Artificial Intelligence for Engineering Design, Analysis and Manufacturing

cambridge.org/aie

A scenario-integrated approach for functional design of smart systems

Fajun Gui[1] and Yong Chen[2]

[1]School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China and [2]School of Aeronautics and Astronautics, Shanghai Jiao Tong University, Shanghai, China

Research Article

Cite this article: Gui F, Chen Y (2021). A scenario-integrated approach for functional design of smart systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **35**, 165–179. https://doi.org/10.1017/S0890060420000487

Received: 30 May 2020
Revised: 10 September 2020
Accepted: 16 September 2020
First published online: 5 January 2021

Key words:
Functional design; functional modeling; Model-Based Systems Engineering; scenario; smart systems; SysML

Author for correspondence:
Yong Chen, E-mail: mechenyong@sjtu.edu.cn

Abstract

Functional design is regarded as a design activity primarily aimed at clarifying customer needs, and developing the functional architecture and solution concepts for a system under development. Existing functional design approaches are mainly focused on how to assist designers in searching for solution principles for desired products, which, however, do not adequately take into account the interactions between a smart system under development and its environment, and cannot explicitly represent the complex functional logic of the system, resulting in that they cannot effectively assist designers in the functional design of smart systems. Therefore, this paper proposes a scenario-integrated approach for functional design of smart systems to address the above issues. Based on the concept of scenario in software engineering, the proposed approach explicitly elaborates how to employ scenarios to express subjective customer needs and how to generate the functional architectures and the corresponding solution concepts through a structured process. The functional design of the automated doors-unlocking system of a smart vehicle is employed to illustrate the proposed approach, which also demonstrates that the proposed approach is suitable for functional design of smart systems.

Introduction

As a significant activity of system development, functional design is primarily responsible for analyzing customer needs and developing possible solution concepts for a desired system. Since functional design can provide a systematic approach for transforming functions into technical solutions, many engineering design researchers in engineering design community have carried out considerable functional design research, with the development of many valuable approaches (Cole, 1998; Suh, 2001; Pahl and Beitz, 2007; Erden *et al.*, 2008; Aurisicchio *et al.*, 2013; Booth *et al.*, 2015).

Although the existing functional design approaches have some merits, they still cannot effectively assist designers in the function design of smart systems. Hereby, a smart system often refers to the system that can interact with human users and systems in the environment, and adapt its behavior to achieve users' goals or expectations, which means that the functionality of a smart system not only depends on its internal solution principles, but also the interactions with human users and external systems in the environment.

However, most of the existing functional design approaches mainly focus on generating solution principles for desired functions, and thus do not adequately take into consideration the complex interactions between a system under development, human users and its environment. Another limitation lies in that the functional design process is usually simplified as the functional decomposition process in the traditional design theories (Pahl and Beitz, 2007; Erden *et al.*, 2008), which, however, do not take into consideration the complex functional logic between different subfunctions, and can make it difficult for designers to employ these approaches to address the functional design of smart systems. Therefore, it is becoming increasingly necessary to develop an effective functional design approach that can take into account not only the solution principles of system under development, but also the interactions between human users, objects in the environment, and the system under development.

Since scenario is a concept that is often used to document interaction sequences that are indispensable to achieve a specific goal in software engineering (Pohl, 2010), this paper attempts to develop a scenario-integrated approach for functional design of smart systems. This paper is organized as follows. The next section reviews the related work. In the section "Scenario and Its Representation," the concept of scenario is introduced and how to represent a scenario in a structured manner is elaborated. Subsequently, a scenario-integrated approach for the functional design of smart systems is proposed in the section "A Scenario-Integrated Approach for Functional Design." With an automated doors-unlocking system of a smart vehicle as an example, the section "Case Study" illustrates how the proposed approach can be employed to achieve the functional design of smart systems, followed by a discussion in the section "Discussion." Finally, the last section concludes this paper.

Related work

Since functional design is a crucial stage for system development, there has been a lot of research on the theory or methodology of functional design. Due to limited space, a brief review is given as below to illustrate the related work.

Among the various existing functional design approaches, the most significant one is the systematic design approach proposed by Pahl and Beitz (2007), which regards functional design as a process of decomposing an overall function into subfunctions, searching suitable solution principles for the subfunctions, and combining the solutions of subfunctions into a whole technical solution to achieve the desired function. Based on the functional decomposition idea, researchers have also developed some automated approaches for functional design. For example, Chakrabarti and Bligh (1996) have developed a function-based approach for generating principle solutions of mechanical devices; Campbell *et al.* (2003) have proposed the A-design approach to achieve the functional design of electromechanical systems, which has combined multiobjective optimization, multiagent systems, and automated design synthesis; Chen *et al.* (2012) have developed a knowledge-based framework for the functional design of multidisciplinary systems; Yuan *et al.* (2016) have proposed a hybrid approach to automate the process of functional decomposition of complex mechatronic systems to obtain function structure and solutions.

In addition to the above functional decomposition approaches, researchers have also developed some other functional design approaches. For example, Suh (2001) has proposed the axiomatic approach for functional design, where functional design is regarded as a zigzag mapping process between functional requirements and design parameters of solution concepts; Umeda *et al.* (1990, 1996) have proposed the Function–Behavior–State model, where the functions are decomposed into subfunctions until they can be associated with physical features, and knowledge of the decomposition of functions and alternative solutions can be retrieved from a knowledge base; Kitamura *et al.* (2004) have developed a knowledge-based function decomposition system named "function way server," which can support designers to decompose the function based on providing various decomposition solutions that will realize the goal; Mhenni *et al.* (2014) have proposed a Systems Modeling Language (SysML)-based methodology for mechatronic system architectural design, which can assist designers to collect requirements and identify candidate solutions and select the final physical architecture of the system.

It can be found that most of the existing functional design approaches deal with how to generate solution principles for a system under development, and do not involve interactions between a system under development and the environmental objects in the design process. Therefore, such approaches cannot effectively support the functional design of smart systems, which often involves various interactions.

Since smart systems are becoming more and more popular, there are also some research on the design of smart systems in recent years. A typical stream of such research is the context-aware design (e.g., Hong *et al.*, 2009; Engelenburg *et al.*, 2019). However, such context-aware design approaches are focused more on the context of systems and do not pay much attention to the functional design process. Therefore, it is still valuable to develop a comprehensive approach for the functional design of smart systems.

Scenario and its representation

The term *scenario* is a concept that is commonly used in software engineering, which software engineers often employ to describe the interactions between various users and software systems (Carroll, 2000; Kaindl, 2000; Alexander and Stevens, 2002). Since smart systems also involve complex interactions between systems, human users, and environments, this research attempts to import the concept of scenario into the approach for the functional design of smart systems. In this section, the concept of scenario will be introduced, together with the approach for representing scenarios.

Concept of scenario

In software engineering, the concept of scenario is proposed to express the interactions between user activities and a software system (Carroll, 2000; Kaindl, 2000; Alexander and Stevens, 2002). According to Pohl (2010), a scenario can be defined as a set of concrete interaction sequences of satisfying or failing to satisfy a goal (or set of goals). Therefore, a scenario is typically composed of a sequence of interaction steps executed to meet a goal and thus can illustrate the value of the system for its users. It is evident that a scenario can establish a link between the customer needs and the relevant context; therefore, a scenario can refine a customer need into more detailed description.

According to software engineering, a scenario can be represented with a set of interaction sequences, together with some contextual elements. Interaction sequences can refer to human operations that are exerted on a system of interest, the responsive events and activities of the system, and the interactions between the objects in the related environment and the system. The contextual elements of a scenario include *actors*, *goals*, *time*, *location*, *preconditions*, and *postconditions*. Hereby, an *actor* refers to a human user or an external system that will interact with the system of interest in the execution of the scenario; the element *goal* is an intention regarding an objective or usage of a system, which should be satisfied by the scenario; the element *time* tells when the scenario occurs; the element *location* defines the information about where it takes place; finally, a *precondition* refers to a state or condition that the system of interest should be in before a scenario is performed, while a *postcondition* indicates a state or condition that the system should be in after the execution of the scenario is completed. It should be noted that not all elements mentioned above is indispensable in a scenario, since some elements can be omitted when the scenario can be understood by engineers.

Based on the above analysis, the scenario, *braking control in a car*, can be described as below:

> A driver was driving a car on the motorway. When the driver noticed that the car in front of him was braking, and the distance between the front car and his own car was rapidly shortening, the driver stepped on the brake pedal, so that the safety distance between the two cars could be maintained to avoid a rear-end collision.

In the above scenario, the driver is an *actor*. The *goal* is to maintain the safety distance. Since the *time* element is not necessary for this scenario, it is omitted. The element *location* is defined as on the motorway. A *precondition* of the scenario is that the car in front of the driver was braking and the distance between the two cars was rapidly shortening, and a *postcondition* refers to the state that *the safety distance between the two cars is maintained.*

Since a goal can be satisfied by different means, multiple scenarios may exist for the same goal. These scenarios can often be classified into three types, that is, main scenario, alternative scenarios, and exception scenarios (Pohl, 2010). They can all achieve the same goal. To be specific, a main scenario describes the interaction sequences that are normally executed in order to satisfy a specific goal. An alternative scenario documents the interaction sequences that can replace the main scenario for satisfying the goal. An exception scenario documents an interaction sequence that is executed when an exceptional event occurs.

Scenario representation

Generally, a scenario is often informally described in natural language (Rumbaugh *et al.*, 1991; Jacobson, 1993; Alexander and Stevens, 2002). Although such natural language-based descriptions are convenient for designers to elicit and describe scenarios (Liu *et al.*, 2012), it is difficult for computers to understand and process them. Therefore, this research will employ a model-based approach for representing scenarios, which can be much easier for computers to process the scenario information. Since a sequence diagram in SysML (Friedenthal *et al.*, 2006, 2014) usually describes a sequence of message exchanges between the related actors and the system of interest, the sequence diagram is employed to represent the interactions in a scenario.

To represent a scenario with a sequence diagram model, the first task that should be fulfilled is to extract the contextual elements from the context of a scenario. As mentioned before, the contextual elements of a scenario primarily include *actors*, *goals, time*, *preconditions*, *and postconditions.* For example, in the scenario, *to navigate a car to destination*, the *actor* elements can be extracted as the driver, the global positioning system (GPS), and so on. A list of objects can be adopted to express the actors in the above scenario, that is, *Actors* = {driver, GPS, …}. The *goal* of this scenario is that the navigation system can guide the driver to drive his/her car to destination. Hereby, some contextual elements (e.g., the *time* element in this case) can be omitted since they are not necessary for engineers to understand the scenario. A complete set of the contextual elements of the above scenario can be represented in Figure 1.

After the contextual elements of a scenario are extracted, it is then necessary for designers to identify the interaction sequences in the scenario. To model each interaction in sequence diagram, the actors of each interaction step in the execution process should be specified, including the actors who invokes the interaction activities and those who are influenced by the actions of others. Then, all actors identified before can be represented as *lifelines* in a sequence diagram model. After the actors related to a system under development have been defined, designers should then determine the activity or behavior of each interaction step, that is, how the actors operate the system or how the system responds to the actors. For example, in the above scenario, *to navigate a car to destination*, one of the interaction steps can be identified as: *the driver inputs the destination*, which can be modeled as a *message* that points to the navigation system from the driver, as shown in Figure 2a.

Figure 2a shows the sequence diagram model of the main scenario, *to navigate a car to destination*, which represents the normal interaction sequence that is executed to satisfy the goal. As stated above, the goal can also be satisfied by other alternative scenarios (i.e., alternative interaction sequences). For example, for the interaction activity, *the driver inputs the destination with hand*, alternative solutions can be *voice inputs*, *keyboard inputs*, and so on, as illustrated in Figure 2b. As seen in the model, a combined fragment with an *alt* interaction operator is used to represent an alternative scenario. In addition, a combined fragment with a *break* interaction operator can be employed to illustrate an exception scenario. For brevity, examples of exception scenarios are not given in this section.

From the above example, it can be found that the scenario of *navigating a car to destination* can be explicitly represented with the sequence diagram models, each interaction step or activity between the driver, the navigation system, and the GPS is clearly defined.

Since complex interactions between a smart system, human users, and the external systems in the context are quite common when the smart system implements its functions, it is essential to take into account the interactions for the functional design process. Based on the above analysis, it can be found that the concept of scenario and its representation are very suitable for documenting and modeling the contextual information and interaction sequences of a smart system.

A scenario-integrated approach for functional design

Method overview

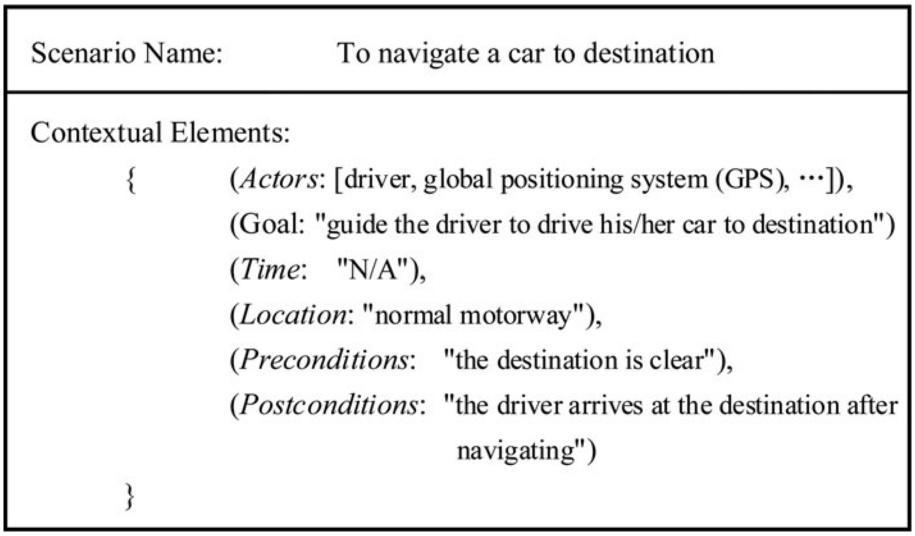
Since most of the existing functional design approaches do not address the complex interaction issue, they cannot provide effective support for the functional design of smart systems, which usually involves complex interactions and functional logic. Therefore, it is necessary to develop a suitable approach for supporting the functional design of the smart systems.

Based on the model-based method for representing a scenario, a scenario-integrated approach for functional design is proposed here, which mainly involves four stages, that is, scenario analysis, functional synthesis, solution analysis, and logic development, as shown in Figure 3. This section will illustrate each stage in details as below.

In the scenario analysis stage, customer needs are refined with scenarios, so that the interactions of a smart system with human users and the surrounding environment can be clarified. At the functional synthesis stage, the general functions will be identified and be decomposed into subfunctions or even sub-subfunctions, followed by the solution generation process for the identified functions. After completing the functional synthesis process, designers are required to analyze various enabling conditions, performance, and side effects of the solution concepts, which are necessary for evaluating the feasibility of the proposed solution concepts in the solution analysis stage. The final stage, that is, the logic development stage, is responsible for logic development, which illustrates how functional logic can be employed to integrate various subsystems into a whole smart system for achieving desired functionalities.

Scenario analysis

Generally, functional design activities start from customer needs. However, since customer needs are usually verbally described and are often subjective and ambiguous, it is then necessary to transform such customer needs into technical functions, which often involve explicit inputs and outputs (Chen *et al.*, 2015). For example, the customer need, *a car can achieve intelligent navigation*, is very ambiguous and subjective, since the description not only fails

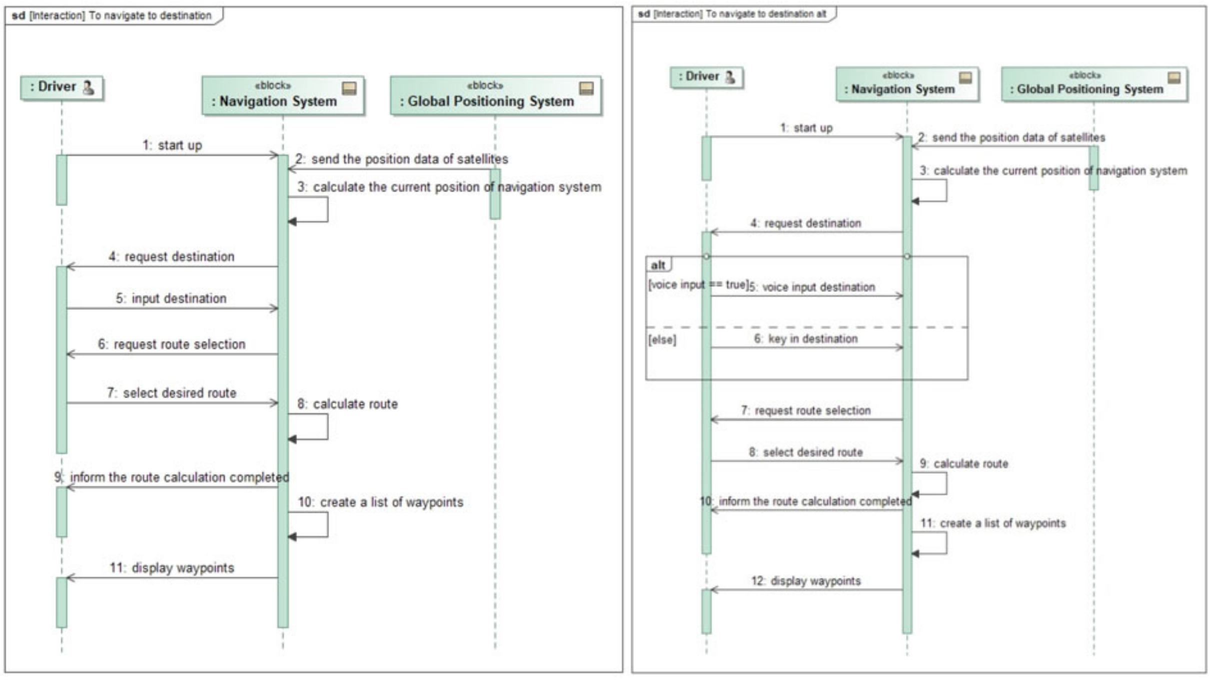


Scenario Name: To navigate a car to destination

Contextual Elements:

{ (*Actors*: [driver, global positioning system (GPS), ⋯]),
(Goal: "guide the driver to drive his/her car to destination")
(*Time*: "N/A"),
(*Location*: "normal motorway"),
(*Preconditions*: "the destination is clear"),
(*Postconditions*: "the driver arrives at the destination after navigating")
}

Fig. 1. The contextual elements of the destination-navigating scenario.


sd [Interaction] To navigate to destination
: Driver
«block» : Navigation System
«block» : Global Positioning System
1: start up
2: send the position data of satellites
3: calculate the current position of navigation system
4: request destination
5: input destination
6: request route selection
7: select desired route
8: calculate route
9: inform the route calculation completed
10: create a list of waypoints
11: display waypoints
sd [Interaction] To navigate to destination alt
: Driver
«block» : Navigation System
«block» : Global Positioning System
1: start up
2: send the position data of satellites
3: calculate the current position of navigation system
4: request destination
alt
[voice input == true]
5: voice input destination
[else]
6: key in destination
7: request route selection
8: select desired route
9: calculate route
10: inform the route calculation completed
11: create a list of waypoints
12: display waypoints

(a) The Sequence Diagram model of the main scenario *to navigate a car to destination*

(b) The Sequence Diagram model of an alternative scenario *to navigate a car to destination*

Fig. 2. The Sequence diagram models of the scenario *to navigate a car to destination*.

to include all actors, but also leaves a wide space for designers to explain what "intelligent navigation" means. Therefore, such subjective and ambiguous descriptions of customer needs should be represented in an explicit and unambiguous manner, so that it can be used for functional design. As mentioned before, a model-based representation of scenarios, that is, a sequence diagram model in SysML, can be employed to further clarify and express customer needs. Therefore, this research will develop a sequence diagram-based approach for scenario analysis to refine customer needs.

As mentioned before, a scenario can be represented with the sequence diagram model. As a result, ambiguous customer needs could then be translated into a series of interaction activities, which are easier for system designers to understand what the customer needs are. The general process of scenario analysis is elaborated as below.

First, designers should extract the contextual elements of a scenario, including *actors* (which involves both human users and external systems), *goals*, *time*, *location*, *preconditions*, and *postconditions*. Every contextual element should be specified according to

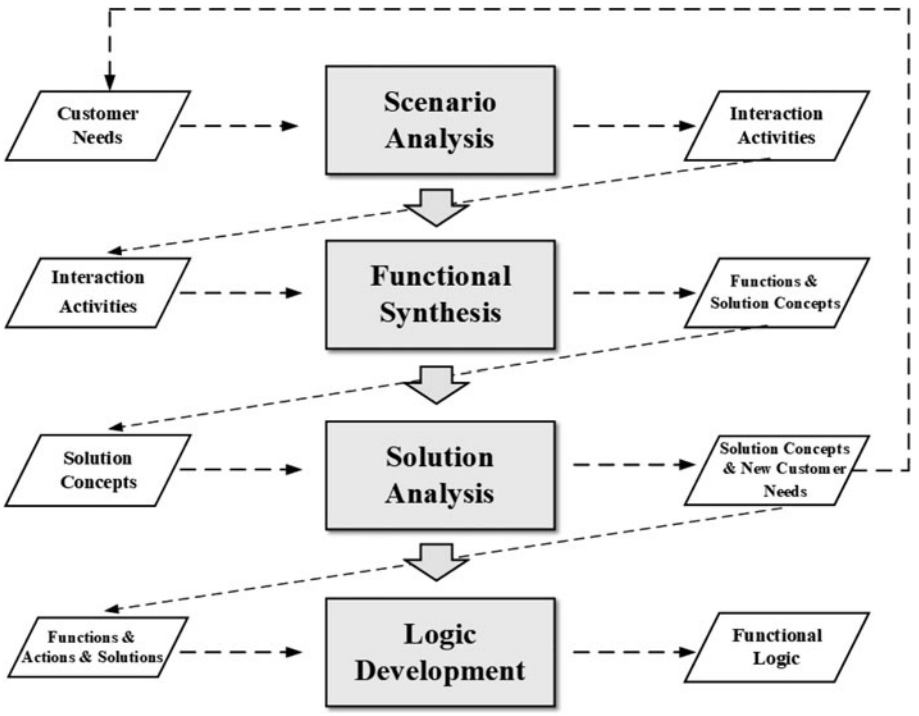


Fig. 3. The framework of the scenario-integrated approach for functional design.

the method that has been stated previously. For example, in the scenario, *to navigate a car intelligently*, which corresponds to the aforementioned customer need, *a car can achieve intelligent navigation*, the element *actors* that can be identified includes a driver, a real-time traffic data system, a GPS, and so forth. Therefore, the *actors* can be defined in a list of objects, that is, *Actors* = {driver, real-time traffic data system, GPS, …}. The *goal* is that a car can navigate intelligently. The complete set of the contextual elements of this scenario is not given here, because it is similar to that of the scenario in Figure 1.

Second, based on the actors that have been identified in the first step, designers are required to specify the activities of each actor in the execution of the scenario, including interaction activities between actors and the system of interest, as well as those between different actors. For example, in the above scenario, *to navigate a car intelligently*, some of the driver's activities can be identified as follows, for example, starting up the navigation system, inputting destination, and selecting a desirable route. In addition, activities of the navigation system include requesting the destination, requesting remote traffic data, calculating the driving route, displaying the map of driving route, warning traffic jams, and so on. Note that some complex activities could be properly decomposed and extended. For instance, the above activity *navigation system displays the map of driving route* could be decomposed into several subactivities, such as *navigation system identifies related section of the destination*, *navigation system calculates the waypoints of the route*, and *navigation system displays the related area of the waypoints*.

Finally, as the interaction activities of the system under development and those of the actors being specified, each interaction step can be expressed with the element *message* in sequence diagram. For brevity, only the interaction sequence in the main scenario, *to navigate a car intelligently*, is shown in Figure 4. The related alternative scenarios and exception scenarios are not modeled here.

The above example illustrates how a subjective and abstract customer need can be transformed into a detail interaction sequence through a scenario analysis process to represent the corresponding scenario. The interaction activities specified in this stage are basic inputs of the subsequent functional design stages.

Functional synthesis

After a sequence of explicit interaction activities has been generated in the previous stage, it is then necessary to carry out the functional synthesis process for functional design. During the functional synthesis process, the general functions of a system will be identified from the interaction activities and then be decomposed into subfunctions or even sub-subfunctions, followed by a synthesis process of searching for suitable solution principles for the functions generated.

A general function can be identified from an interaction activity through extracting the basic inputs and outputs of the function. Hereby, since the subject of an interaction activity can be the system under development or an actor (which can be a human user or an external system), the process of identifying general functions can be classified into two categories accordingly. In the first category, where the subject of an interaction activity is the system under development, the inputs and outputs can be extracted to capture the corresponding function. For example, in Figure 4, for the interaction activity, *the navigation system requests remote traffic data*, the input is an information flow, that is, *a message of requesting remote traffic data*; the output is an information flow, that is, *a request for remote traffic data*; therefore, the general function can be described as: *the navigation system shall request remote traffic data*. In the second category, the subject of an activity is a human user or an external system in the context, which means that the activity is implemented by an actor, rather than the system under development. In this situation, in order to interact with the external executor, the system under

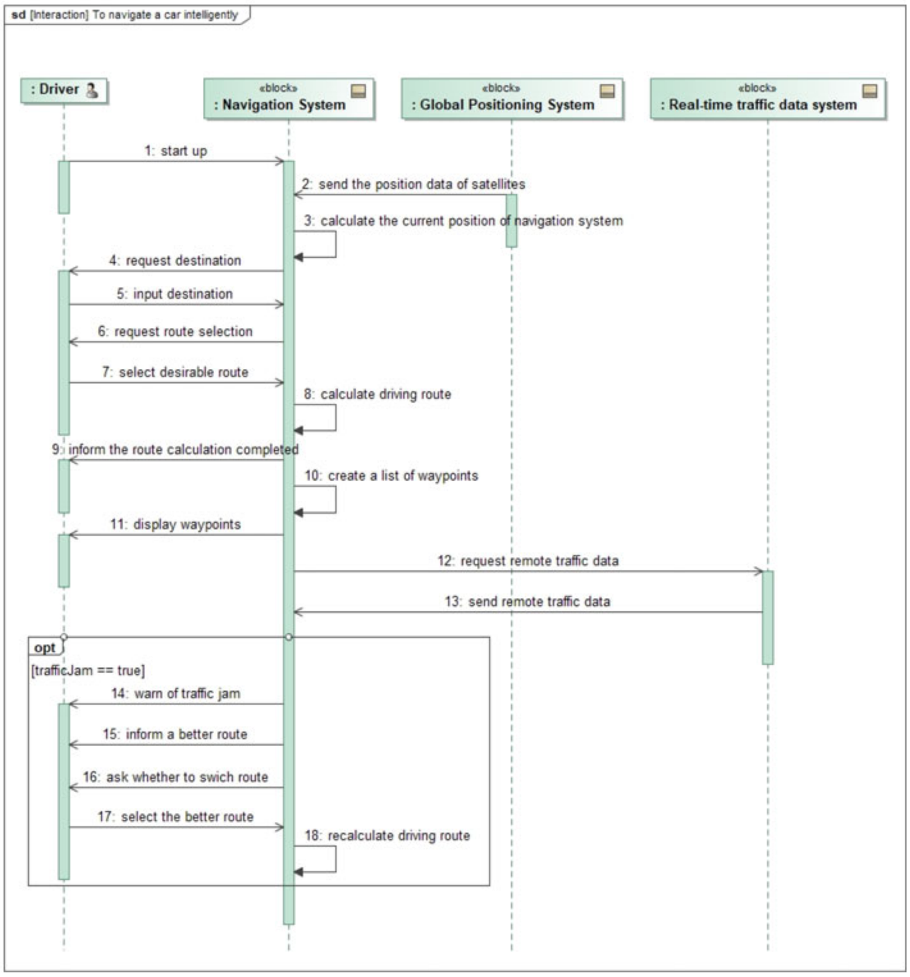

sd [Interaction] To navigate a car intelligently
: Driver
«block» : Navigation System
«block» : Global Positioning System
«block» : Real-time traffic data system
1: start up
2: send the position data of satellites
3: calculate the current position of navigation system
4: request destination
5: input destination
6: request route selection
7: select desirable route
8: calculate driving route
9: inform the route calculation completed
10: create a list of waypoints
11: display waypoints
12: request remote traffic data
13: send remote traffic data
opt
[trafficJam == true]
14: warn of traffic jam
15: inform a better route
16: ask whether to swich route
17: select the better route
18: recalculate driving route

Fig. 4. The Sequence Diagram model of the main scenario *to navigate a car intelligently*.

development should provide a related interface function, to achieve the general function for receiving the external inputs. Taking the activity, *the driver selects the better route*, as an example (shown in Figure 4), the subject of the activity is the driver. In order for the driver to fulfill the selecting operation, the navigation system should provide an interface function for the system to receive the input from the driver. Here, the input for the above general function of the navigation system can be extracted as an information flow, that is, *a message of route selection*, and the output is *a message about the result of route selection*. Therefore, the general function of the navigation system can be derived as: *the navigation system shall receive the route selection result from the driver*.

After a general function has been identified, a designer then usually needs to decompose the function into subfunctions or even sub-subfunctions of lower complexity for facilitating the subsequent solution generation process. Hereby, the flow-based functional decomposition method proposed by Pahl and Beitz (2007) is employed to implement the decomposition task. In addition, this research imports a graphical representation method called activity diagram from SysML to represent the functional decomposition process which specifies how the input flows of a general function can be transformed into output flows. Specifically, each subfunction or sub-subfunction can be represented with an *action* node in an activity diagram and should be described in an active mode. In addition, the elements *activity parameter* and *pin* can be employed to represent the input and output flows of each function. With the above function, *the navigation system shall request remote traffic data*, as an example, the functional decomposition process can be specified with an activity diagram model that established using a commercial functional modeling software, as shown in Figure 5. As a result, the above general function is decomposed into several subfunctions, including *the navigation system shall generate a request for accessing the communication network*, *the navigation system shall access the communication system*, *the navigation system shall send request for real-time traffic data*, and so on. It can be found that the activity diagram model can explicitly show the functional decomposition process with subfunctions, and the related inputs and outputs are clearly defined. Note that the decomposition of a complex function should be ended when the generated subfunctions can be achieved by related components.

The final task of the functional synthesis stage is to search for alternative solution principles for the generated subfunctions. Hereby, the mapping relationship between functions and solutions in the axiomatic design approach proposed by Suh (2001) and the systematic approach proposed by Pahl and Beitz (2007) can be imported for the solution generation process. Specifically, a designer can compare the input and output flows and the intended actions of a desired function with those of the solution principles in his/her memory or in a functional design catalog, and then choose some solution alternatives based on

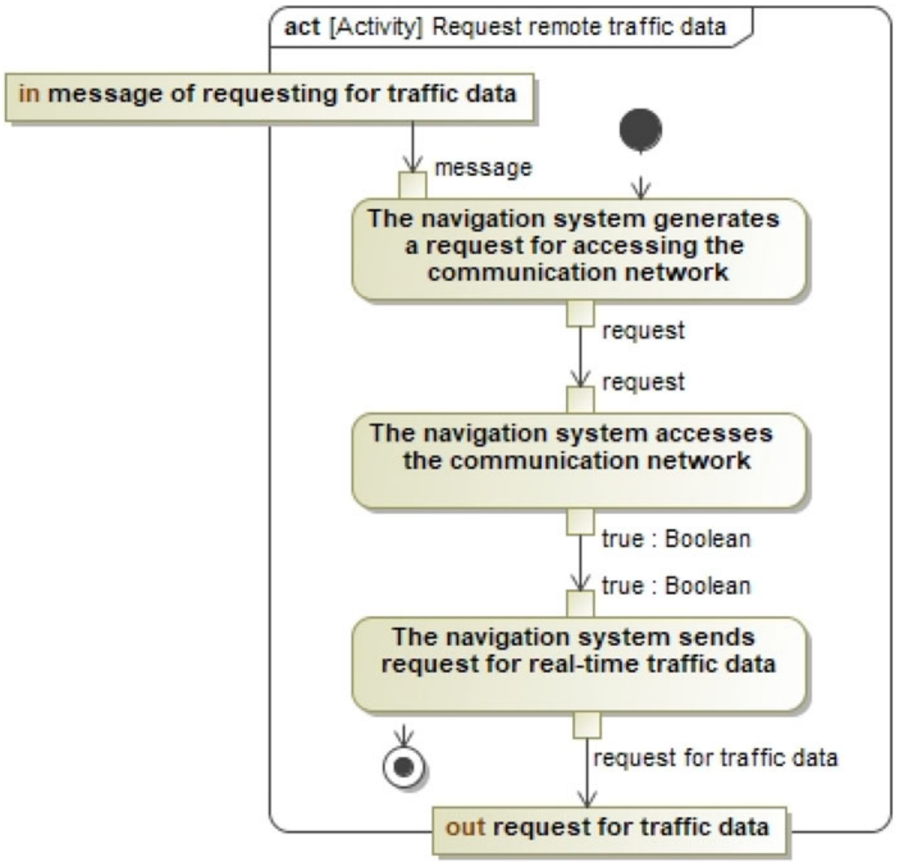

act [Activity] Request remote traffic data
in message of requesting for traffic data
message
The navigation system generates a request for accessing the communication network
request
request
The navigation system accesses the communication network
true : Boolean
true : Boolean
The navigation system sends request for real-time traffic data
request for traffic data
out request for traffic data

Fig. 5. The Activity Diagram model of the remote traffic data-requesting function.

the similarity of the inputs and outputs. For example, for the function, *the navigation system shall access the communication network*, the input and the output flows are both the information flow, *a message of accessing network*, after searching in a functional design catalog, some solution principles with similar inputs and outputs can be found, such as *cellular mobile network*, *wireless network*, and *satellite communication connection*. In addition, a knowledge-based approach proposed in our previous research (Chen *et al*., 2012) can also be adopted to assist designers in retrieving possible solution principles from a solution base, and then integrating them into a whole solution for achieving a desired function.

As seen above, through extracting the inputs and outputs from interaction activities, the functions of a system under development can be identified and then decomposed into subfunctions, and some alternative solution principles can be generated based on the mapping relationship between functions and solutions in the functional synthesis stage. In addition, functions of an external system can also be defined when they provide support for the smart system under development to complete a task. Meanwhile, the smart system also should provide corresponding interface functions to receive the inputs. Therefore, our approach can not only generate functions of the system of interest, but also can capture the functions the external systems in the context should provide.

Solution analysis

After the solution concepts have been generated for a function, it then comes into the solution analysis stage, where designers are required to further analyze these solutions to determine whether they can meet the functions and customer needs in all aspects, such as enabling conditions, performance of the inputs and outputs, and side effects (Roozenburg and Eekels, 1995; Pahl and Beitz, 2007).

If an enabling condition is necessary for a solution to work, a new customer need can then be derived. For example, for the solution concept of *cellular mobile network*, two enabling conditions can be derived, that is, the enabling condition that the direct current is prerequisite to power on the system, and that the system should be able to receive cellular network signal. Therefore, according to these two enabling conditions, two new customer needs can be generated, that is, *the navigation system needs direct current to power on*, and *the system can receive cellular network signal*.

As to the side-effects analysis, if an unintended action is generated from a solution concept, designers should decide whether the action can be acceptable or not. The acceptability of a side effect depends on the results from the action on the system of interest. More detailed analysis can be found in our previous research (Chen *et al*., 2015). If a side effect is unacceptable, the designer should either abandon the corresponding solution concept or propose a new need to solve it. In addition, the performance of a solution concept should also be analyzed, and a new customer need can be derived to improve the performance when it is unsatisfying.

Once a new customer need is derived, a new cycle of functional design should be triggered. The iterative analysis process should be continued until all new needs are satisfied.

Note that a state-based solution analysis process is clarified here. According to our recent research, a system, especially a smart system, may possess multiple different states in different situations, which means that the current state of a system can switch to another one, depending on the changing environment. Generally, a system can achieve different functions in different states, and it usually should be in a certain state if it is required to achieve a specific function. Otherwise, a new customer need should be derived, that is, to transform the system into a required working state. A typical example of a state transition function is that the state of landing gear system of an aircraft should change from the gear-up state into the gear-down state when landing.

Logic development

After a complete function architecture and the corresponding solution concepts have been generated in the above stages, it then comes to the final stage, that is, logic development stage, which is responsible for designing suitable function logic to integrate various subsystems into a whole smart system for fulfilling desired functionalities.

During the functional design process, functional logic primarily involves temporal logic and decision logic. Hereby, the temporal logic can be further classified as two types, that is, sequential logic and parallel logic. Sequential logic means that multiple actions or functions should be carried out in an explicitly sequential order. For example, the functional logic between the action, *a driver starts up the navigation system*, and the action, *the navigation system displays a map of current location*, should be sequential logic, because they have to be executed one by one. In contrast, parallel logic means that multiple actions do not interfere with each other when performed simultaneously. For instance, the action, *the navigation system displays the current driving route*, and the action, *the navigation system broadcasts the current speed*, can take place at the same time. Therefore, the parallel logic relation should be set between them.

Decision logic will appear in the situation where a decision should be made. Different execution sequences can be triggered under different decisions. For example, as for the action, *the navigation system asks the driver whether to switch route*, two different subsequent action sequences can be activated, that is, the sequence of driving on the original route and that of a new chosen route, according to the decision of the driver.

These two categories of functional logic allow designers to combine all probable actions or functions in different situations, to generate various functional architecture and solution concepts, and then to integrate corresponding subsystems into a whole smart system.

In this paper, the swim lane model (also known as activity partition) in SysML is employed to illustrate the functional logic. The swim lane model at least has two major advantages. On one hand, such model can specify the necessary functional logic, which is the primary concern in this stage. On the other hand, the model can display the relationships between actions (or functions) and actors or the system under development that will implement the actions (or functions), which is convenient for allocating functions to different actors or the system under development. Specifically, in swim lane, *control flows* and *object flows* can be used to represent the sequential logic between different actions; *fork nodes* and *join nodes* are employed to specify the parallel logic; *decision nodes* can be adopted to define the decision logic.

Case study

In this section, we will employ the functional design of a smart vehicle to illustrate the proposed functional design approach. The primary customer need here is related to a self-service function that allows a renter to unlock the doors of a vehicle with a smartphone by himself or herself. Therefore, the primary function here is related to an automated doors-unlocking system for a smart vehicle. Based on the above customer need, the design process of the automated doors-unlocking function is introduced as follows.

Refining customer needs

According to the scenario-integrated approach proposed above, the first stage is scenario analysis, where a designer is required to refine a customer need into a scenario and transform it into a series of explicit interaction activities in the scenario.

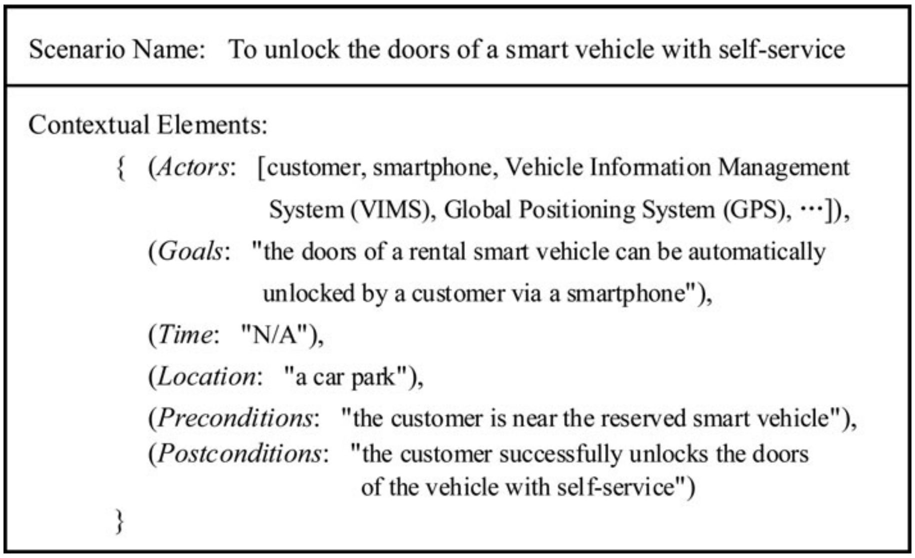
As stated above, the customer need in this case is that *a customer can unlock the doors of a smart vehicle for renting by himself or herself*. Here, the scenario, *to unlock the doors of a smart vehicle with self-service*, is used to refine the above need. According to our approach, first, we should extract the contextual elements of the scenario. To achieve the complex function of automated doors-unlocking, a smart vehicle alone is not sufficient, it is obvious that some external systems in the context are necessary to provide support, such as a smartphone, a remote vehicle information management system (which will be abbreviated as VIMS), a global positioning system (abbreviated as GPS). Therefore, the *actor* elements of the scenario can be specified as an object set, that is, *Actors* = {customer, smartphone, VIMS, GPS, …}. The element *goal* can be described as: *the doors of a rental smart vehicle can be automatically unlocked by a customer via a smartphone*. After all elements have been extracted, the complete set of contextual elements of the above scenario can be depicted in Figure 6.

Thereafter, the possible activities of each actor and the system under development should be clarified according to the implementation of the scenario, and the interactions between different actors should also be involved. For example, an activity of the customer can be described as: *the customer creates a request message of doors-unlocking for a smart vehicle with a smartphone*, and an activity of the smart vehicle can be clarified as: *the smart vehicle unlocks the electronic locks*, and so forth. Then, the interaction activities identified before should be represented in a sequence diagram model. The interaction sequences of the aforementioned scenario, *to unlock the doors of a smart vehicle with self-service*, are illustrated in Figure 7. Note that only the main successful scenario, that is, *the doors of a smart vehicle can be unlocked successfully*, is taken into consideration in this case for brevity. If some related alternative scenarios and exception scenarios are created, more interaction sequences will be generated.

Synthesizing system functions

In the second stage, that is, the functional synthesis stage, designers should extract the inputs and outputs from the interaction activities specified before to identify general functions, decompose the general functions into subfunctions, and search for alternative solution concepts for the subfunctions.

Since there are two categories of functional identifying process, two examples will be given here to illustrate the functional synthesis. One example is the interaction activity, *the VIMS sends a*



Scenario Name: To unlock the doors of a smart vehicle with self-service

Contextual Elements:

{ (*Actors*: [customer, smartphone, Vehicle Information Management System (VIMS), Global Positioning System (GPS), ⋯]),
(*Goals*: "the doors of a rental smart vehicle can be automatically unlocked by a customer via a smartphone"),
(*Time*: "N/A"),
(*Location*: "a car park"),
(*Preconditions*: "the customer is near the reserved smart vehicle"),
(*Postconditions*: "the customer successfully unlocks the doors of the vehicle with self-service")
}

Fig. 6. The contextual elements of the doors-unlocking scenario.

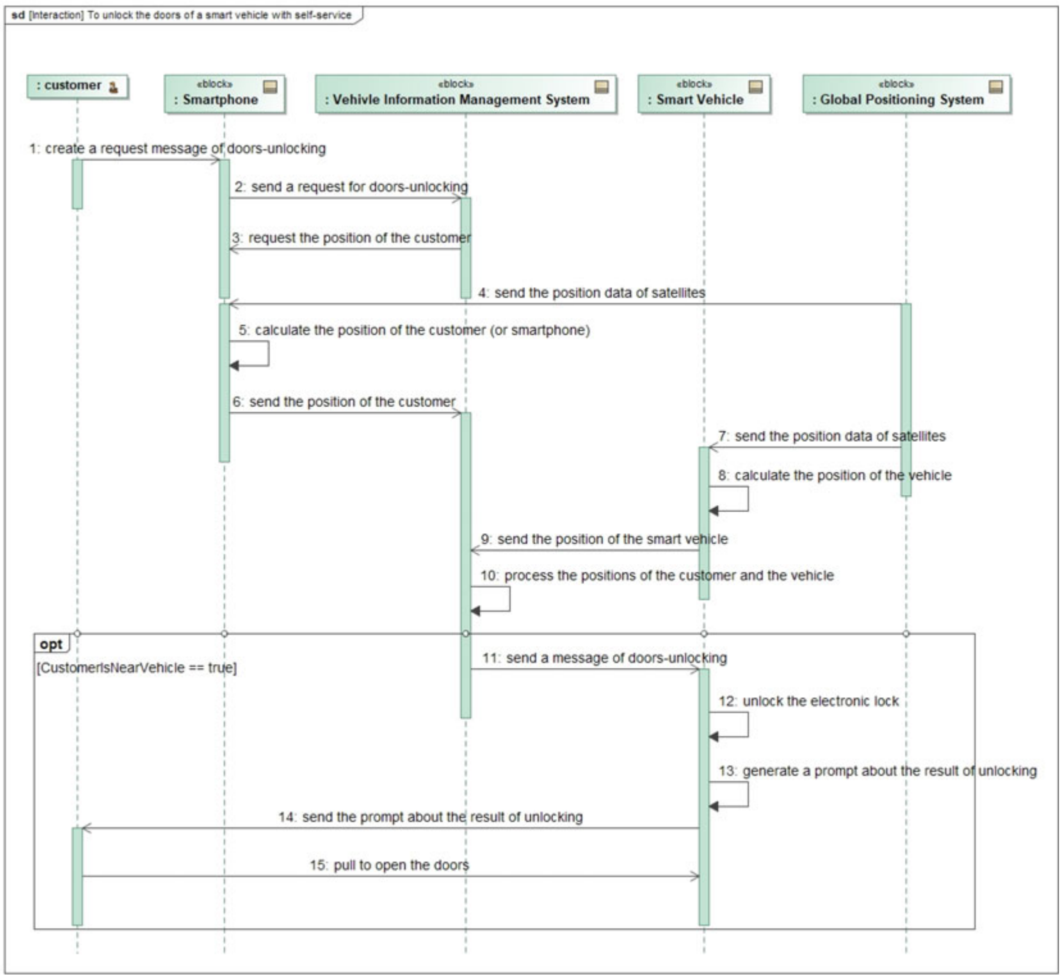

sd [Interaction] To unlock the doors of a smart vehicle with self-service
: customer
«block» : Smartphone
«block» : Vehivle Information Management System
«block» : Smart Vehicle
«block» : Global Positioning System
1: create a request message of doors-unlocking
2: send a request for doors-unlocking
3: request the position of the customer
4: send the position data of satellites
5: calculate the position of the customer (or smartphone)
6: send the position of the customer
7: send the position data of satellites
8: calculate the position of the vehicle
9: send the position of the smart vehicle
10: process the positions of the customer and the vehicle
opt
[CustomerIsNearVehicle == true]
11: send a message of doors-unlocking
12: unlock the electronic lock
13: generate a prompt about the result of unlocking
14: send the prompt about the result of unlocking
15: pull to open the doors

Fig. 7. The Sequence Diagram model of the doors-unlocking scenario.

message of doors-unlocking to the smart vehicle, in which the subject of the activity is an external system in the context, rather than the smart vehicle. The inputs of the above activity are information flows, that is, *a position data of the customer and a position data of the smart vehicle*; the output is an information flow, that is, *a doors-unlocking message*, which will be an input of the next activity. Therefore, the general function of the VIMS can be identified as, *the VIMS shall send a message of doors-unlocking to the smart* *vehicle*. As stated in the section "functional synthesis," in order to interact with the VIMS, the smart vehicle should provide an interface function to receive the inputs from VIMS; therefore, the general function of smart vehicle corresponding to the above function can be identified as, *the smart vehicle shall receive the message of doors-unlocking from the VIMS*. The other example corresponds to the other category of function identifying that the system under development is the subject of an interaction activity, that is, *the smart vehicle unlocks the electronic lock*. For this activity, the input can be extracted as an information flow, that is, *a message of doors-unlocking*; the output is also an information flow, that is, *a message about the result of the doors-unlocking action*. As a result, the general function can be identified as: *the smart vehicle shall unlock the electronic lock*.

After some general functions have been identified, designers should manage to decompose the complex general functions to subfunctions according to the flow-based functional decomposition approach, and represent the decomposition results with activity diagrams. For example, for the function, *the VIMS shall send a message of doors-unlocking to the smart vehicle*, the functional decomposition process can be represented with the activity diagram in Figure 8a based on the transformation of the input flow, that is, *a position data of the customer and a position data of the smart vehicle*. Therefore, the above function can be decomposed into some subfunctions, including *the VIMS shall calculate the distance between the customer and the smart vehicle*, *the VIMS shall check the distance between the customer and the smart vehicle*, *the VIMS shall generate a message of doors-unlocking for the smart vehicle* (or *the VIMS shall reject the request for unlocking doors from the customer*), and *the VIMS shall send the message of doors-unlocking to the smart vehicle*. For another function, *the smart vehicle shall unlock the electronic lock*, which transforms the input flow, *a message of doors-unlocking*, to the output flow, *a message about the result of the doors-unlocking action*, can be decomposed into several subfunctions as shown in Figure 8b.

Due to the limited space, it is impossible to illustrate all activity diagram models for the functional decomposition process here. Figure 9 illustrates the relationship between some typical interaction activities and the subfunctions identified. Note that the general functions that can be identified from the interaction activities are not listed in the figure for brevity. It can be found that not only the functions of the smart system, but also the supporting functions of external systems can be identified in our approach.

After the complex general functions have been decomposed into subfunctions, designers should then search for some alternative solution concepts for each subfunction. As stated before, an approach similar to the combination of the axiomatic design approach by Suh (2001) and the systematic design approach by Pahl and Beitz (2007) is employed for generating solutions; therefore, several solution concepts can be found for a function through comparing the inputs and outputs of them. In this case, taking the function, *the smartphone shall receive a message for starting up the rental app*, as an example, the input flow of the desired function can be *a message for starting up the app*, and the output flow should be *a starting-up message*. The designer can search for solution concepts in his/her memory and compare the input and output flows of the function with those of the solutions, multiple solution alternatives with similar inputs and outputs can then be found, such as *pressure sensing technology of touch screen*, *speech recognition technology*, *cellphone keypads*, and *external device*. For another function, *the smartphone shall send the position data of the customer to the VIMS*, the input flow of this function can be *position data of the customer*, and the output flow should also be *the position data of the customer*. After comparing the input and output flows of the desired function with those of the technical solutions in a designer's memory or in a functional design catalog, some alternative solutions can be generated, such as *GPS module* and *Bluetooth*.

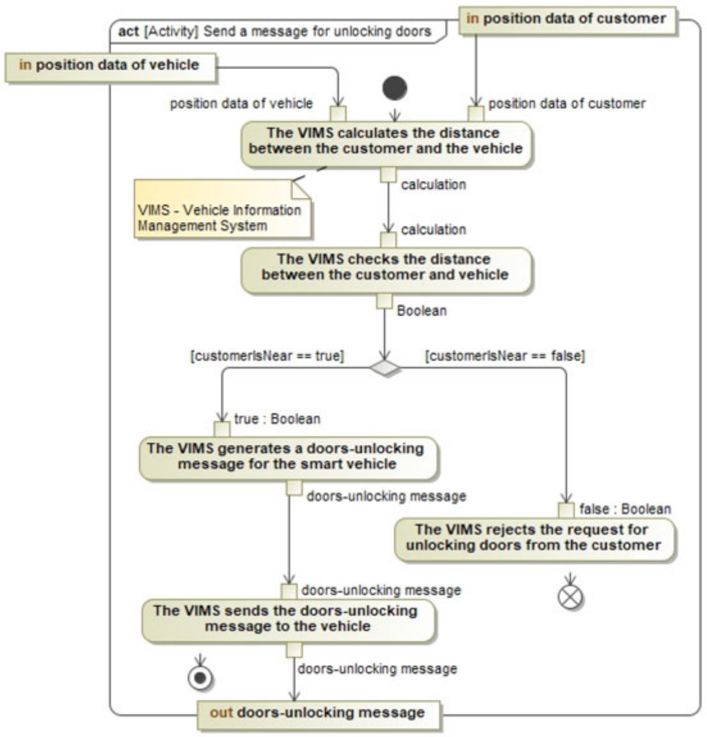
Analyzing solution concepts

The third stage is solution analysis, which is responsible for further analyzing the solution concepts generated in the previous stage for choosing suitable ones. Specifically, the enabling conditions, performance of the inputs and outputs, and side effects of the solutions should be evaluated to determine the feasibility of them.

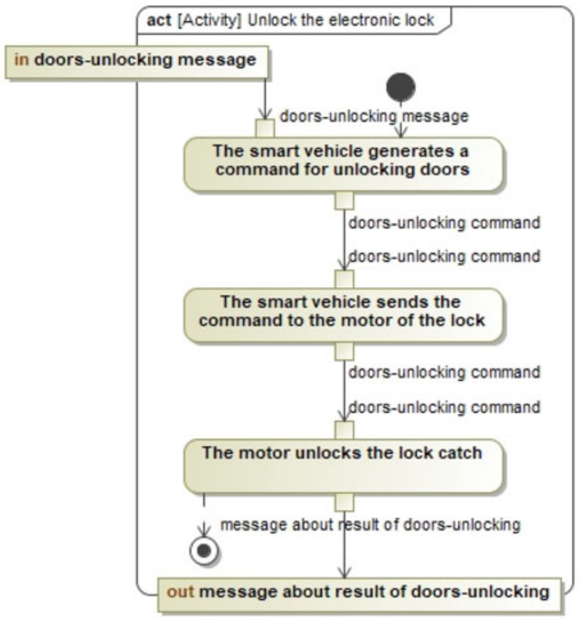
Assume that the previous functional synthesis process has generated the following solution concepts: *pressure sensing technology of touch screen*, *speech recognition technology*, *cellphone keypads*, and *external device*. The solution analysis process can be carried out as follows. In this case, due to the limited space of a *cellphone*, it is almost impossible for a cellphone to employ a keypad to receive inputs, resulting in that the keypad solution has to be abandoned. It is also impossible to employ an *external device* for receiving information inputs, since such a device is often inconvenient for users to carry. For the solution, *speech recognition technology*, it can be employed to realize some simple functions, but cannot be used to implement multiple complex commands continually at present. For the solution, *pressure sensing technology of touch screen*, it now has been widely utilized by smartphones and can satisfy the demand for reacting to continuous operations. Therefore, *touch screen technology* and *speech recognition technology* can both be employed and integrated into a smartphone.

Another example for analyzing enabling conditions or performance of a solution concept is stated as below. In this case, the solutions, *GPS module* and *Bluetooth*, are employed to achieve the function, *the smartphone shall send the position data of the customer to the VIMS*. As for the solution concept *GPS module*, in order to achieve the *position data-sending* function, an enabling condition that the smartphone should access the communication network is needed. Therefore, a new customer need is derived here, that is, *the smartphone needs to access the network*. To satisfy this new need, a new cycle of functional design process should be started. Since this need is relatively simple, it is not necessary to establish a sequence diagram model and activity diagram models. Therefore, the alternative solutions can be directly generated, such as *cellular network technology* and *WiFi network technology*. Note that if the cellular network cannot be accessed at a location (e.g., an underground car park) for sending the position information to the VIMS, the *Bluetooth* solution can then be employed to fulfill the function of sending the position information. Therefore, these two solution concepts can be integrated into a smartphone to ensure that the function above can be achieved. It should be noted that if a new customer need derived is complex enough, a complete functional design process as above should be carried out to generate the technical solutions, including establishing sequence diagram models and activity diagram models.

The above design cycles should be iteratively implemented until all customer needs have been addressed. Due to the limited space, it is impossible to analyze all solutions here. In a similar way, solution concepts corresponding to each function can be obtained. Some solution analysis results are illustrated in Figure 10. It should be noted that several different subfunctions or sub-subfunctions are likely to be realized by the same technical


act [Activity] Send a message for unlocking doors
in position data of customer
in position data of vehicle
position data of vehicle
position data of customer
The VIMS calculates the distance between the customer and the vehicle
calculation
VIMS - Vehicle Information Management System
calculation
The VIMS checks the distance between the customer and vehicle
Boolean
[customerIsNear == true]
[customerIsNear == false]
true : Boolean
The VIMS generates a doors-unlocking message for the smart vehicle
doors-unlocking message
false : Boolean
The VIMS rejects the request for unlocking doors from the customer
doors-unlocking message
The VIMS sends the doors-unlocking message to the vehicle
doors-unlocking message
out doors-unlocking message

(a) The activity diagram model ofthe function for sending doors-unlocking message


act [Activity] Unlock the electronic lock
in doors-unlocking message
doors-unlocking message
The smart vehicle generates a command for unlocking doors
doors-unlocking command
doors-unlocking command
The smart vehicle sends the command to the motor of the lock
doors-unlocking command
doors-unlocking command
The motor unlocks the lock catch
message about result of doors-unlocking
out message about result of doors-unlocking

(b) The activity diagram model of the electronic lock unlocking function

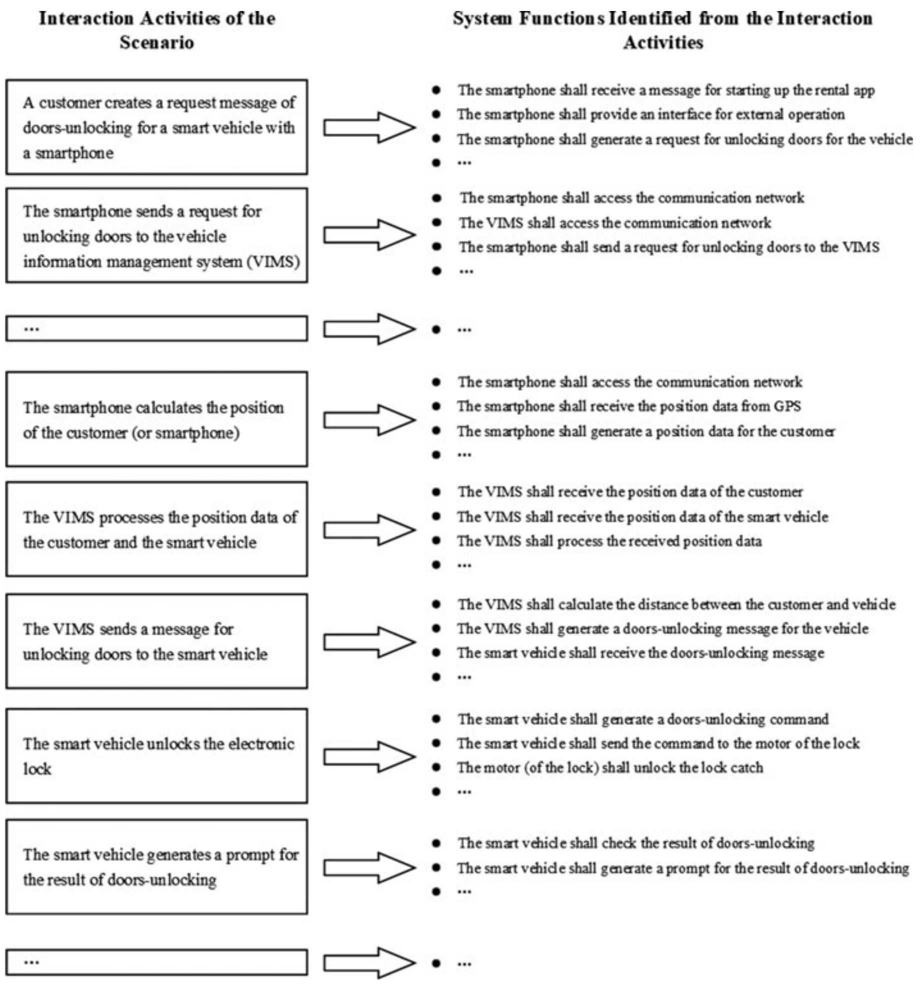
Fig. 8. The Activity Diagram models of two general functions identified from activities.

solution, which means that there can be a one-to-many mapping relationship between solution concepts and functions, as shown in Figure 10.

Developing functional logic

The last stage is functional logic development, where a system engineer is required to figure out the functional logic among all the functions or actions. Specifically, the temporal logic is employed when the execution sequence in an activity diagram should be determined between different actions. In contrast, the decision logic should be defined if a decision is necessary to be made, and different results will be generated according to different decisions.

Taking the interaction activity, *the VIMS sends a doors-unlocking message to the smart vehicle*, as an example, the



Interaction Activities of the Scenario

System Functions Identified from the Interaction Activities

A customer creates a request message of doors-unlocking for a smart vehicle with a smartphone

- The smartphone shall receive a message for starting up the rental app
- The smartphone shall provide an interface for external operation
- The smartphone shall generate a request for unlocking doors for the vehicle
- …

The smartphone sends a request for unlocking doors to the vehicle information management system (VIMS)

- The smartphone shall access the communication network
- The VIMS shall access the communication network
- The smartphone shall send a request for unlocking doors to the VIMS
- …

…

- …

The smartphone calculates the position of the customer (or smartphone)

- The smartphone shall access the communication network
- The smartphone shall receive the position data from GPS
- The smartphone shall generate a position data for the customer
- …

The VIMS processes the position data of the customer and the smart vehicle

- The VIMS shall receive the position data of the customer
- The VIMS shall receive the position data of the smart vehicle
- The VIMS shall process the received position data
- …

The VIMS sends a message for unlocking doors to the smart vehicle

- The VIMS shall calculate the distance between the customer and vehicle
- The VIMS shall generate a doors-unlocking message for the vehicle
- The smart vehicle shall receive the doors-unlocking message
- …

The smart vehicle unlocks the electronic lock

- The smart vehicle shall generate a doors-unlocking command
- The smart vehicle shall send the command to the motor of the lock
- The motor (of the lock) shall unlock the lock catch
- …

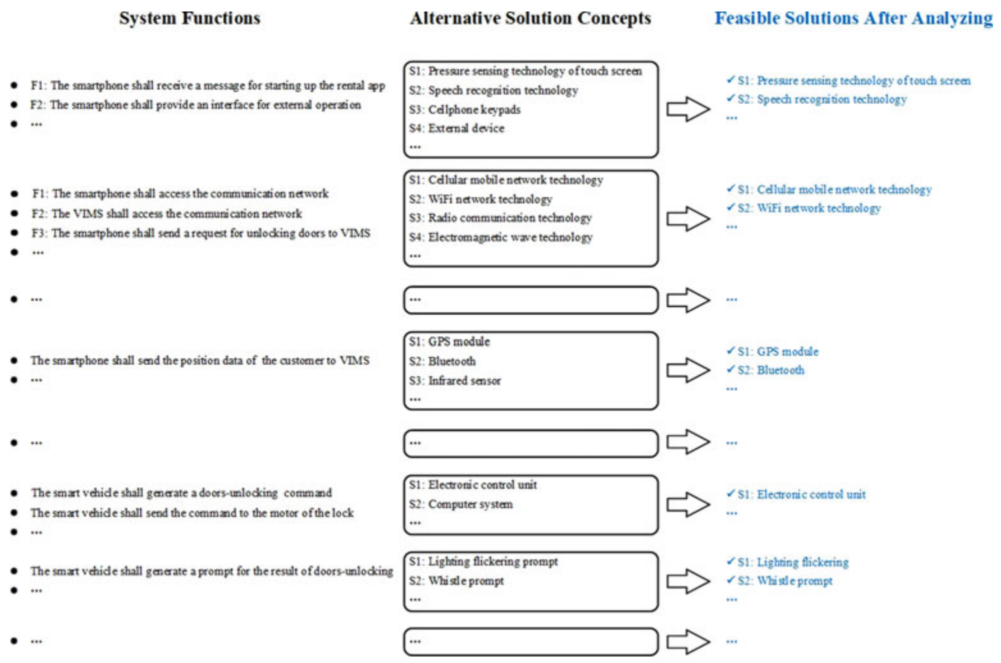
The smart vehicle generates a prompt for the result of doors-unlocking

- The smart vehicle shall check the result of doors-unlocking
- The smart vehicle shall generate a prompt for the result of doors-unlocking
- …

…

- …

Fig. 9. The system functions identified from the interaction activities in the scenario *to unlock the doors of a smart vehicle with self-service*.



System Functions | Alternative Solution Concepts | Feasible Solutions After Analyzing

- F1: The smartphone shall receive a message for starting up the rental app
- F2: The smartphone shall provide an interface for external operation
- …

S1: Pressure sensing technology of touch screen
S2: Speech recognition technology
S3: Cellphone keypads
S4: External device
…

✓ S1: Pressure sensing technology of touch screen
✓ S2: Speech recognition technology
…

- F1: The smartphone shall access the communication network
- F2: The VIMS shall access the communication network
- F3: The smartphone shall send a request for unlocking doors to VIMS
- …

S1: Cellular mobile network technology
S2: WiFi network technology
S3: Radio communication technology
S4: Electromagnetic wave technology
…

✓ S1: Cellular mobile network technology
✓ S2: WiFi network technology
…

- …

…

…

- The smartphone shall send the position data of the customer to VIMS
- …

S1: GPS module
S2: Bluetooth
S3: Infrared sensor
…

✓ S1: GPS module
✓ S2: Bluetooth
…

- …

…

…

- The smart vehicle shall generate a doors-unlocking command
- The smart vehicle shall send the command to the motor of the lock
- …

S1: Electronic control unit
S2: Computer system
…

✓ S1: Electronic control unit
…

- The smart vehicle shall generate a prompt for the result of doors-unlocking
- …

S1: Lighting flickering prompt
S2: Whistle prompt
…

✓ S1: Lighting flickering
✓ S2: Whistle prompt
…

- …

…

…

Fig. 10. The mapping relationship between systemfunctions and solution concepts.

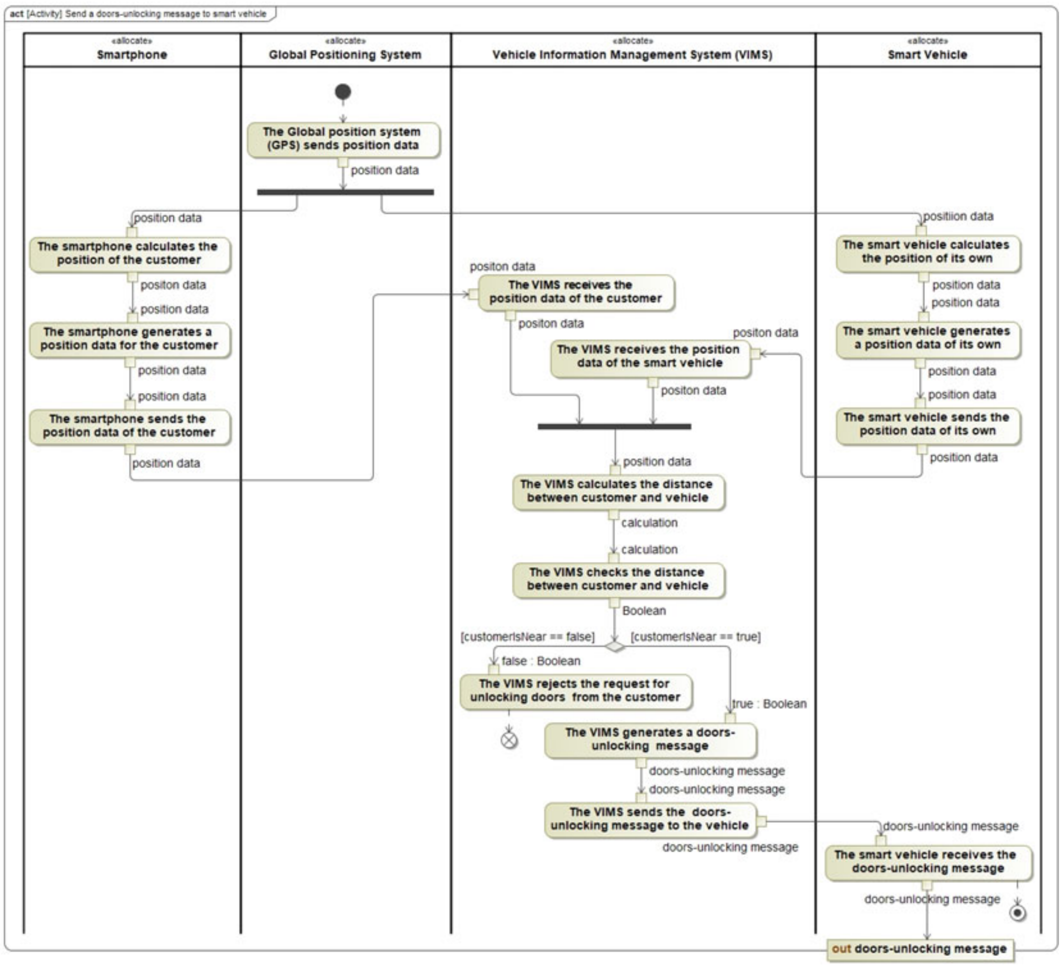

act [Activity] Send a doors-unlocking message to smart vehicle
«allocate» Smartphone
«allocate» Global Positioning System
«allocate» Vehicle Information Management System (VIMS)
«allocate» Smart Vehicle
The Global position system (GPS) sends position data
The smartphone calculates the position of the customer
The smartphone generates a position data for the customer
The smartphone sends the position data of the customer
The VIMS receives the position data of the customer
The VIMS receives the position data of the smart vehicle
The VIMS calculates the distance between customer and vehicle
The VIMS checks the distance between customer and vehicle
[customerIsNear == false]
[customerIsNear == true]
The VIMS rejects the request for unlocking doors from the customer
The VIMS generates a doors-unlocking message
The VIMS sends the doors-unlocking message to the vehicle
The smart vehicle calculates the position of its own
The smart vehicle generates a position data of its own
The smart vehicle sends the position data of its own
The smart vehicle receives the doors-unlocking message
out doors-unlocking message

Fig. 11. The swim lane model of *sending a doors-unlocking message to smart vehicle*.

relevant execution sequence and functional logic are illustrated in the swim lane model in Figure 11. To create a relatively complete functional logic, an exceptional execution sequence is also considered in the model.

As seen in Figure 11, a *fork node* is created before the action, *the smartphone calculates the position of the customer* (*or smartphone*), and the action, *the smart vehicle calculates the position of its own*, which means that the two actions can take place at the same time and the order of completion of them is nondeterministic. The *object flow* between the action, *the smartphone sends the position data of the customer*, and the action, *the VIMS receives the position data of the customer*, represents that these two actions should be executed in a sequential order. The output flow of the former action will be transferred to the latter one as the input flow. In addition, a *decision node* has been defined after the action, *the VIMS checks the distance between the customer and the smart vehicle*, to specify a decision logic. Specifically, if the output of the above action is *false*, which means the VIMS judges that the customer is not within an effective distance from the smart vehicle, then the request for unlocking doors will be rejected. Thereafter, a corresponding prompt will be sent to the customer, which is not defined in the model for brevity. If the result *true* is generated from the action, which means the customer is standing near the smart vehicle, then the other action sequence will be triggered, that is, *a doors-unlocking message* will be generated and sent to the smart vehicle. It can be found that the result of functional allocation is also explicitly specified in the swim lane model, since each action is clearly appointed to its executor in a single swim lane. For example, the action, *the smart vehicle receives the doors-unlocking message*, is assigned to the swim lane *smart vehicle*.

In a similar way, the complete logic development process can be accomplished. The functional logic generated here can then be used to integrate various actions or functions together. As a result, based on the functional allocation, the system under development and some external systems can be integrated into an entire smart system to fulfill desired functions.

Discussion

Based on the above case study, it can be found that the proposed scenario-integrated approach for functional design can assist system engineers to generate the functional architecture, solution concepts, and complex functional logic of a smart system. The approach can also be supported with various diagram models of SysML, which is a standard methodology for Model-Based Systems Engineering. To better explain our scenario-integrated approach for functional design, several influential functional design approaches are analyzed here for comparison.

The first one is the systematic design approach proposed by Pahl and Beitz (2007), where the functional design is simplified as a functional decomposition based on input and output flows. However, the decomposition approach is only suitable for the traditional mechanical systems, especially on the device-level products, where only the internal functions of a product are analyzed. When used for the functional design of smart systems, the functional decomposition approach would have two major drawbacks. On one hand, it does not fully address the interactions between the system under development and the external systems in the environment, which the functional design of a smart system must deal with. On the other hand, the functional decomposition approach does not support the modeling of complex functional logic (e.g., decision logic), which is very common in smart systems. In contrast, our scenario-integrated approach for functional design can not only deal with the interactions related to a smart system, but also the complex functional logic involved in a smart system.

The second one is axiomatic design approach proposed by Suh (2001), in which a functional design process is considered as a zigzag mapping process between the customer domain, the function domain, and the physical domain. Similar to the functional decomposition approach, the zigzag mapping model also does not take into consideration the interactions between systems under development, human users, and the environment, which is important for capturing the supporting functions of external systems and the corresponding functional interfaces of system under development. The zigzag mapping model also does not include functional logic development, which, therefore, is not suitable for the functional design of a smart system. As mentioned above, since scenario models and activity diagram models have been integrated in our approach, both the interactions and the functional logic of a smart system can be well addressed in our functional design approach.

The third one is Harmony for Systems Engineering approach (abbreviated as Harmony SE) (Hoffmann, 2011), which aims at developing a SysML-based design methodology for the development of cyber-physical systems. In Harmony SE, the concept of use case is used to describe a specific operational aspect of a system, which is similar to the concept of scenario in our approach. However, compared with our approach, Harmony SE does not explicitly elaborate how functions can be identified and decomposed, and does not take into consideration the supporting functions of external systems. Another limitation is that Harmony SE lacks a detailed understanding about the solution analysis process, where engineers are required to analyze the enabling conditions, performance, and side effects of a solution alternative, and to derive new customer needs if necessary.

Based on the above discussion, it can be demonstrated that our approach can support the functional design of smart systems in a more comprehensive way. It should be noted that with the advent of new generation information technologies (IT) in industry and product design, such as big data and cloud, Internet of Things, the manner of functional design can also be affected and enriched. By integrating such technologies into smart systems, systems can become more intelligent and can interact with users and its environment in a more effective manner.

Conclusions

As smart systems are becoming ubiquitous, it is increasingly significant to have a systematic approach for guiding designers to achieve the functional design of smart systems, which is vital for reducing the development cycle and enhancing the design quality. Since the traditional functional design approaches often do not deal with the interactions and functional logic, they are not suitable for guiding the functional design of smart systems. Therefore, this paper is devoted to the development of a scenario-integrated approach for functional design of smart systems.

Since scenarios are often used in software engineering to capture and document interaction sequences between human users and software, the concept of scenario is introduced in this research, to describe the complex interactions between a smart system and its actors, which include both human users and external systems. After elaborating the concept of scenario, a model-based method for representing the scenario is elaborated, which employs sequence diagram models to explicitly describe the interaction sequences for satisfying a specific goal. Thereafter, a scenario-integrated approach for functional design is proposed based on the model-based scenario representation, which can be regarded as a comprehensive methodology that combines the software engineering methodology with the product design methodology. A salient feature of the functional design approach is that it can effectively assist designers in transforming a subjective customer need into interaction sequences, then into system functions and ultimately into the solution concepts of a smart system. The functional design of the automated doors-unlocking system of a smart vehicle is employed as an example to demonstrate that the proposed approach is suitable for the functional design of smart systems. However, it should be acknowledged that our functional design approach is more applicable to functional integration design, where a solution concept is generated through the integration of various functional components, compared with an original or creative design task. For an original or creative design task, it is often difficult to employ our functional design approach, since the contextual information and the solution principles are often unknown to designers, which makes it impossible to carry out the scenario analysis work.

Based on our functional design research, an important future work is to develop a computer-based tool to support and manage the functional design process of smart systems in a more effective manner. It can be found that the scenario-integrated functional design approach proposed here primarily deals with the qualitative aspect of functional design, another future work that should be done is to develop a structured approach for dealing with the quantitative aspects of functional design. In addition, since the smart systems discussed in this paper are still not smart enough, for example, they do not possess some advanced artificial intelligence (AI) skills (e.g., machine learning), it would be necessary to develop a smarter design approach for the functional design of the smart systems that can incorporate such advanced AI abilities.

Acknowledgments. The authors are grateful to the anonymous reviewers for their constructive suggestions.

Funding. This research is supported by the Natural Science Foundation of China (Grant No. 51875346) and the Ministry of Industry and Information Technology of China.

References

Alexander IF and Stevens R (2002) *Writing Better Requirements*. London: Pearson Education.

Aurisicchio M, Bracewell R and Armstrong G (2013) The function analysis diagram: intended benefits and coexistence with other functional models. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **27**, 249–257.

Booth JW, Reid TN, Eckert C and Ramani K (2015) Comparing functional analysis methods for product dissection tasks. *Journal of Mechanical Design* **137**, 081101, 1-10.

Campbell MI, Cagan J and Kotovsky K (2003) The A-design approach to managing automated design synthesis. *Research in Engineering Design* **14**, 12–24.

Carroll JM (2000) Five reasons for scenario-based design. *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences, HICSS-32*. Hawaii: IEEE.

Chakrabarti A and Bligh TP (1996) An approach to functional synthesis of mechanical design concepts: theory, applications and merging research issues. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **10**, 313–331.

Chen Y, Liu ZL and Xie YB (2012) A knowledge-based framework for creative conceptual design of multi-disciplinary systems. *Computer-Aided Design* **44**, 146–153.

Chen Y, Zhao M, Xie YB and Zhang ZN (2015) A new model of conceptual design based on scientific ontology and intentionality theory. Part II: the process model. *Design Studies* **38**, 139–160.

Cole EL (1998) Functional analysis: a system conceptual design tool [and application to ATC system]. *IEEE Transactions on Aerospace and Electronic Systems* **34**, 354–365.

Engelenburg S, Janssen M and Klievink B (2019) Designing context-aware systems: a method for understanding and analysing context in practice. *Journal of Logical and Algebraic Methods in Programming* **103**, 79–104.

Erden MS, Komoto H, Van Beek TJ, D'Amelio V, Echavarria E and Tomiyama T (2008) A review of function modeling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **22**, 147–169.

Friedenthal S, Moore A and Steiner R (2006) OMG systems modeling language (OMG SysML) tutorial. *INCOSE Intl. Symp.*, 11 July 2006, Orlando, Florida, Vol. 9, pp. 65–67.

Friedenthal S, Moore A and Steiner R (2014) *A Practical Guide to SysML: the Systems Modeling Language*. Burlington: Morgan Kaufmann.

Hoffmann HP (2011) Model-basedsystems engineering with rational rhapsody and rational harmony for systems engineering. *Deskbook Release* **3**.

Hong JY, Suh EH and Kim SJ (2009) Context-aware systems: a literature review and classification. *Expert Systems with Applications* **36**, 8509–8522.

Jacobson I (1993) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Bengaluru: Pearson Education India.

Kaindl H (2000) A design process based on a model combining scenarios with goals and functions. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* **30**, 537–551.

Kitamura Y, Kashiwase M, Fuse M and Mizoguchi R (2004) Deployment of ontological framework of functional design knowledge. *Advanced Engineering Informatics* **18**, 115–127.

Liu ZL, Zhang ZN and Chen Y (2012) A scenario-based approach for requirements management in engineering design. *Concurrent Engineering: Research and Applications* **20**, 99–109.

Mhenni F, Choley JY, Penas O, Plateaux R and Hammadi M (2014) A SysML-based methodology for mechatronic systems architectural design. *Advanced Engineering Informatics* **28**, 218–231.

Pahl G and Beitz W (2007) *Engineering Design: A Systematic Approach*, 3rd Edn. London: Springer-Verlag.

Pohl K (2010) *Requirements Engineering: Fundamentals, Principles, and Techniques*. New York: Springer Publishing Company, Inc.

Roozenburg NFM and Eekels J (1995) *Product Design: Fundamental and Methods*. West Sussex: John Wile & Sons.

Rumbaugh J, Blaha M, Premerlani W, Eddy F & Lorensen WE (1991) *Object-Oriented Modeling and Design*, **Vol. 199**. Englewood Cliffs, NJ: Prentice-Hall.

Suh NP (2001) *Axiomatic Design: Advances and Applications*. New York: Oxford University Press.

Umeda Y, Takeda H and Tomiyama T (1990) Function, behaviour, and structure. In Gero JS (ed.), *Applications of Artificial Intelligence in Engineering V*, Vol. **1**. Berlin: Computational Mechanics Publications/Springer-Verlag., pp. 177–193.

Umeda Y, Ishii M, Yoshioka M, Shimomura Y and Tomiyama T (1996) Supporting conceptual design based on the function-behavior-state modeler. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **10**, 275–288.

Yuan L, Liu YS, Sun ZF, Cao YL and Qamar A (2016) A hybrid approach for the automation of functional decomposition in conceptual design. *Journal of Engineering Design* **27**, 333–360.

Fajun Gui is currently a PhD student in the School of Mechanical Engineering at Shanghai Jiao Tong University. He received his bachelor and master degrees from Chang'an University. His research interests include conceptual design, computer-aided design, and systems engineering.

Yong Chen is a Professor of Engineering Design in the School of Aeronautics and Astronautics at Shanghai Jiao Tong University. He received his bachelor and PhD degrees from Zhejiang University. He joined Shanghai Jiao Tong University as a postdoc in 2004. He was a visiting scholar in the IMPACT Laboratory at the University of Southern California for 1 year. Dr. Chen's research interests include many aspects of engineering design research, in particular, conceptual design, design innovation, design knowledge reuse, computer-aided design, and systems engineering.