection of external service and its communication is beyond the scope of this work.

4 Application of the guideline to the case study

The guideline for service identification mentioned in Table 1 was used in this section for the Smart Home case study.

Step 1: Develop an initial architecture. An initial architecture was developed after identifying the devices and separating all the appliances (devices) of the Smart Home case study in the Device Layer from the software application(s) in the Application Layer as depicted in Fig. 4. The user can control the home appliances through the Application Layer. The Service Layer was left empty in this step as no services were identified yet. When they are identified, they would be placed in this layer.

Step 2: Identify candidate operations of the entire system. In this step, the candidate operations were identified by developing a system sequence diagram of the participating entities as illustrated in Fig. 5. The operations identified from the designed system sequence diagram are shown in Table 2. In this case study, only one device, the telephone, was found which was not offering any service. Thus it is treated as a DeviceService.

Step 3: Group the operations into logical units (candidate services). The operations identified in Step 2 were grouped together into the following logical units (candidate services) based on the identified operation categories: ‘Decrease/Increase Volume’, ‘Dim/Full Light’ and

Fig. 4 Initial architecture
"SlowFast AC" operations were grouped in LowHigh service. The sending of messages to cell phone, TV and email account was grouped together in Send service. ‘Check the food item’ is a composite task that involves two operations, that is, ‘Weight food item’ and ‘Read expiry date’ on the food item to check its expiry date. ‘Weight food item’ and ‘Check the temperature’ were grouped together in Check service. However, ‘Read expiry date’ operation can be included both in the Check service and the Read service which already include the ‘Read cooking instructions’. As ‘Read expiry date’ operation matches semantically closest to Read service, therefore it was included in the Read service. The only operation left ‘Order the food item’ was atomic and hence represented as a separate service: Order.

Step 4: Make an initial list (inventory) of candidate services. The identified operations were grouped into five candidate services in the previous step. The complete list including the DeviceService is as listed below:
1. Telephone (DeviceService)
2. LowHigh
3. Send
4. Check
5. Read
6. Order

Step 5: Check existing assets for providing the identified operations. Since there were no current assets (e.g. middleware) used for providing these services in the case study, the list of services remained the same.

Step 6: Refine the list of candidate services. As there was no existing asset and Step 5 did not update the services list in Step 4. Therefore the refinement of the list was not yet needed.

Step 7: Separate and group application operations and repeat Step 6. The Send service includes the message sending operations either to devices or software applications. In this step, the application-dependent operations were separated from the other Send operations into new services as shown in Fig. 6. In Send service, there were two types of operations which depended on some software application: ‘Send SMS to cell phone’ and ‘Send message to user email account’. They were collectively represented as ‘Send message to software applications’ service.

There is only one operation sending a message to devices that is ‘Send Message to TV for display’, therefore a new service (Display service) was introduced in the list, and the list of services was updated.
1. Telephone (DeviceService)
2. LowHigh
3. Send message to software applications
4. Display
5. Check
6. Read
7. Order

Step 8: Separate and group device-centric operations and repeat Step 6. The LowHigh service contained three operations all belonging to different devices: TV, Light and AC, each with different properties. The Send Message to software application service has related to the cell phone and the Internet. Similarly, Check service consisted of two operations belonging to Fridge and Temperature Sensor. The Read service also consists of two operations belonging to Oven and Fridge. The Fridge and the Oven both provide the operation of Send SMS, which was represented separately for both devices. All of these device-centric operations were separated as shown in the updated service inventory below:
1. Telephone (DeviceService)
2. LowHigh volume
3. LowHigh Light
4. LowHigh AC
5. Send SMS (food ready-provided by Oven)
6. Send SMS (food expires-provided by Fridge)
7. Send email
8. Display
9. Weight food Item
10. Check Temperature
11. Read expiry date
12. Read cooking instruction
13. Order

Step 9: Separate and group task-centric operations and repeat Step 6. For this step, a list of tasks needed to be identified first. The list of identified tasks is shown in Table 3.

The Check Temperature and LowHigh AC services are related to Task 3 so they were combined in a Composite service; Temperature Control. The Read cooking instructions service, Display service and Send SMS (food ready) service belongs to the same task (Task 4), thus they were combined into a new composite Cooking service. The Weight Food Item service, Send email to user account service and (Place) Order service belongs to the same task (Task 5), so they were also combined into a new composite service; Food Ordering. Similarly, the Read expiry date service and Send SMS (food expires) service are related to Task 6, therefore they were combined in Check Food composite service. These composite services can be seen in Fig. 7.

Step 10: Introduce control service for composite services (for orchestration) and other supporting services and repeat Step 6. Cooking, Food Ordering and Temperature Control services were all composite services and needed a control service to orchestrate them. Therefore an orchestrating service was added to each of these composite services, as shown in Fig. 8.

Step 11: Identify dependencies of operations and/or services on any external service, separate and group them together and repeat Step 6. The Order service in the Composite Food Ordering service depended on an external Order Processing service provided by the retail store. Thus, this external service was inserted in the list of services.

1. Telephone (DeviceService)
2. LowHigh volume
3. LowHigh Light
4. Composite Cooking
5. Composite Food Ordering
6. Composite Check food
7. Composite Temperature Control
8. Order Processing (external service from Retail store)

After service identification, these identified services were mapped to three layers proposed architecture. The result of the mapping is shown in Fig. 9. It is important to note that the Display service is involved with Low/High Light service and in Composite Cooking service, hence it is represented by its provider the TV and not as part of the Composite Cooking service. However, it has been linked from the Composite Cooking service.

Table 3  Identified operations tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>decrease TV volume while user attends a phone call, volume increases when the user finishes</td>
</tr>
<tr>
<td>T2</td>
<td>lights become dim while the user is watching the TV and full otherwise</td>
</tr>
<tr>
<td>T3</td>
<td>AC automatically becomes slow or fast based on the readings from the Temperature Sensor</td>
</tr>
<tr>
<td>T4</td>
<td>oven reads RFID tag on the food item, cooks and sends a message to the user when food is cooked</td>
</tr>
<tr>
<td>T5</td>
<td>fridge checks the weight of food item, orders to retail store and sends an email to the user</td>
</tr>
<tr>
<td>T6</td>
<td>fridge checks the expiry date of food item and sends a message to the user if the food is expired</td>
</tr>
</tbody>
</table>

Fig. 7  Formation of composite services

Fig. 8  Introduction of control services
5 Related work and discussion

A step-by-step process for service identification is provided in [14, 15]. The services are identified based on the business process in a ‘Top-Down’ manner in [14]. A set of complementary service identification techniques was used in [16]. The three main techniques used are Goal-Service Modelling, domain decomposition and existing asset analysis. The proposed guideline used both the ‘Top-Down’ and ‘Bottom-Up’ analysis techniques and provided a step-by-step guideline for service identification. The ‘Top-Down’ and ‘Bottom-Up’ analysis techniques for service identification are combined in [15] and service types and layers are identified. However, in [15] the input from the stakeholders is given more importance.

The service identification in [12, 13] on the other hand, are based on business process models. The services are derived from the As-is and To-be business processes. A step-by-step guideline for service identification is missing from [12, 13, 16] and their focus is on what to do rather than explain how to do it. The proposed guideline provides a step-by-step procedure for service identification through explaining the ‘How’ part. Furthermore, the guideline does not only depend on business process models, but it considers the workflow of tasks performed by devices in case of a missing business process model.

Moreover, all of these methods [12–16] of service identification deal with enterprise application development and they do not consider devices. Five works [12–16] were selected to be compared to the proposed guideline as they discuss the service identification process but with less or more information. All [12–14, 16] are complete service-oriented development methodologies where the service identification process is only a part of the methodology. Only [15] is a service identification approach and not a complete development methodology. A comparison of the proposed guideline and previous methods is presented in Table 4.

It can be observed from Table 4 that the proposed guideline for service identification follow both Top-Down and Bottom-Up approaches, is step-by-step and is based on device consideration. Furthermore, it also introduces different possibilities that may occur during the process.

The idea of SOA has been applied in the domain of Smart Home [17, 18, 21–25, 27, 28], in which appliances appear as an understandable service to others abstracting the hardware and communication details. In SOA for Smart Home, each appliance shows its services and their interfaces. So the appliances work autonomously, connected together in a loosely coupled fashion while communicating with each other in a standard way [23].

Table 4 Comparison of proposed guideline and other methods

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>based on BPM</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Top-Down</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bottom-Up</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Step-by-Step</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>enterprise software</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>device consideration</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
However, most of the work in this area is concerned with the implementation of service-oriented concepts in Smart Home and very little attention has been given to a systematic SO-SDLC. This paper focuses on the identification of services which is a vital activity in SO-SDLC.

The existing work is mostly focusing on the implementation and very few efforts have been made on the systematic modelling of the Smart Home. In [29] Architecture Analysis and Design Language is used to model the Smart Home, but the modelling is focused more on the physical level than the software. However, it provides a good idea of dividing the whole system architecture into three different layers. The concept of dividing the home appliance into two sub-layers, that is, hardware and service layers are presented in [21, 23]. Orchestration of services takes place at the service layer, thus eliminating the need for a central server [21, 23]. Similarly, the idea of separating the services at the device layer and control layer used in this research was inspired by [21, 24].

6 Conclusion

The development of DERTS is very complex and different from enterprise software. In order to reduce this complexity, new software engineering methods and techniques can be used for their development. Service-orientation has been used for the development of DERTS because of the advantages it provides. This paper proposed a step-by-step guideline for service identification in DERTS environment. Once the services are identified, it is easy to perform other activities of a systematic SO-SDLC. The proposed guideline used both ‘Top-Down’ and ‘Bottom-Up’ analysis techniques to identify services for DERTS, providing a way of identifying services during DERTS development. The guideline was then applied in a Smart Home case study to show its applicability. A comparison was made between some of the existing service identification techniques and the proposed guideline. The results showed that the proposed guideline is suitable for service identification during DERTS development. After the successful identification of services, the next step would be the modelling of the identified services.

7 References

6 Jammes, F., Smit, H.: ‘Service-oriented architectures for devices – the SIRENA view’. Third IEEE Int. Conf. on Industrial Informatics, INDIN’05, 2005
14 Erl, T.: ‘Service-oriented architecture: concepts, technology, and design’ (Prentice Hall PTR, NJ, USA, 2005), p. 792