Generation of Assurance Cases for Medical Devices

Chung-Ling Lin and Wuwei Shen

Abstract In safety critical systems, the manufacturers should provide compelling and comprehensible arguments to demonstrate that their system is well designed and safety of the system to the public is guaranteed. These arguments are usually represented by an assurance case. However, one of challenging issues facing the safety critical industry is how to produce an assurance case that provides a set of well-structured arguments connecting safety requirements and a body of evidence produced during software development. In this paper, we take the medical systems industry into account to illustrate how an assurance case can be generated when a software process is employed. In particularly, we consider the Generic Insulin infusion Pump (GIIP) to show how an assurance case can be produced via during a popular software development process, called Rational Unified Process (RUP).

Keywords Assurance cases · UML profile · OCL · Medical device software · Safety critical system

1 Introduction

One of the most challenging issues facing the safety critical industry is how to develop an assurance case providing a compelling and comprehensible argument to demonstrate that a safety critical system is well designed so its safety is guaranteed when it is in use. In safety critical domains, there are many international standards where many safety-related requirements are specified. A well-designed assurance case should successfully link the evidence to the specific safety objective of the system in a convincing way. But, how to produce such an assurance case has become an important issue. Many manufacturers have found the generation of an assurance case after a software system has been developed is quite time consuming

C.-L. Lin (✉) · W. Shen
Department of Computer Science, Western Michigan University,
Kalamazoo, MI, USA
e-mail: chung-ling.lin@wmich.edu

W. Shen
e-mail: wuwei.shen@wmich.edu

R. Lee (ed.), *Computer and Information Science*, Studies in Computational Intelligence 566, DOI 10.1007/978-3-319-10509-3_10

and error-prone. One of the important reasons is that developers should recall the details made during the software development process in order to build an argument linking the evidence and the corresponding claim(s). Then, a question has been raised: can an assurance case be generated when a specific software process has been employed?

In this paper, we take medical device software into account. In the medicals device industry, all medical devices must pass the FDA pre-market review before a new product can be deployed to the market because the FDA regulators are entitled to ensure that each new product is safe and reliable to the public. To better design a software system embedded into a medical device, some international standards such as ISO14791 [1] have been proposed. Recently, some guidance documents based on a specific medical device such as infusion pumps were released [2]. These standards and guidance documents aim to help medical device manufacturers to design medical device software according to recommendation in these documents. Consequently, the quality of medical device software can be improved.

However, how to conform to the regulation-requirements in these standards and guidance-has bothered many medical device manufacturers. In the medical device industry, many companies have already put in place their own cultures such as their own policies and processes to achieve the objectives of the regulation. They have found that it is a painful process to build a compelling assurance case which consists of information scattered over a large body of artifacts such as the hazards-analysis report and test and validation report, which they have produced before. Even worse, when constructing an assurance case, developers must spend more time to recall many design decisions made before. For instance, why a requirement has been decomposed into several sub-requirements and what criterion has been used at that time. Recall of these design decisions is always time consuming and error prone. Thus, the construction of an assurance case becomes one of persistent complains from medical device manufactures and stifle the motivation and creativity of manufacturers in building safer and more reliable medical devices.

In this paper, we strive for a new method that can automatically combine a specific software development process with the assurance information to reduce the burden of medical device manufacturers. We consider the goal structure notation (GSN) [3] as a backbone to build an assurance case. In order to integrate a specific development process employed by a medical device manufacturer, we employ the Rational Unified Process (RUP) [4] as a representative development process. We illustrate our approach with the case study of the Generic Insulin Infusion Pump (GIIP) [5], which we co-developed with FDA staff in the past years. In fact, in our previous work, the traceability established between different artifacts produced during the RUP process enable to track a system's design downstream to the implementation and upstream to the rationale. The traceability information leverages the understanding of how a system has been designed to satisfy the relevant safety requirement and thus expedite the regulator review process. Simultaneously, the traceability information becomes an integral part of constructing an assurance case.

The remainder of this paper is organized as follows: In Sect. 2, we introduce an assurance case and its GSN notation. In Sect. 3, we employ the Generic Insulin Infusion Pump case study to illustrate how to produce an assurance report when the Rational Unified Process is applied. Some related work is discussed in Sect. 4, and we draw a conclusion in Sect. 5.

2 Assurance Case and Its GSN Notation

An assurance case is important for safety critical systems in that it provides an argument from the developers about why a safety critical system can work well when it is in use. In the medical device industry, an assurance case of a medical device should demonstrate that the device does not harm a patient but also improve a patient's health. In order to assure the correctness of safety-critical software systems, a convincing assurance case should consist of the following elements:

- Claim. A claim is a statement that claims about some important property of a system such as safety and security.
- Argument. An argument is reasoning of why a claim can be supported. The reasoning can be done via the justification between claims and sub-claims or claims and evidence.
- Context. A context gives assumptions made about the whole assurance case.

A convincing and valid argument of why a system meets its assurance requirements is the heart of an assurance case. An assurance case should consist of extensive references to evidence used by the system. In general, an assurance case is a collection of claims, arguments, and evidence that are created to support the contention that a system will satisfy the particular requirements.

Certain claims can be directly supported by evidence, which usually refers to external documents collected by some systematic methods and procedures. In general, a structure of an assurance case is a tree-like structure with the top element as the root claim. Currently, there are two different notations to denote an assurance case, i.e. Goal Structuring Notation (GSN) [3] and Claims-Arguments-Evidence (CAE) Notation [6, 7].

In this paper, we adopt the GSN notation to represent an assurance case. In GSN, a rectangle node represents a claim, such as C0 ("All relevant hazards have been considered") in Fig. 1. A parallelogram node represents argument reasoning. In Fig. 1, the parallelogram A0 represents that the following argument: the decomposition of claim C0 is based on the categories recommended by the Guidance for Infusion Pump, i.e. "Total Product Life Cycle: Infusion Pump—Premarket Notification Submission" (abbreviated as Guidance in the rest of the paper) [2]. A rounded rectangle node denotes the relevant information used in an assurance case. For instance, C1 in Fig. 1 denotes the Guidance [2]. A diamond decorator node represents that information related to the node will be supported later. Finally, a circle node denotes evidence.

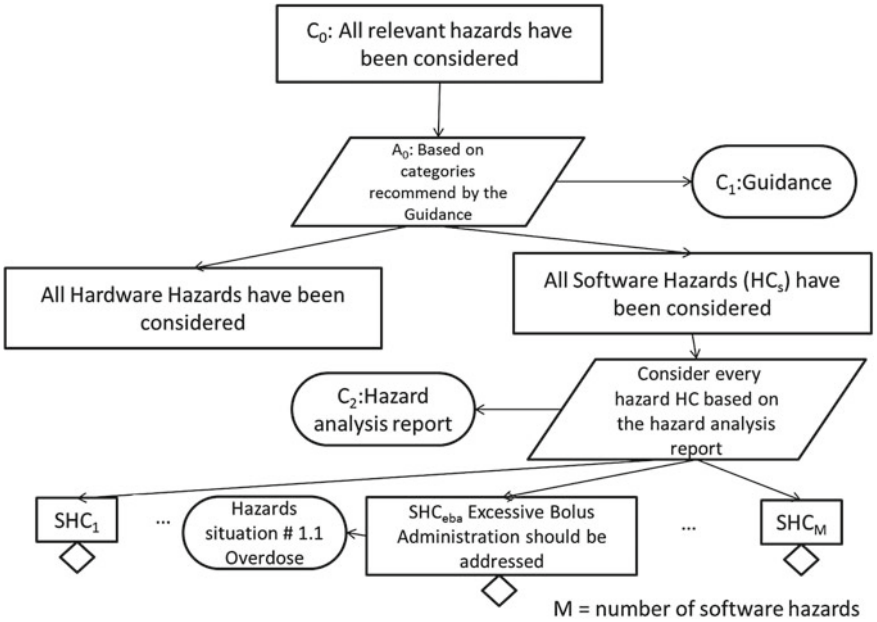

Fig. 1 *Top* level structure of an assurance case for GIIP

3 Development Process for GIIP

We apply the Rational Unified Process (RUP) to develop the Generic Insulin Infusion Pump case study due to the following several reasons. First, RUP is an iterative software development process created by Rational Software Cooperation, which was bought by IBM in 2003. It consists of various types of activities, each of which produces one or several different artifacts. RUP can be adapted to satisfy different needs during a software development process. Second, RUP has a good tool-based support. Various tools from IBM support different activities during software development. For example, RequisitePro [8] supports to produce artifacts at early stage of a software development process while Rational Software Architect (RSA) [9] can leverage the model design and software implementation. Because of the above reasons, RUP has gained a great popularity in industry.

In general, the RUP has four phases for a software development cycle. The key to the RUP is that a traditional software development process lies within all of the four phases, each of which has its own objective and milestone at the end. The milestone of each phase can be checked via the corresponding artifacts to validate whether the objective has been accomplished or not. These four phases are: Inception, Elaboration, Construction, and Transition. During the Inception phase, the primary objective is to scope the system adequately as a basis for validating initial costing and budgets. The primary objective of the Elaboration phase is to mitigate the key

risk items; and at the same time, domain analysis is made and the architecture of the project gets its basic form. Consequently, some design artifacts such as a USE CSAE model should be available for the later phases. The primary objective of the Construction phase is to build the software system. The main activities should include the bulk of coding activities. As a result of the activity, software implementation as software artifacts should be done. Finally, during the Transition phase, the objective is "transit" the system from development into production, making it available to and understood by the end user. The main activities include to beta testing the system to validate it against the end users' expectations. The corresponding test and validation report should be produced.

The GIIP problem was proposed as a case study by FDA to study how a software system embedded into an insulin infusion pump can be designed and validated. In addition to the requirements related to an insulin infusion pump application, a software system designed for the GIIP application should also consider some specific standard or guidance documents in the medical device domain, such as "Total Product Life Cycle: Infusion Pump—Premarket Notification Submission" [2]. In this paper, we will demonstrate how to generate an assurance case when the RUP process has been employed.

First, an assurance case is necessary for each premarket submission from an infusion pump manufacturer. According to the Guidance, it says: "In making this demonstration of substantial equivalence for your infusion pump, FDA recommends that you submit your information through a framework known as an assurance case or assurance case report. An assurance case is a formal method for demonstrating the validity of a claim by providing a convincing argument together with supporting evidence." Furthermore, a claim, according to the Guidance, is "a statement about a property of the system". To make a claim, the Guidance identifies 8 different hazard categories. FDA further requires each submission to "clearly describe the method used to analyze the hazards and each hazardous event mitigation". Next, we illustrate how an assurance case can be generated during the RUP process. Due to the space limit, we cannot show all the artifacts produced during the RUP. We concentrate on the Inception/Construction Phase, which mainly produces the requirement document, feature description document, use case report, and a sequence diagram. As the starting point, we claim that all relevant hazards have been considered during the Construction Phase. Then, this claim can be decomposed into eight sub-claims according to the Guidance such as Hardware Hazards. The argument of this decomposition is shown by A0 and C1 in Fig. 1. In this paper, we only consider Software Hazards denoted as HCs. The Software Hazards can also be divided into some sub-subclaims and in this case we consider the "Excessive Bolus Administration" denoted as SHC_{eba}.

During the Construction Phase, first the safety requirement document should be produced. In this document, all safety requirements related to an infusion pump should be addressed. In the case of claim SHC_{eba}, the following argument is built: "Each hazard should be well addressed in a hazardous event mitigation way via safety requirements". In this argument, the phrase "well addressed" means as follows: the disjoint of relevant subclaim/sub-arguement/evidence should be complete (exhaustive) while the conjoint of the subclaim/sub-arguement/evidence should be

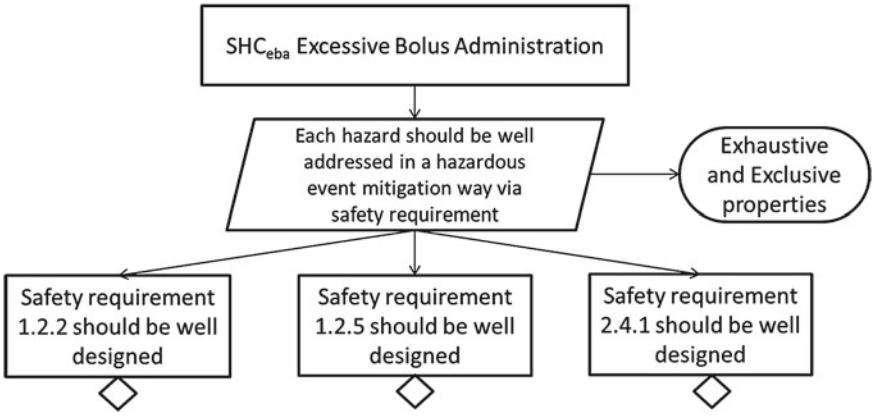

Fig. 2 Structure between SHCeba and safety requirements

empty (exclusive). More specifically, during the Inception Phase, a design decision about safety requirements should be made as follows: all safety requirements should satisfy the exhaustive and exclusive properties. If there is only one safety requirement produced:

- 1.2.2 The pump shall allow the user to set at least two basal profiles at the same time, and require the user to activate no more than one profile at any single point in time.

then the above argument is not complete since how the hazard SHC_{eba} can be mitigated when a user set only one basal profile is not addressed. So, when developers apply the RUP to produce the safety requirements documents, the flaw in the assurance case is observed. Figure 2 shows the structure between hazard SHC_{eba} and the related safety requirements.

Assume that the other safety requirement related to when a user sets one basal profile is considered continue to apply the RUP to design features document. A features document lists all the features that an infusion pump system should achieve. Similar to the previous argument structure, we should build an argument that "Each safety requirement should be well addressed via system features". In this case, we assume that the following features are produced:

- R3113-2: The component shall be able to manipulate the Basal Profiles record in the following ways:
  - Add a new profile to the record if doing so will not exceed the record's capacity (see Requirement R3116 for more detail).
- R3114: The component shall not accept any invalid basal profile that the user programs into the Basal Profiles record. A valid basal profile includes one or more segments, each of which is defined as a pair (effective period, basal rate), where the basal rate shall range from 0.05 Unit/h to x Unit/h and the effective period is

defined by its starting time (of day) and ending time (of day). The ending time of a basal segment shall be no earlier than its starting time. In a valid basal profile, the effective periods of two distinct segments shall not overlap with each other, and effective periods of all segments shall cover 24 h of day.

- R3115: If the Basal Profiles record is not empty, the component shall allow the user to activate (via IUID) any profile stored in the record, i.e. scheduling basal delivery according to this profile. If the profile to be activated is not currently active, the component shall deactivate the currently active one first and then activate this profile. The activation of basal profiles has to be confirmed by the user before it can take effect.
- R3116: When the user selects (via IUID) to program a new basal profile, the component shall check if the Basal Profiles record has reached its storage limit. If so, the component shall instruct IUID to prompt the user to either quit programming or select an existing profile to override. Otherwise, the component shall acquire the newly programmed profile from IUID and check its validity. If the new profile is valid, the component shall add it into the Basal Profiles record without affecting other profiles in the record or selection of the active basal profile. If the new profile is not valid, the component shall reject this profile and inform IUID of the rejection.

In the case of claim safety requirement 1.2.2 is well designed, two arguments are built based on the fact that each safety requirement is mapped to a set of features that satisfy the exhaustive and exclusive properties and each safety requirement should be connected with related features. For the first argument, the exhaustive and exclusive properties mean that each feature considers one scenario for a safety requirement. For example, safety 1.2.2 can be decomposed into the following scenarios that should be addressed by the features: (1) A function that allows the user to add new profiles, (2) The definition of a valid profile that can be accepted by the system, (3) A function that allows the user to activate profile, and (4) the constraint of activating a profile. In this example, R3113-2 considers adding a new profile, which is further explained by R3116. These two features are related to the scenario 1 of safety 1.2.2. The scenario 2 is addressed by R3114 which defines a valid structure of a profile. When a basal profile is added, R3115 shows the feature of setting an added basal profile to be an active basal profile and is related to scenarios 3 and 4 of safety 1.2.2. From these descriptions, the design decision is based on that these features should satisfy the exclusive and exhaustive properties. Namely, any two of these features have no overlapping, and safety requirement 1.2.2 is completely addressed by these features. For the second argument, the traceability links between safety requirements and system features are considered as evidence to support this argument. Figure 3 shows the arguments and sub-claims decomposed from safety requirement 1.2.2. Figure 4 shows the traceability matrix between safety requirements and system features.

Next, we establish an argument via a design decision made about features via some artifacts produced during the next activity of the RUP as shown in Fig. 5. In this case, we enter the Construction Phase in the RUP which mainly produces some design artifacts. Some important artifacts to be produced include use case reports,

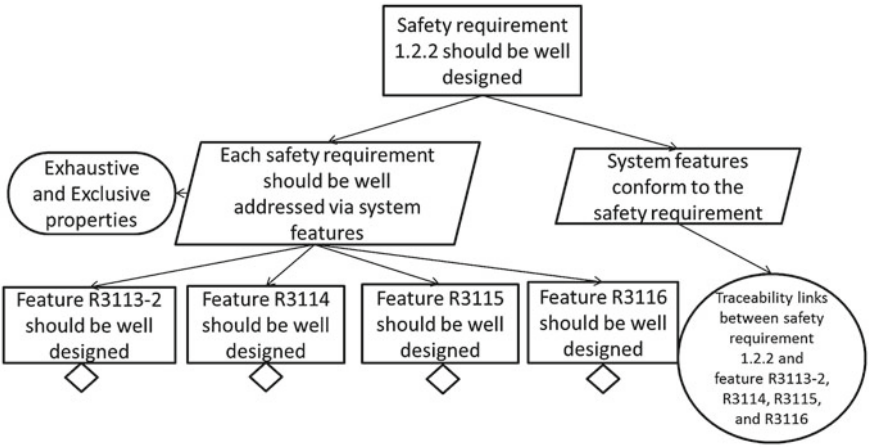

Fig. 3 Structure between safety requirement 1.2.2 and features

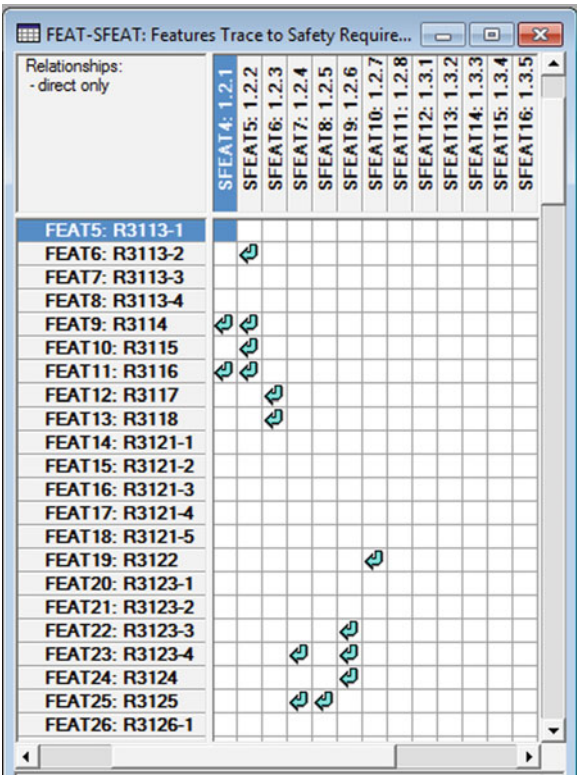

Fig. 4 Traceability matrix between safety requirements and system features

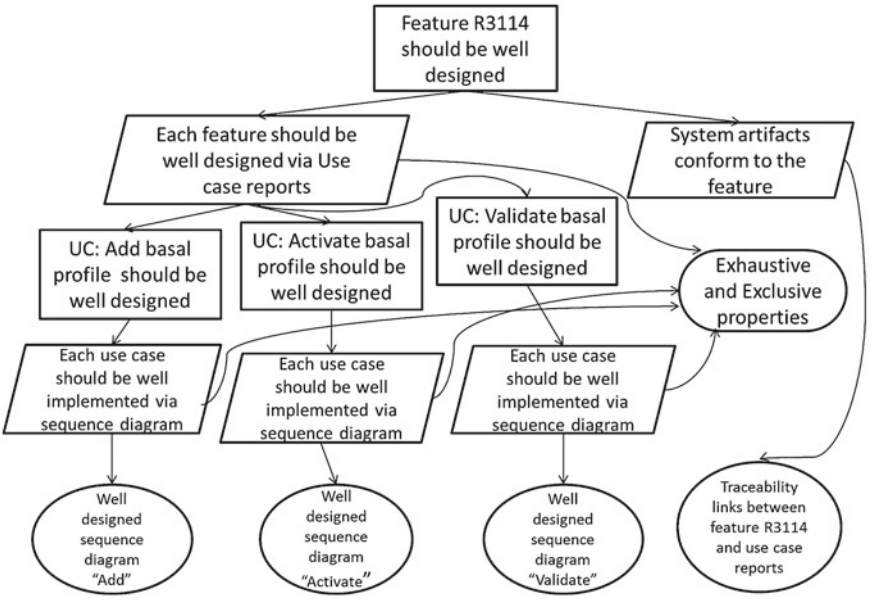

Fig. 5 Structure of feature R3114


2.1 UCL22.2 Basic Flow
2.1.1 BEGIN
This use case starts when user input a new basal profile through IUID
2.1.2 System response
Profile Management: Check if enough space for new basal profile
2.1.3 System response
Profile Management: Validate new basal profile
2.1.4 System response
Profile Management: Add new profile to profile record
2.1.5 System response
Profile Management: Output the add success message to the user through IUID.]
2.1 UCL26.2 Basic Flow
2.1.1 BEGIN
This use case begins when user select a existing profile to active through an event to active a profile
2.1.2 System response
Profile management: De-active current active profile
2.1.3 System response
Profile management: Set new active profile
2.1.4 System response
Schedule management: Set to delivery mode
2.1.5 System response
Profile management: output "Activate success" message to the user through IUID.]
2.1 UCL26.2 Basic Flow
2.1.1 BEGIN
The use case starts when the component receiving a new basal profile from the user
2.1.2 System response
Profile management: Check each segment contains a pair of effective period and basal rate. If not, return invalid
2.1.3 System response
Profile management: Check the basal rate is from 0.05 unit/hour to the maximum value define by the user. If not, return invalid.
2.1.4 System response
Profile management: Check if start time late then end time for each segment. If yes, return invalid.
2.1.5 System response
Profile management: Check if there exists overlap between two or more segment. If yes, return invalid
2.1.6 System response
Profile management: Check if the effective period covers 24 hours of a day. If not, return invalid.
2.1.7 System response
Profile management: Return Valid profile.]

Fig. 6 Use case reports related to feature R3114

and sequence models. Likewise, we can continue to consider the similar arguments, each feature should be well designed via a use case report and each feature should be connected with related system artifacts. Here we use Feature R3114 as an example. Figure 6 shows three use case reports that targets on the three different scenarios related to R3114 and Fig. 7 shows the traceability matrix between features and use case reports.

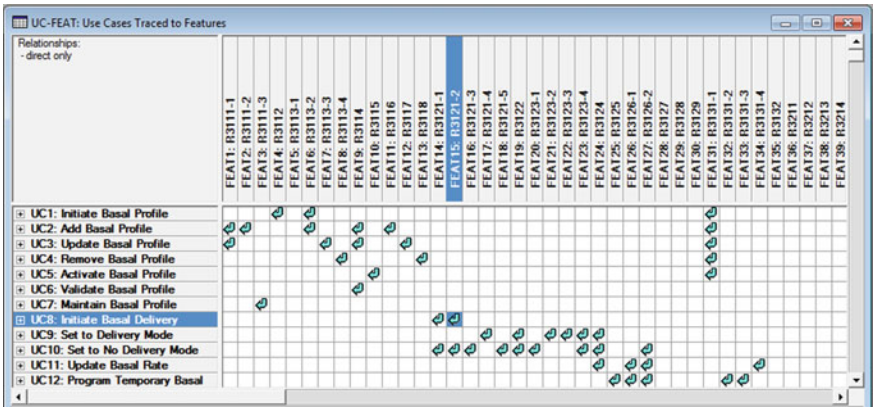

Fig. 7 Traceability matrix between system features and use case reports

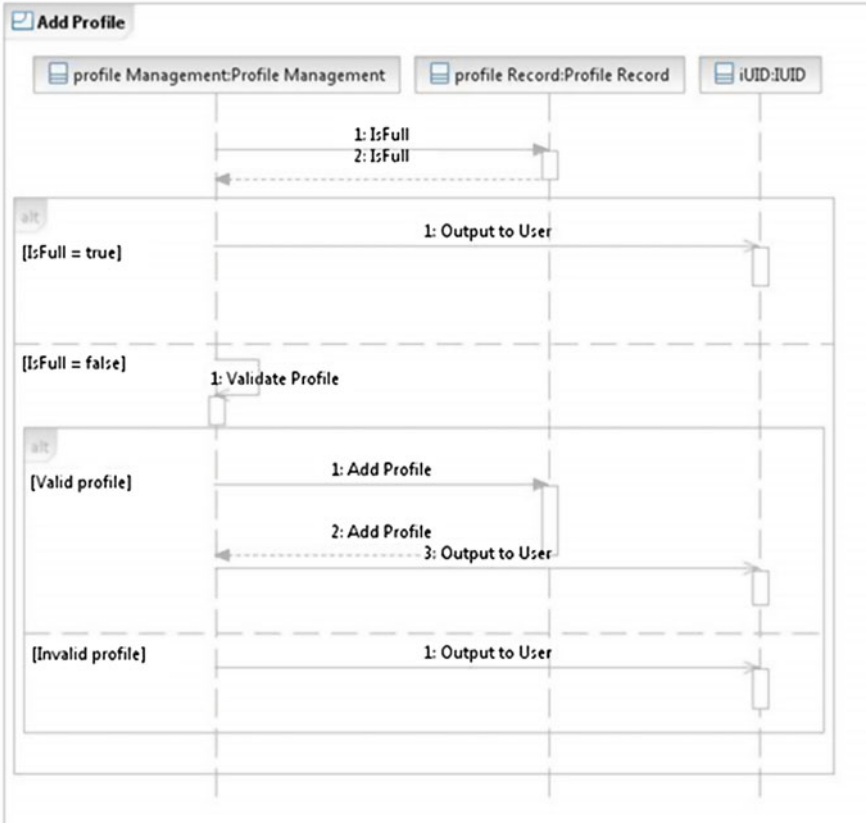

Fig. 8 Sequence diagram for add profile

Last, due to space, we assume that the RUP process stops at the second argument, i.e. each use case is designed. In the real development process, we continue to develop the system in which a design class model and a final C++ implementation are produced. Under this assumption, these sequence diagrams serve as evidence of the whole argument hierarchy. The logical connection between a high-level hazard and the final sequence diagram reflects how each hazard has been analyzed and mitigated. We show a sequence diagram related to Use Case "Add basal profile" in Fig. 8. Here, it is worth discussing what "each use case should be well designed" means. We follow the definition of exhaustive and exclusive properties to design sequence diagrams from use case models. In order to comply with the exclusive property, every use case is implemented by different sequence diagram. Each sequence diagram describes different system interactions and no overlapping between any two sequence diagrams. To fulfill the exhaustive property, every basic flow and alternative flows of a use case can be mapped to different message in the sequence diagram. Every lifeline designed in the sequence diagram is already defined in the use case. That is, all of the information in the use case is completely captured and implemented by the sequence diagram.

4 Related Work

Construction of an assurance case has been a hot topic since the software-intensive safety-critical systems are becoming popular and increasing complex. How to build a compelling argument draws more attention in safety-critical industry. Many challenges have remained in the assurance case community [10]. Various techniques have been proposed to address the problems and difficulties in the construction of an assurance case.

Hawkins et al. presented a new approach that incorporates a confidence argument in a safety argument [11]. Traditionally, a safety argument includes the confidence about this argument and the separation of a confidence argument can make both arguments clarity of purpose and helps to avoid some redundancy in arguments and evidence.

Jee et al. discussed the construction of an assurance case for the pace-maker software via a model-driven development approach [12], which is similar to ours. However, their approach emphasizes on the later stage of a software development such as a timed automata model as a design model and C code as implementation language. The approach considers the application of the results from a model checker called UPPAAL and measurement based on timing analysis as evidence.

Attwood et al. proposed to apply a linguistic model of understanding to identify mismatches and provide guidance on composition and integration when constructing an assurance case. Dominguez et al. presented an experience in developing an assurance case for a rebreather system via the Goal Structuring Notation. Ray et al. demonstrate an approach for safety assurance case argumentation based on the Generic Patient Analgesic Pump (GPCA). As a governing agency in medical device

industry, a document recently released by FDA recommends the submission of an assurance case as part of pre-market submission.

Rushby observed that an assurance/safety case is an argument that helps increase confidence in the soundness of a given case. One of the most important ways to check an argument is formal logic. Thus Rushby proposed a method to formalize safety cases and one important ramification of this work is the use of automated tools to check the logical soundness of a safety case [13]. Another advantage of formalization is the development of metamodel for various tactics of argument.

The automotive industry is another specific domain requiring the construction of a safety case. For example, the automotive standard ISO26262 [14] requires the development of a safety case for electrical and/or electronic (E/E) systems whose malfunction has the potential to lead to an unreasonable level of risk. Many researchers have developed various strategies to design a safety case for an automotive software system. Birch et al. investigated the main argument structures of a safety case and the relationships among these structures when assessing functional safety in that ISO26262 does not specify how a safety argument should be evaluated in the functional safety assessment process [15]. Birch et al. emphasized on the product-based safety rational when constructing a safety argument.

Westman et al. demonstrated that the contract theory can be employed to construct safety requirements in ISO26262 [16]. Contracts are used to separate the responsibility of a system from its environment by imposing safety requirements on the environment as assumptions. To check an automotive software system against ISO 26262, contract theory provides the verification of consistency and completeness on the safety requirements.

Stürmer et al. proposed a novel approach to study whether an automotive software system is compliant with ISO26262 via reviewing software models [17]. Since model-based development has gain the popularity in the automotive industry, the early detection of model artifacts that violate the safety requirements in ISO26262 can greatly improve the quality of an automotive software system. Stürmer et al. combined an automated and manual review to detect any violation of ISO26262.

5 Conclusion and Future Work

Developing a software system that, software engineers can guarantee, satisfies the regulatory requirements is one of the most challenging issues facing the software engineering community. In this paper, we propose a novel approach which integrates the construction of an assurance case into a software development process. While we only consider the Rational Unified Process, our approach can be applied to all other software development processes employed by medical device manufacturers. We aim to save the time and labor to generate a convincing and compelling argument for a system and our approach cannot only be used by some regulatory bodies but also improve manufacturers' capability to understand the quality of a system they have designed.

While our approach is still in the preliminary stage, we think the approach paves the way to leverage the capability of regulatory review and even software certification via an assurance case in an automatic way. With the recent progress made in the Model Driven Engineering (MDE) community, we think some important techniques advanced in the MDE community can be adapted to regulatory review and software certification. In both regulatory review and software certification, one important issue is to investigate whether a system has achieved the claimed requirements. In fact, an assurance case has been widely proposed to establish an argument in a logically consistent fashion.

We have noticed that a metamodel for a structured assurance case, called SACM [18], has been proposed by the Object Management Group while GSN and CAE have been used as popular notations in the assurance case community. Obviously, the assurance case community does not lack of the notation to represent an assurance case. The introduction of SACM is in fact consistent with the latest development fashion in MDE, aiming to bridge the gap between the problem domain and the implementation domain.

On the other hand, a UML profile mechanism has been widely employed to model different artifacts produced during software development process. Also, a UML profile can model a development process. For instance, a UML profile for business modeling proposed by IBM aims at the application of UML notation to represent the artifacts produced during BPMN [19]. In fact, many industry companies have their own metamodel capturing the specific development process used in their company. Based on this fact, we think the application of the metamodels that represent a development process and an assurance case structure respectively can facilitate the regulatory review and software certification from the following two aspects.

First, inspired by the forward engineering features in MDE, we take the advantage of the two different metamodels that can help to generate an assurance case. One important feature of a metamodel is to leverage the ability of model transformation. Thus, using the model transformation techniques we can produce an assurance case from artifacts produced during a development process. In this case, developers can record all design decision made to produce various artifacts during a development process. Consequently, there is no additional time and effort to build an assurance case separately.

Second, the popular technique in MDE to retrieve a design model from an implementation motivates us to retrieve an assurance case from the produced artifacts. This so-called reverse engineering is quite useful when a system has been designed before some standards and guidance documents are proposed. Obviously, the metamodels can help to dive into text in the related documents that can be possibly established an argument structure. Next, thanks to the latest development in information retrieval, we can consider to apply some techniques such as vector space model to retrieve the relevant information to recover an assurance case.

An assurance case provides a powerful method for medical device manufacturers to convince some regulation agencies such as FDA that their system is compliant with safety requirements under some guidance documents or standards. Our future work based on the approach will concentrate on the application of the MDE

techniques. Thus, the generation of an assurance case can be done in an automatic and systematic approach so we should finally leverage the capability of regulatory review and software certification.

References

1. Medical devices—Application of risk management to medical devices, ISO 14971
2. US Food and Drug Administration, Guidance for Industry and FDA Staff-Total Product Life Cycle: Infusion Pump- Premarket Notification[510 (k)] Submissions. April 2010
3. Kelly, T., Weaver, R.: The Goal Structuring Notation—A Safety Argument Notation, in dependable systems and networks 2004 workshop on assurance cases (2004)
4. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional, Amsterdam (2003)
5. FDA, Generic Insulin Infusion Pump Functional Specifications (2011)
6. Adelard. The Adelard Safety Case Editor—ASCE. http://adelard.co.uk/software/asce/ (2003)
7. Bishop, P.G., Bloomfield, R.E.: The SHIP Safety Case Approach, in Safe Comp 95, pp. 437–451. Springer, London (1995)
8. Zielczynski, P.: Requirements Management Using IBM Rational RequisitePro. IBM Press, Upper Saddle River (2008)
9. Leroux, D., Nally, M., Hussey, K.: Rational software architect: a tool for domain-specific modeling. IBM Syst. J. **45**(3), 555–568 (2006)
10. Langari, A., Maibaum, T.: Safety Cases: A Review of Challenges (2013)
11. Hawkins, R., Kelly, T., Knight, J., Graydon, P.: A New Approach to Create Clear Safety Arguments, In Nineteenth Safety-Critical Systems Symposium. Southampton, UK (2011)
12. Jee, E., Lee, I., Sokolsky, O.: Assurance Cases in Model-Driven Development of the Pacemaker Software, LNCS 6416 (2010)
13. Rushby, J.: Formalization in Safety Cases. In Eighteenth Safety-Critical Systems Symposium, pp. 3–17 (2010)
14. CD ISO, Road vehicles-Functional safety, International Standard ISO/FDIS, vol. 26262 (2011)
15. Birch, J.: Safety cases and their role in ISO 26262 functional safety assessment. In Computer Safety, Reliability, and Security, pp. 154–165. Springer (2013)
16. Westman, J., Nyberg, M., Törngren, M.: Structuring safety requirements in ISO 26262 using contract theory. In Computer Safety, Reliability, and Security. pp. 166–177, Springer (2013)
17. Stürmer, I., Salecker, E., Pohlheim, H.: Reviewing software models in compliance with ISO 26262. In Computer Safety, Reliability, and Security. pp. 258–267, Springer (2012)
18. OMG. Structured Assurance Case Metamodel (SACM)—Version 1.0. http://www.omg.org/spec/SACM/
19. Johnston, S.: Rational UML Profile for business modeling, IBM Developer Works. http://www.ibm.com/developerworks/rational/library/5167.html, (2004)