A Study on Dependability Assurance in System Modeling

Qiang Zhi

Abstract

Models and architectures in software engineering are simplified representations that focus on certain properties of design objects. The description, visualization, and documentation for information systems can be simplified by modeling. Through modeling, abstraction degree of the system design will be increased, and verification at an early stage of system development becomes possible.

In this paper, we analyze and study the problem of dependability assurance in system modeling. We analyze the deficiencies of existing methods, and propose new methods to improve these deficiencies. In existing methods, the architecture diagram describing target system is different from assurance diagram, this complicates development and management. We propose an Intra Model Security Assurance method that integrates system architecture diagram and assurance case diagram to improve the efficiency of development and management. Besides, we propose a visualized assurance approach for Enterprise Architecture, which includes Technology Layer assurance, Application Layer assurance, and Business Layer assurance.

Another problem in existing methods is that the automaticity of dependability evidence for complex systems is inefficient. In a system with complex process interactions, the dependability of interactions among the system processes is often difficult to describe in existing methods, because this would increase the complexity of the description. We propose a composite safety assurance method, and use model checking to assure the dependability of interactions. Moreover, we provide a formalized procedure for dependability assurance, which boosts the confidence of system dependability.

Contents

Figures

Tables

1 Introduction

In software engineering, models and architectures are simplified representations that focus on certain properties of design objects. The description, visualization, and documentation for information systems can be simplified by modeling. Through modeling, abstraction degree of the system design will be increased, and verification at an early stage of system development becomes possible. Besides, system modeling at the early-stage of software development is meaningful to prevent rework caused by design mistakes. However, only modeling is not enough to ensure complete dependability.

In this paper, we conducted a study on dependability assurance in system modeling. We first analyze the previous methods to find out the shortcomings, and then propose new methods to improve these shortcomings. The concept that has to be mentioned here is assurance case. Assurance case is widely recognized as the fundamental document to certify safety-critical systems. The purpose of developing assurance cases is to ensure the safety of a system. This research will focus on how to ingeniously combine modeling and assurance case to ensure system dependability as the main work. Here is another concept that we have to mention is “dependability”. In this paper, we used safety, security, and dependability to represent reliability of a system. Safety is a system attribute that reflects the system's ability to operate without threatening people or the environment, for example, insulin pump, which is a medical equipment that will be introduced later. Security is a system attribute that reflects the system's ability to protect itself from external attack, for example, cloud storage, which will be introduced as a case study in this paper. Dependability is a measure of a system's availability, reliability, and its maintainability, and maintenance support performance, and, in some cases, other characteristics such as durability, safety and security.

This research is basically based on ArchiMate. ArchiMate is a visual modeling language for describing Enterprise Architecture. It is an open and independent modeling language to support the description, analysis and visualization of architecture within and across business domains in an unambiguous way. In this paper, the composite safety assurance approach is modeled by using ArchiMate, system components, processes, relationships, and architectures can be well treated.

In Chapter 3 – Chapter 5, we introduced the integration method of system architecture and assurance case, the dependability assurance method of the three-layer enterprise architecture, and the safety assurance method in the design of medical equipment. In Chapter 6, we propose a 4-steps assurance approach for complex systems with interactions between processes.

Assurance cases for architecture diagrams are developed independently by using previous approaches. In Chapter 3, we propose a new method, to develop both assurance cases and architectures in the same diagram. It enables to efficiently assure security by reducing the cognition and operation gaps caused by manipulating different diagrams such as security assurance cases and architecture diagrams. The effectiveness of the proposed method is also showed by experimental evaluation. According to the experimental results, proposed approach is superior to the previous approach for assuring security.

Besides, to ensure reliability between systems, describing both system architecture and assurance arguments among system elements is considered necessary. There are proposals for system architecture assurance, but using these previous methods often requires developing different diagrams with different editors. Because the visual sense of the previous methods is inadequate, errors readily occur when manipulating different diagrams. Therefore, it is essential that the assurance of dependability between components and systems is visualized and easy to understand. In Chapter 4, an integrated approach to describe the relationship between system actors and system architecture is proposed, and this assurance method is suitable for three-layer enterprise architecture. A case study is carried out and the comparison to the previous approach d* framework is also explained. The comparison results show that the proposed approach is more suitable for ensuring dependability of system architecture.

The content of this work can be applied to many aspects. For example, the safety of medical devices is critical, and safety assurance is necessary for the production and development of medical devices. For this, in Chapter 5, we propose a method to describe safety assurance between healthcare system components. A case study on the insulin pump system, which is a medical device, is carried out to explain the method. Moreover, a comparison with d* framework, which is a dependability assurance approach, is explained to show the effectiveness of the method. The comparison results show that the proposed approach is more suitable for ensuring safety in safety-critical healthcare system architecture.

Moreover, a problem for previous system assurance methods, is that the relationships between system architectures and assurance arguments are ambiguous. Also, there is no reasonable verification on whether there are problems with the state transition of the system process. In Chapter 6, we proposed a new approach for system assurance, and a formal process is provided to illustrate this approach. This approach consists of four steps: visualization of the interactions between components, processes model checking, dependability case creation, and composite safety assurance. Also, a case study on an

automatic driving system is carried out to confirm the effectiveness of this approach.

In Chapter 3, the proposed method Intra Model Security Assurance (IMSA) was applied to Technology Layer and Application Layer in the system architecture. In Chapter 4, we simplified the IMSA method to apply to the Business Layer in the system architecture. Then, we defined a composite assurance method for the safety-critical systems in Chapter 5. At last, we provide a formalized procedure for safety assurance, which boosts the confidence of system safety. The following table shows the correspondence between issues and achievements.

Issues	Methods	Chapter
System architecture diagram is different from assurance case diagram	IMSA	3
There is no dependability assurance method for hierarchically structured systems	Visualized Assurance Approach for Enterprise Architecture	4
Dependability assurance for complex systems	Composite Safety Assurance	5
The manual documentation for the dependability evidence of complex systems is inefficient	Composite Safety Assurance using Model Checking	6

2 Background

System modeling is indispensable in software development [1]. Unified Modeling Language (UML) [2] is a modeling language for object-oriented analysis and design, and Systems Modeling Language (SysML) [3] encompasses the entire system, including software and hardware. However, these traditional modeling languages do not directly implement dependability assurance between components and systems.

As system assurance is the discipline that identifies and mitigates or removes exploitable vulnerabilities, it is increasingly important for both commercial and governmental activities. In the process of system development, dependability analysis for system components and functions is generally considered to be indispensable [1]. As for system assurance, conventionally, the certifier determines the safety of a system through checklist items, there were few explicit discussions about why the system was safe if the checklist items were met. Therefore, it is important to discuss not only the procedures and tests, but also why those procedures and tests maintain safety. Assurance case [4] is a document for discussing the safety of a system based on the test results and verification results as evidence. An assurance case is a structured argument, supported by evidence, used to demonstrate that a system exhibits some complex properties such as safety, security or reliability in a given environment. Assurance cases focused on safety are usually called “safety cases”.

Assurance cases had been developed separately to assure safety, security, and dependability for architectural artifacts. We have proposed security and dependability case development methods so far. These methods assumed assurance cases are different artifacts from those of architectural diagrams. Using different diagrams for describing assurance cases leads to some problems. For example, linking architectural elements and the corresponding claims in assurance cases is necessary, understanding and memorizing different diagram structures is necessary. Besides, manipulating different diagrams simultaneously is also necessary.

We know that the development of assurance case is often based on system architecture or system model, it is necessary to mention the model checking regarding the safety of system model. In computer science, model checking [5] is exhaustively and automatically checking whether a given model meets a given specification. NATO AEP-67, defines system assurance as confidence to systems [6]. So far, system diagrams used to represent system components, processes, relationships, and architectures have been developed independently, the visualization of arguments between system architecture and assurance cases is insufficient, and the relationships between system architecture and assurance case are often ambiguous. Model checking is a method for algorithmic verification of formal

systems, and it is often applied to hardware design. As software is undecidable, algorithmic approaches alone may not be perfect and may not prove or disprove. But for systems with explicit state transitions, model checking is also applicable.

It must be mentioned here that the object of assurance case is not only the system architecture, but also the interaction between system elements, called interdependency. Interdependency management is important for developing contemporary systems that comprise acquired components and communications. To assure the dependability of a system A, assuring the internal dependability of A, interdependency of A, and internal dependability of all the systems with which A is interdependent will be necessary. Although Yu [7] showed that the network of intentions among the actors could be represented using the i*framework, the problem of how to treat the dependability of systems has not been solved. Some other methods can be combined with system models to achieve dependability assurance. When a system architecture contains assurance cases, d*framework [8] can be used to assure system dependability. An assurance case is a goal tree that shows a top claim based on decomposition into sub-claims, with evidence to show the validity of the bottom claims. The general form of an assurance case is a goal tree including a Top Claim, Sub Claims, and Bottom Claims. The validity of the bottom claims should be proven by evidence.

Next, I would like to introduce some important concepts and research progress on dependability assurance of system architecture.

The Goal Structuring Notation (GSN) was proposed [9] and widely used to develop assurance and safety cases. The Open Group (TOG) Real Time & Embedded Systems Forum focuses on standardizing for high assurance, secure, and dependable open systems . The purpose of the O-DA (Open Dependability through Assuredness) [10] standard is providing the concept of modeling dependability, building assurance cases, and achieving agreement on accountability on the event of actual or potential failures. Dependability Case (D-case) [11] [12], a derivative of GSN, is a technology and a tool used to describe dependability. In addition, when there is a need to describe the relationship between models, safety case construction [13] can be used. The argument patterns [14] have been proposed to help engineers develop assurance cases. A security argument pattern and security case based on common criteria [15] have been proposed for assuring security. Mobile security assurance approaches based on attributed GSN [16] had been proposed. Attributed GSN can manage values assigned to claim nodes to represent goal achievement ratio. These attributes are useful to compare solutions.

The more generic approach to generate assurance cases based on the elements and relationships of the architecture described in ArchiMate [17] [18] had been proposed.

ArchiMate is a visual modeling language for describing Enterprise Architecture. UML (Unified Modeling Language) focuses only on modeling software. SysML (Systems Modeling language) focuses on systems modeling. ArchiMate can be used to model business, application, and technology architectures as well as motivation aspects. The motivation aspects include business drivers, principles, assessments, goals, requirements, and constraints. The motivation elements can be used to represent claim goals and evidence. A non-functional requirements (NFR) framework [19] can also be modeled as the function of motivation elements in ArchiMate. The elements in the business and technology layers cannot be represented by UML or SysML. SysML has the capability of requirements diagram description, but different types of elements are indistinguishable in SysML. However, the elements in the business layer, application layer, and technology layer can be related by the motivation elements in ArchiMate.

There are many studies on ArchiMate. Grandy and others [20] proposed an integrated approach on EA and security risk management. Their approach was limited by the capability of ArchiMate 2.0 that is a former version of the current Archi-Mate 3.0. For example, their approach did not use the influence relationship between requirements and countermeasure concepts. Feltus and others described a meta-model for SCADA (Supervisory Control and Data Acquisition) systems and then by using ArchiMate they described components behavior for mitigating cyber-crime actions. The approach did not clarify the influence impacts on assurance between particular crime mitigation components and risk management policies. Korman and others [21] compared ArchiMate 2.0 concepts with various risk assessment methods. They clarified the concept coverage of ArchiMate. Although they concluded that ArchiMate models might be a source of guidance for risk assessments, they did not clarify the detailed security assurance method using ArchiMate. Band [22] published a White Paper to provide guidelines to ArchiMate users on modeling enterprise risk and security. The contribution of this White Paper has been called a “Risk and Security Overlay” (RSO) of the ArchiMate language. Abbass and others [23] described an Information System Security Risk Management (ISSRM) model by the constructs of ArchiMate. Mayer and Feltus evaluated the risk and security model of ArchiMate using ISSRM [24]. The completeness of the RSO visual expressions has been evaluated with regard to ISSRM and cognitive effectiveness by using the nine principles of Moody [25].

3 IMSA - Intra Model Security Assurance

3.1 IMSA

Although we developed a method [26] to generate assurance cases exhaustively based on architecture diagrams, the method has not been widely accepted by field engineers in Japan. Our investigations on Japanese engineers showed that they did not want exhaustive assurance cases, but wanted more focused local assurance cases. For the local assurance cases, it is troublesome to manage different two diagrams such as architectural and assurance case diagrams.

In this section, we describe assurance claims and evidence in a united architectural diagram, the local assurance of architecture will be realized by only using a united architectural diagram consolidating architecture and security case descriptions. In our opinion, assurance case elements could be represented in the architectural diagrams, so it could be easy to assure architectures because there is no need to change different diagrams to check elements among diagrams. Here we propose an Intra Model Security Assurance (IMSA) approach for developing security cases by relating elements of security case with those of architecture in architecture diagrams. Using the approach will reduce the gap between exchanging architecture and security case diagrams. The approach will also have the possibility of efficiently assuring security of architectures. To clarify the superiority of the proposed method against to the previous approach, a case study and an experiment are conducted. In the case study, diagrams for architecture and security case for both methods were developed and compared for a secure retrieval of cloud storage service. In the experiment, diagrams for architecture and security case for both methods were also developed and compared for Healthcare device and Smart house systems. For comparison, two groups of subjects are assigned to answer questions on both approaches. The time and correctness of questions are evaluated to compare both methods.

3.1.1 Meta model of Architecture and Security case

It is necessary to represent architecture and security case in the same diagrams. Figure 1 integrates meta-models of architecture and security case. The meta-model of architecture consists of the target of assurance, elements and relationships. The target of assurance represents the system as a whole. The meta-model of security case consists of target of assurance, property, risk, countermeasure, and evidence. The evidence will be realized by elements of the target system.

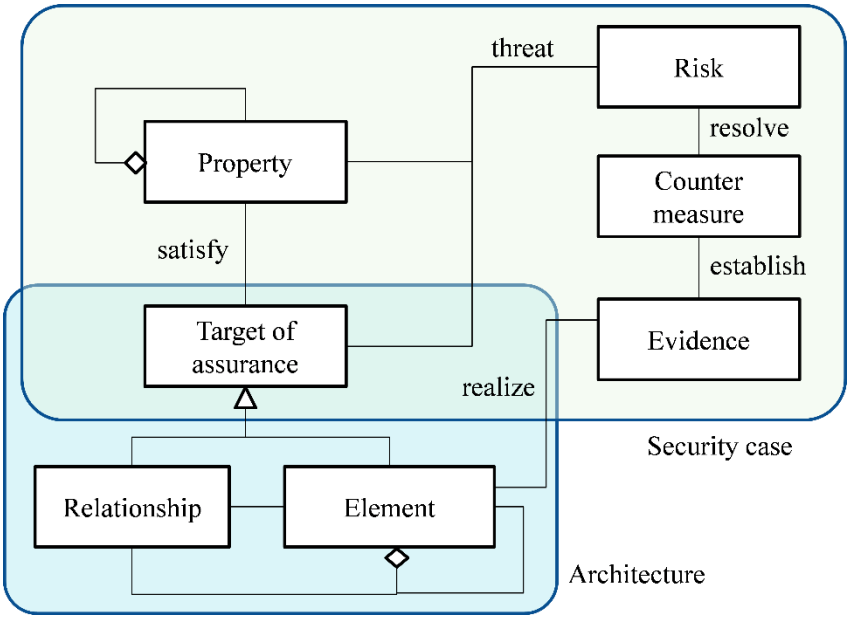

Figure 1 Meta model of architecture and security case

3.1.2 Using ArchiMate for Security Case

As ArchiMate provides elements to describe motivations of architectures. The motivation elements are driver, assessment, goal, and requirement. Table 1 shows an interrelationship among the metamodel, security case and motivation elements.

Table 1 Meta model, Security case and Motivation elements of ArchiMate

Meta model	Security case	Motivation elements
Property	Top Claim	Driver
Risk	Context	Assessment
Countermeasure	Sub claim	Goal
Evidence	Evidence	Requirement

In security case, property, risk, countermeasure, and evidence of meta-model are described by top claim, context, sub claim, and evidence, respectively. In ArchiMate, property, risk, countermeasure, and evidence of meta-model are described by driver, assessment, goal, and requirement, respectively. Figure 2 shows a generic example of security case configuration in ArchiMate. Although the figure only shows one risk for property, many numbers of risks can be allocated to the property. In the same way, other numbers of countermeasure and evidence can also be added for risk and countermeasure, respectively.

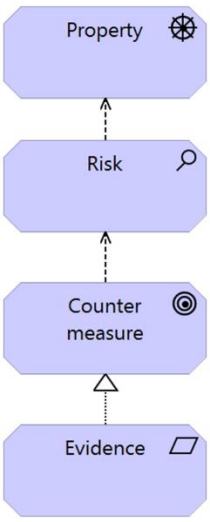

Figure 2 Security case configuration in ArchiMate motivation elements

3.2 Case Study

This section gives an example of secure retrieval on cloud storage, and explains in ArchiMate.

3.2.1 Target System

The target system is composed of a search server, a cloud storage server, and a user who wants to search files. All data stored on cloud storage server is encrypted, including the file that stores search information. The cloud storage server can't decrypt any data because it does not have decryption keys. The search server can only get the search information file from the cloud storage server and only has the decryption keys of search index. The search server decrypts the corresponding index file based on user's attributes so that the user only can search in the corresponding scope by his own attribute. The encryption algorithm in this target system is Ciphertext-Policy Attribute-Based Encryption, CP-ABE, which is a new Public-Key cryptography and it is suitable for data sharing. This target system is shown in Figure 3.

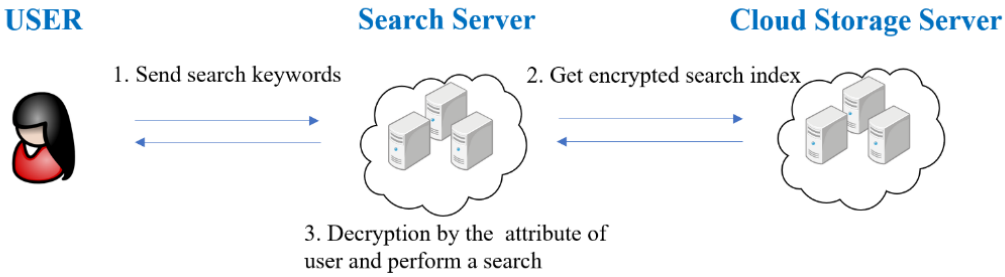

Figure 3 Secure Retrieval on Cloud Storage

3.2.2 Target System in IMSA

The target architecture described in ArchiMate is shown in Figure 4. Figure 4 shows application and technology architecture. In the technology layer, Cloud Storage Server stores Encrypted Index and Encrypted file data. The CP-ABE tool on Search Server has Encrypting and Decrypting functions. Network realizes Communication Protection.

In the application layer, Search Processing is composed of Attribute Checking, Decoding and Searching. Attribute Checking function checks the Attribute from the user, Decoding function decrypts Encrypted Index, then Searching function performs a retrieval. Attribute Checking, Decoding and Searching realize Checking Service, Decoding Service and Searching Service respectively. Checking Service, Decoding Service and Searching Service serve Secure Search. User Device, Search Server and Cloud Storage Server are connected by Network. Communication Protection which realized by Network encrypts communication. System software on server realizes Login Check and Identity Check.

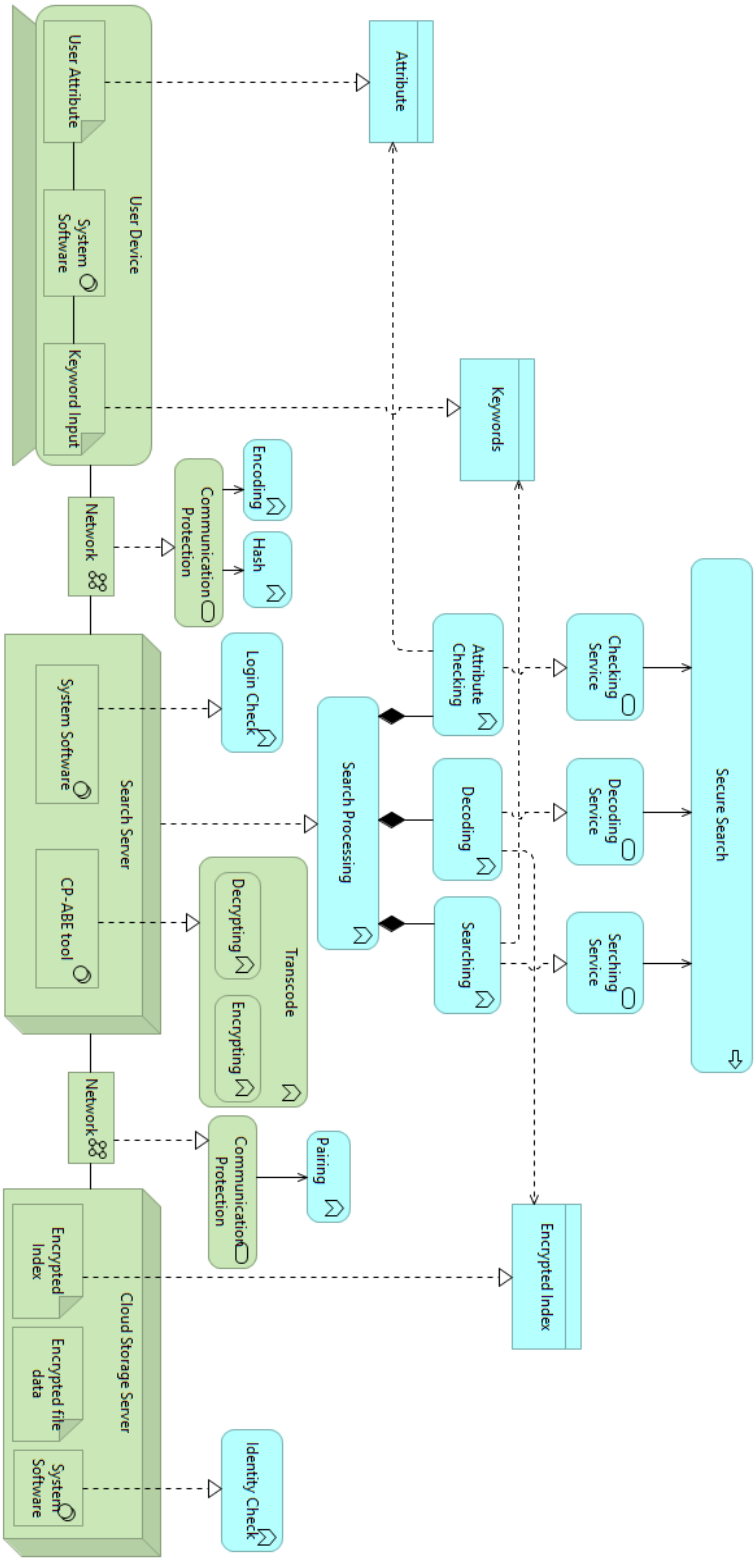

Figure 4 Secure Retrieval on Cloud Storage Architecture in ArchiMate

The security of the target system architecture is analyzed from the point of confidentiality includes data confidentiality, communication confidentiality and retrieval

confidentiality. The security case for the confidentiality on the target system architecture is shown in Figure 5. Figure 5 integrates the target system architecture and the corresponding security case in the same ArchiMate diagram.

Security issues include Storage Server Data Leakage, Index Data Tampering, exceeding authority, Search Server Data Leakage, Attribute Forgery, Keyword Disclosure and Keyword Tampering. Detect Illegal Users will be effective countermeasure for Server Data Leakage. The countermeasures for Index Data Tampering, Keyword Disclosure, Keyword Tampering and exceeding authority are to increase the difficulty of implementing these actions. The evidence for the countermeasures of Server Data Leakage is Login Authentication and User

Authentication, The evidence for the countermeasure of Index Data Tampering is Pairing, The evidence for the countermeasure of exceeding authority is Search Scope Judgement, the evidence for the countermeasure of Attribute Forgery is Digital Signature, the evidence for the countermeasures of Keyword Disclosure and Keyword Tampering are Communication Encryption and Hash Check, respectively. These requirements are realized by Identity Check function, Login Check function, Pairing function, Secure Search process, Attribute Checking function, Encoding function and Hush function, respectively.

3.2.3 Target System in D-case

The security case for the target system is also described by using D-case as follows. D-Case is a tool to describe GSN, which had been developed in the course of DEOS project. Figure 6 shows the security case of target system.

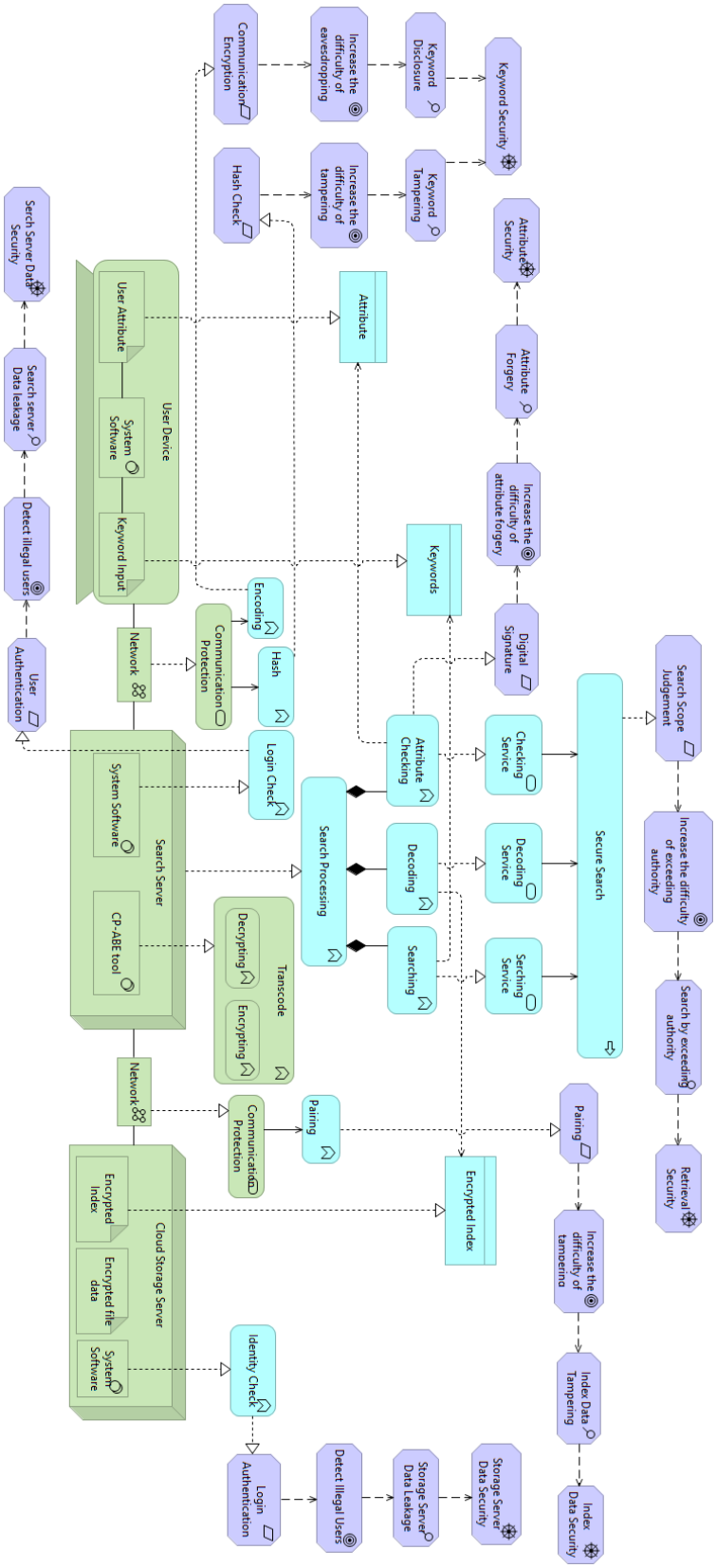

Figure 5 Intra Security Case Example for Secure Retrieval on Cloud Storage

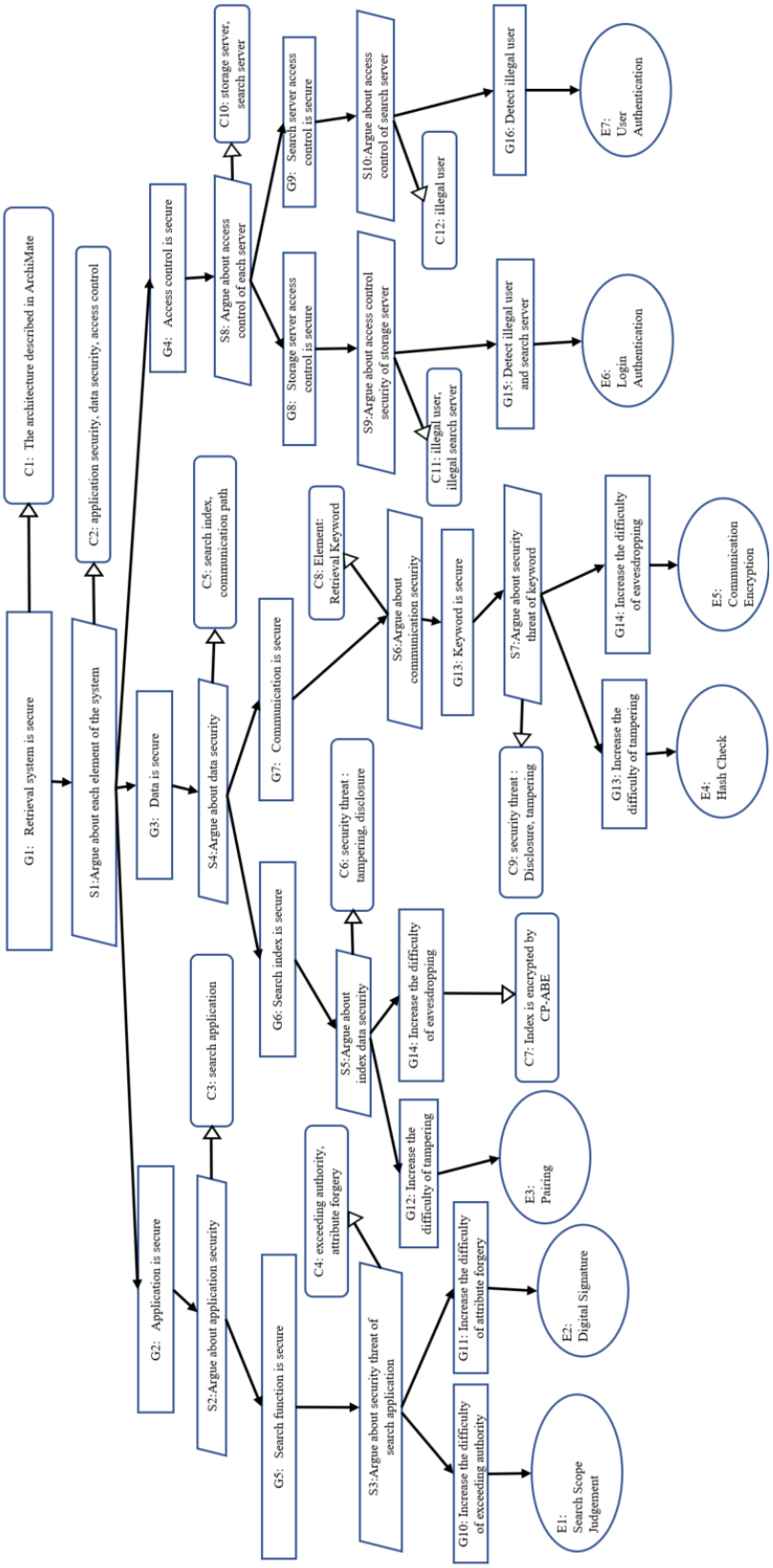

Figure 6 Security Case Using D-case

3.2.4 Comparison

Table 2 shows the number of nodes of the system. The number of architecture nodes for the target system is 30. The number of security case nodes for D-Case in the target system is 48. The number of security case nodes for ArchiMate in the target system is 27. The number of nodes for ArchiMate is smaller than those of D-Case and the reduction is about 44%. The reason is there is no need to describe context and decomposition nodes in ArchiMate. In D-Case, 12 context nodes are used to correspond with architecture elements and claims. Moreover, ten decomposition nodes are necessary to decompose claim nodes into sub claim nodes in D-Case.

Table 2 Number of Nodes of models and security cases

Meta model	Number of Nodes
Model in ArchiMate	30
Security case in D-Case	48
Security case in ArchiMate	27

Table 3 shows the number of relationships of the system. The number of relationships for the target system is 30. The number of relationships for D-case in the target system is 44. The number of security case relationships in ArchiMate for the target system is 28. The number of relationships for ArchiMate are smaller than those of D-Case and the reduction is about 36%. The reduction of relationships is smaller than the reduction of nodes and the reason is that there are 7 relationships between security case and system model.

Table 3 Number of Relationship of models and security cases

Meta Model	Number of Relationship
Model in ArchiMate	30
Security case in D-Case	44
Security case in ArchiMate	28
Between security case and model	7

3.3 Experiment

In order to evaluate the proposed method, we compare the method with the previous approach. We conducted experiments on two systems to improve the accuracy of the experimental results, and selected Healthcare device system and smart house system as

the experiment object. Table 4 shows the number of nodes of the systems. The number of architecture nodes for the Healthcare device and Smart house systems are 28 and 36, respectively. The numbers of security case nodes for D-Case in the Healthcare device and Smart house systems are 103 and 150, respectively. The number of security case nodes for ArchiMate in the Healthcare device and Smart house systems are 55 and 72, respectively. Six subjects are decomposed into two groups A and B. Each group contains three subjects. Subjects are all students who know D-case and ArchiMate. In the comparative experiment, six subjects are ordered to answer the four questions defined for each system in Table 5.

Table 4 Number of Nodes of models and security cases

Method	Healthcare device system	Smart house system
Model in ArchiMate	28	36
Security case in D-Case	103	150
Security case in ArchiMate	55	72

Table 5 Questions of experiments

Questions	Healthcare device system	Smart house system
What is the Threat for the asset?	Healthcare data	Operation information
What is the evidence for the countermeasure of the threat?	Threat for the pairing key	Threat for the smart device states
What element realizes the evidence?	Elements to realize above evidence	Elements to realize above evidence
What is the function does not realize the evidence?	Missing functions for the above evidence	Missing functions for the above evidence

Table 6 shows the combination of subject groups and target systems. Each group consists of three members.

Table 6 Combination of the experiment

Group	Healthcare device system	Smart house system
A	Previous	Proposed(IMSA)

B	Proposed(IMSA)	Previous

In the experiment, the average time to answer questions and the average ration of correct answers are collected. The results of the experiment are shown in Figure 7 and Figure 8. Figure 7 shows the comparison of average time to answer questions. The figure shows the average time to answer for the proposed approach is less than those of previous approach for two groups.

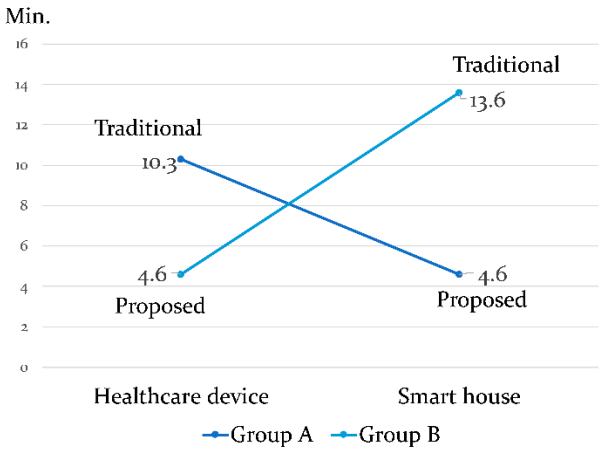

Figure 7 Comparison of average time to answer questions

Figure 8 shows the comparison of average correct ratios of questions. The figure shows the average correct answer ratio for the proposed approach is greater than those of previous approach for two groups.

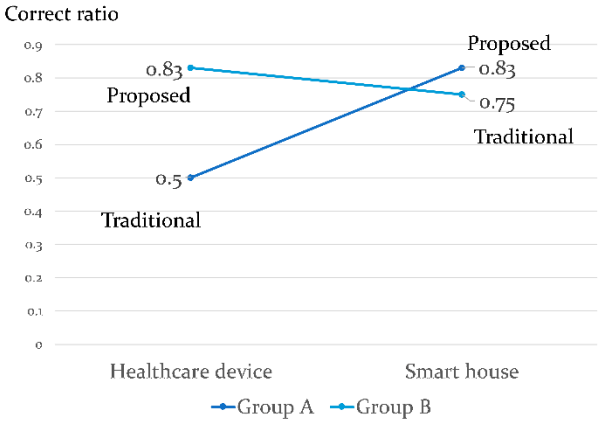

Figure 8 Comparison of average correct ratio of questions

3.4 Discussion

3.4.1 Comparison of the previous and proposed approaches

Table 7 summarizes the comparison between proposed and previous approaches to assure security of the architectures. The proposed approach can easily correspond with

security cases and architecture elements. Previous approach can not directly relate elements of assurance cases and architectures. In case of the previous approach, it is necessary to use specific security case editor other than architecture editors for developing and exploring security cases. The mapping rules between security case and architecture diagrams are necessary to develop security cases for the previous approach. The mapping rules of the previous approach should be collected as security case patterns for different architecture domains. Different from the previous approach, the proposed approach only needs to define the usage rule of motivation elements for security case as shown in Table 7.

Table 7 Comparison of approaches

Items	Previous approach	Proposed Approach
Artifacts	Architecture diagram and GSN	ArchiMate
Relations among elements	By using names indirectly	By relating nodes directly
Editing operations	By using 2 diagram editors	By using ArchiMate editor
Rules of diagramming	Mapping rules between 2 diagrams	Description rules of security case in ArchiMate

3.4.2 Effectiveness

The case study in Section 3.2 showed that the proposed approach could reduce the number of nodes and relationships by approximately 40%. The correctness ratio of the proposed method was higher than those of the previous method. The time to answer questions of the proposed method was also less than those of the previous method. These results showed the effectiveness of the proposed method. This result positively verified our hypothesis that using one diagram is superior to using two different diagrams for assuring security.

3.4.3 Limitation

The number of subjects in the experiment was small. The experiments using more number of subjects are necessary to generalize the result of the paper. The productivity and quality of the proposed method were not evaluated. It is also necessary to evaluate these fundamental properties for the proposed approach.

4 Visualized Assurance Approach for Enterprise Architecture

In d*framework, an actor node is used to relate the assurance case, the interaction parameters in system architecture can be represented by D (X, Y) in the d*framework. In detail, X and Y are related systems and D (X, Y) is the inter-dependability condition to assure that Y meets the dependability requirements of X. Such a relationship is termed an "open depend-on relationship." If an actor is undefined, the undefined actor is described by "*." The relationships in system architecture can be graphically represented in the d*framework, but the visual sense of the d*framework is inadequate. The premise of development using the d*framework is the existence of a collaboration diagram; therefore, the scope of the d*framework is limited when assuring dependability between software components. Moreover, in terms of visualization, the performance of previous methods is not outstanding. A newly unified and visualized method that describes not only the system architecture but also assurance cases is necessary.

This section proposed a composite dependability assurance approach to describe dependability arguments among actors including business actors, application actors, and technology actors. This is completed using ArchiMate, which is a modeling language for development of enterprise architecture models. ArchiMate provides a clear method for visualizing construction and business processes: operation, organizational structure, information flow, application service, and technology infrastructure.

4.1 Visualized Assurance Approach

In this section, a general structure of visualization for enterprise architecture is described in Figure 9, and Figure 10 shows a model that can integrate system architecture and security cases. The mapping between system architecture and actors in the business layer is also clarified in Figure 10 This model enables improvement of the efficiency of dependability assurance and of the visual sense. This is because the cognition and operation gaps caused by manipulating different diagrams, such as assurance case and system architecture diagrams, are reduced by using an integrated diagram and a unified modeling language. Compared to previous methods, this approach can also provide visual convenience for people in different fields. In the following, an example is provided to describe the internal behavioral relationship between actor and assurance case.

In this section, the meta-model for assurance case and business layer actors of the system architecture is proposed. In previous research [27] [28], the security assurance methods for the application and technology layers were only clarified; no proposal was made for the relationship between the actors of the business layer. For visualization of the dependability in the business layer, herein is presented an interdependence relation

method that is explained in next section using ArchiMate. Regarding the assurance cases in the application and technical layers, the previous research methods remain applicable.

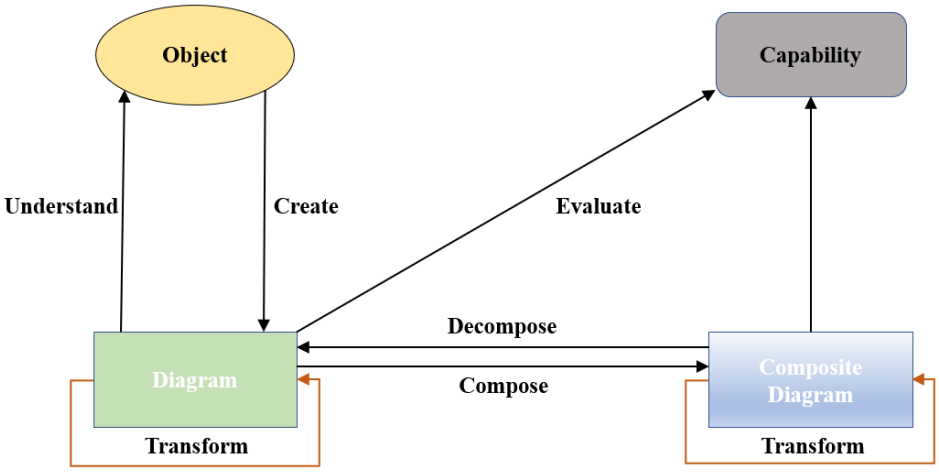

Figure 9 General structure of visualization for enterprise architecture

4.1.1 Model of the Relationship for the Business Layer and Assurance Case

Figure 10 shows the meta-model that integrates the system architecture and assurance case. The business layer of the system architecture is represented by orange in the diagram. In this meta-model, the business layer consists of two actors, which show an interdependency relationship. In the assurance case, there are four elements: property, risk, countermeasure, and evidence. The evidence will be realized by actor Y in the business layer; then, the countermeasure will be proved by the evidence. Next, the risk that threatens the property will be resolved by the countermeasure. Lastly, the property satisfies actor X in the business layer. A case study is used in Section 4.2 to analyze this model.

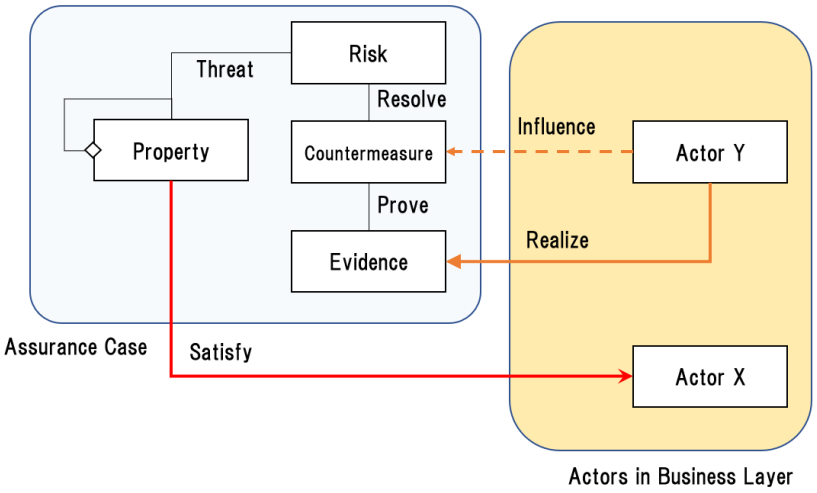

Figure 1 0 Model of relationship for business layer and assurance case

4.1.2 Using ArchiMate to Define the Mapping Relationship between the Assurance Case and Actor

Table 8 shows the mapping relationships for the model in Figure 10.

Table 8 Mapping between the meta-model and ArchiMate elements

Meta model	ArchiMate
Property	Driver
Risk	Assessment
Countermeasure	Goal
Evidence	Requirements
Actor	Business Actor

In the previous study [24], the internal relationship of assurance case in ArchiMate was proposed, this section proposed a method for combining the assurance case with the actors in business layer as shown in Figure 11. In this model, the interrelationship between business actor and assurance case is described by the influence, association and realization relationship. Actor Y realizes the requirement and influence the countermeasure, the property associates Actor X.

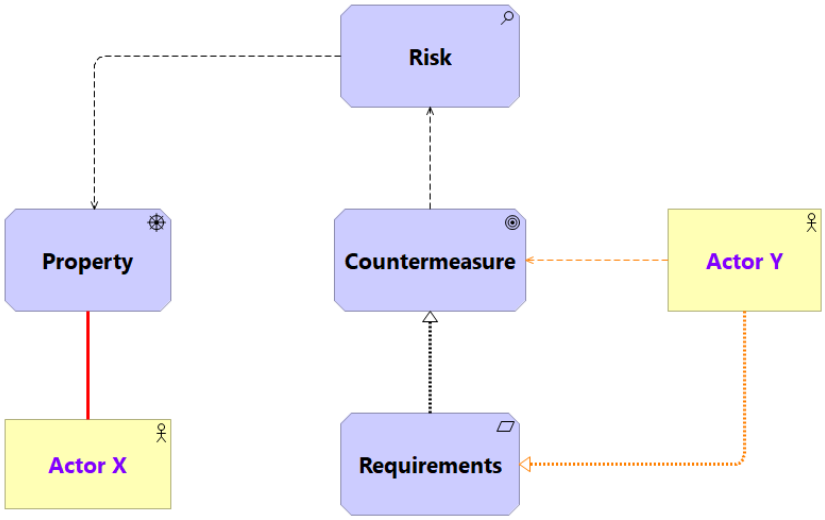

Figure 1 1 Model of relationship for business layer and assurance case in ArchiMate

4.1.3 Definition of the Mapping between the Business Actor and Composite Dependability Goals

To describe the dependability goals in the d*framework, a method for mapping the depend-on relationship between actors is proposed. For a goal, actor X depends on another actor, Y. The goal realized by actor Y will influence Actor X. In this study, the depend-on relationship is expressed by the association and realization relationships in ArchiMate. Figure 12 shows an example of the depend-on relationship. In this figure, a student is associated with the goal expressed as “The lecture is interesting,” and the teacher realizes this goal.

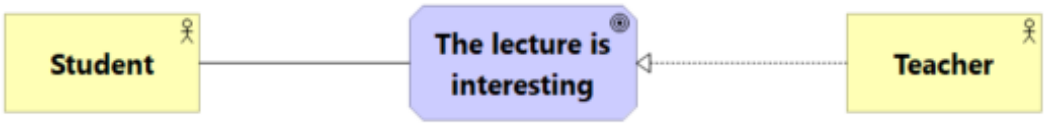

Figure 1 2 Example of a depend-on relationship

4.1.4 Combination of Depend-on Relationship and Assurance Case

To clarify the depend-on relationship between actors, the composite dependability goal will be decomposed into an assurance case such as that shown in Figure 11. In the case presented here, a student wants to have a dependable cram school (college preparation school) but is worried that the teaching level of the teacher is not sufficiently high. He needs a qualified teacher. In this case, that the cram school is dependable will be the

property; the risk is insufficient teaching level; the countermeasure is verifying the teacher's teaching level; and lastly, an adequate teacher qualification will be the evidence.

According to Table 1, property, risk, countermeasure, and evidence will be represented by the drivers, assessment, goal, and requirements in ArchiMate. Figure 13 shows this case in ArchiMate.

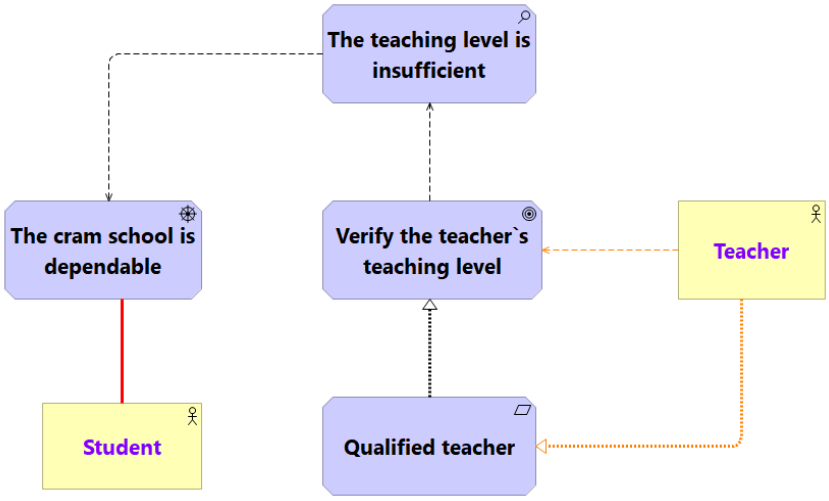

Figure 1 3 Using ArchiMate to describe the dependency relationship

4.2. CASE STUDY

4.2.1 Digital Signature Process

To explain this proposal, a case study is highlighted in the following. The target system is the digital signature process and the dependability among the sender, receiver, and key generator is analyzed. Although this process is typically more complicated, the system structure is simplified here to explain the approach of this section accurately. Figure 14 shows the architecture of the digital signature process using ArchiMate.

In this diagram, there are 3 actors, 6 business processes, 2 application elements, and 4 technology elements. Actors include sender, receiver, and key generator. These business processes include requests for key generation, signature generation, key pair generation, decryption signature, hash calculation, and comparison. In the application layer, there is a sender interface and a receiver interface. In the technology layer, there is a sender device, receiver device, and key generator device.

The digital signature process works as follows:

1. Sender requests key generation from key generator.
2. Key generator generates and sends key pair to sender.

3. Sender sends public key to receiver.
4. Sender encrypts hash value of data; sends to receiver.
5. Receiver decrypts encrypted data with public key; makes comparison.

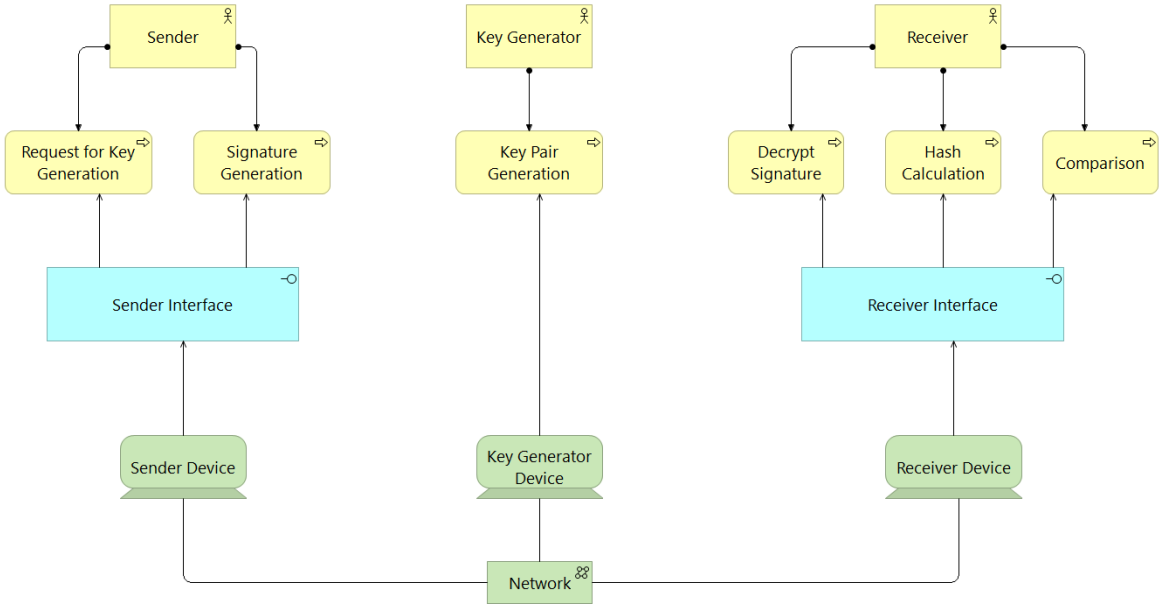

Figure 1 4 Digital signature process in ArchiMate.

4.2.2 Depend-on Relationship in a Digital Signature Process

For this digital signature process, all of the depend-on relationships among the actors are explained as follows:

1. The sender depends on “Generated key is reliable” for the key generator.
2. The receiver depends on “The signature is dependable” for the sender.

In Section 4.1, the model of a relationship for the business layer and assurance case in ArchiMate is introduced. Figure 15 describes the composite dependability by using the proposed approach based on Figure 14. Because the depend-on relationship between actors in the business layer is only highlighted, Figure 15 does not show the elements of Risk, Countermeasure, and Evidence for each assurance case. When the details of an assurance case are required, all elements should be treated similar to the example in Figure 15. This is done by using the method proposed in Section 4.1, and the elements in the application and technology layers are used as evidence because of the previous proposal [24].

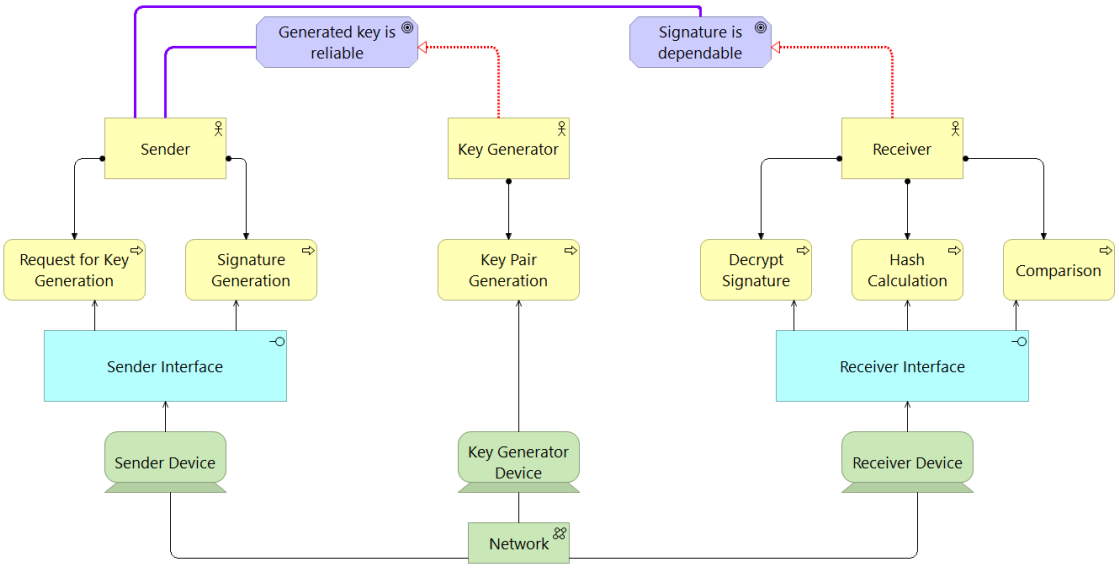

Figure 1 5 Depend-on relationships of the digital signature process in ArchiMate

4.2.3 Assurance Case in the Application and Technology Layers

Figure 15 only defines the depend-on relationships; if the corresponding evidence for dependability is necessary, according to the model in Section 4.1, the evidence for these depend-on relationships can be treated as follows. In Figure 16, taking “Signature is dependable” as an example, the realization relationship between Evidence and Goal is clarified.

In the receiver interface, there are 3 functions: signature decryption, hash calculation and hash comparison. If the result of the hash comparison is consistent, the goal of “Signature is dependable” can be achieved.

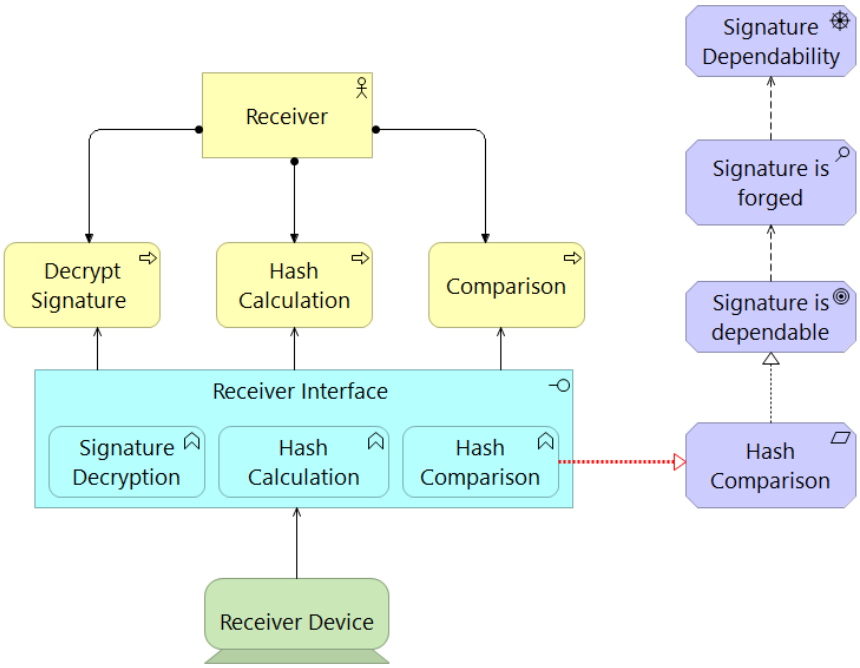

Figure 1 6 An assurance case for the digital signature process in ArchiMate

In a complex system architecture, there may be many actors. To rapidly clarify dependencies among complex relationships, it is necessary to highlight the depend-on relationships among actors in an acceptable manner. Figure 17 shows the depend-on relationships between business actors based on Figure 17. For the assurance case in Figure 16, the evidence for the assurance case is the hash comparison function, but the relationship between the hash comparison function and the receiver actor in the business layer is a serving relation; therefore, it can be said that the dependability of a signature is realized by the receiver actor, as shown in Figure 17.

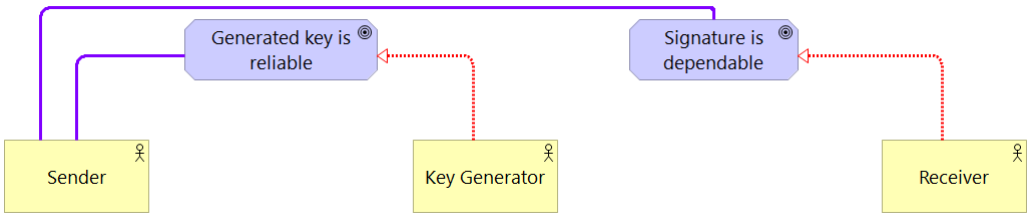

Figure 1 7 Depend-on relationships between business actors

4.3. Discussion

The method for description of the interdependency relationship between business

actors was previously illustrated. The d*framework can also be used to assure an interdependency relationship. In the following section, the d*framework and the proposed approach are compared in detail, and the effectiveness and limitations of the new proposal are discussed.

4.3.1 Comparison of ArchiMate and d*framework

In Table 9, the comparisons between the proposed method and the d*framework include actor, dependability claim, relationship, and system configuration. For the actor, only the module node can be used in the d*framework, but in ArchiMate, all of the following (Business Actor, Component, Interface, Device, and Artifact), can be used as an actor. For example, the hash comparison function is regarded as an actor in Figure 16 and the business actor is regarded as an actor in Figure 17.

Table 9 Comparison between ArchiMate and d*framework

Objects	ArchiMate	d*framework
Actor	Business Actor, Component, Interface, Device, Artifact	Module
Dependability Claim	Requirement, Goal, Assessment, Driver	Goal
Relationship	Realize, Association, Influence, Serving	Depend on
System Configuration	Enterprise Architecture	Collaboration diagram

For the dependability claim, the goal represents the dependability claim in d*framework. However, in ArchiMate, the dependability claim can be represented as a requirement, goal, assessment, or driver, as shown in Figure 11.

For the relationship, d*framework can only use the depend-on relationship to represent the relationship between actors. However, in ArchiMate, the relationships of Realize, Association, Influence, and Serving can be used to represent relationships between actors in a more delicate way.

For the system configuration (because the d*framework does not have the capability to describe system architecture), collaboration diagrams are indispensable. However, because ArchiMate can be used for modeling system architecture, system architecture and dependability relationships can be well visualized.

Here, an additional example of a medical system is provided for comparing the number of architecture nodes and the number of relationships, and then explained in the

ArchiMate and d*framework, respectively.

Figure 18 shows the assurance case and the corresponding system architecture in the proposed approach. This a healthcare system for blood pressure measurement. The user can measure blood pressure by using this device and the measured data can be sent to a cloud server via mobile phone. The data are received and stored by the cloud server. Here, only data confidentiality is considered for the assurance case of the target system architecture.

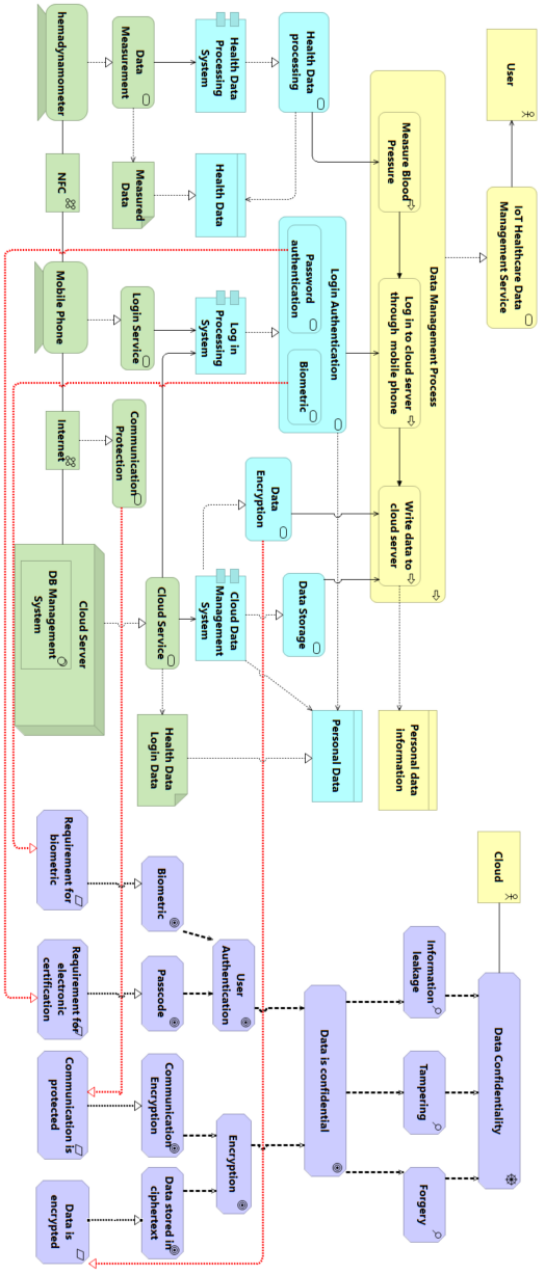

Figure 1 8 Assurance case of a healthcare system in ArchiMate

In Figure. 18, the upper part is system architecture, and the lower part (purple) is the

assurance case for the system architecture. It shows that the proposed approach can directly assure the system architecture.

Figure 19 shows the framework, in which the definition of the d*framework is as follows.

・Actor is the node type for a system. Actor can have goals, strategies, evidence, and contexts.

・Goal is achieved by Actor. It is represented by node type, and it can be decomposed into sub-goals or sub-strategies.

・A strategy clarifies how to achieve Goal. A strategy can also be decomposed into sub-goals and sub-strategies.

・Evidence is the node type supporting Goal, such as test reports, specifications, and procedure manuals.

・Context is the external information regarding Goals and Strategies.

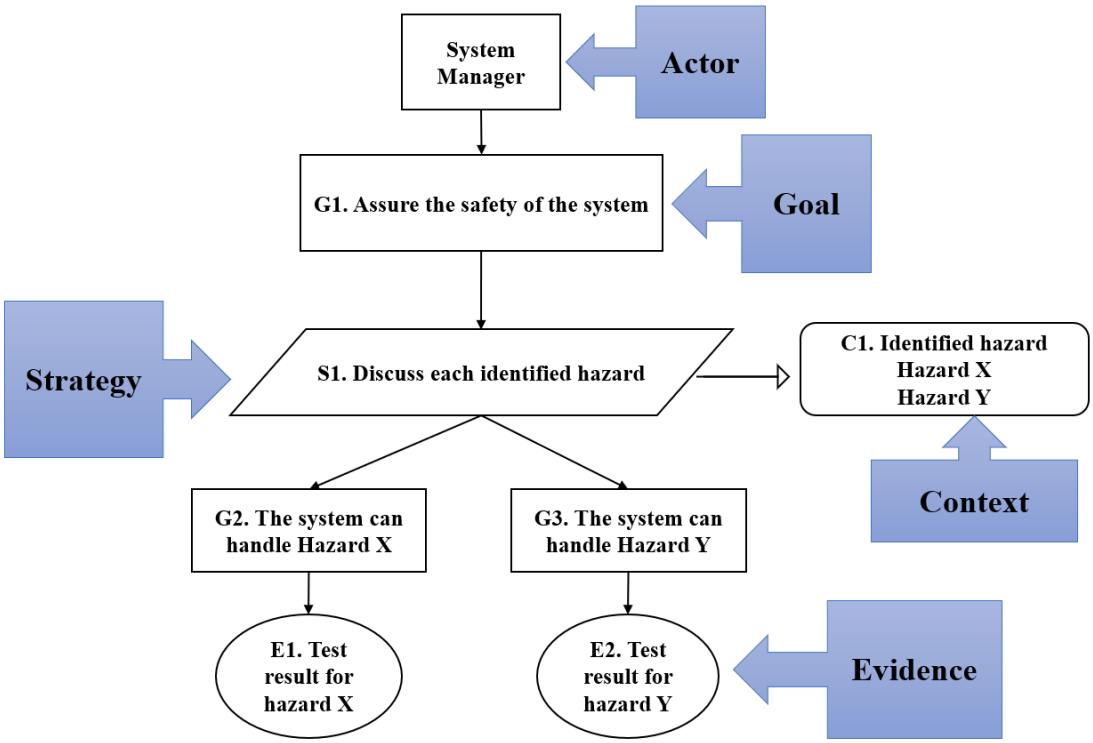

Figure 1 9 Outline of d*framework

Figure 20 shows an example of an assurance case in the d*framework. D-Case Editor [29] was developed as a tool for creating D-Case in the DEOS project, and the example shown below was created by the D-Case Editor.

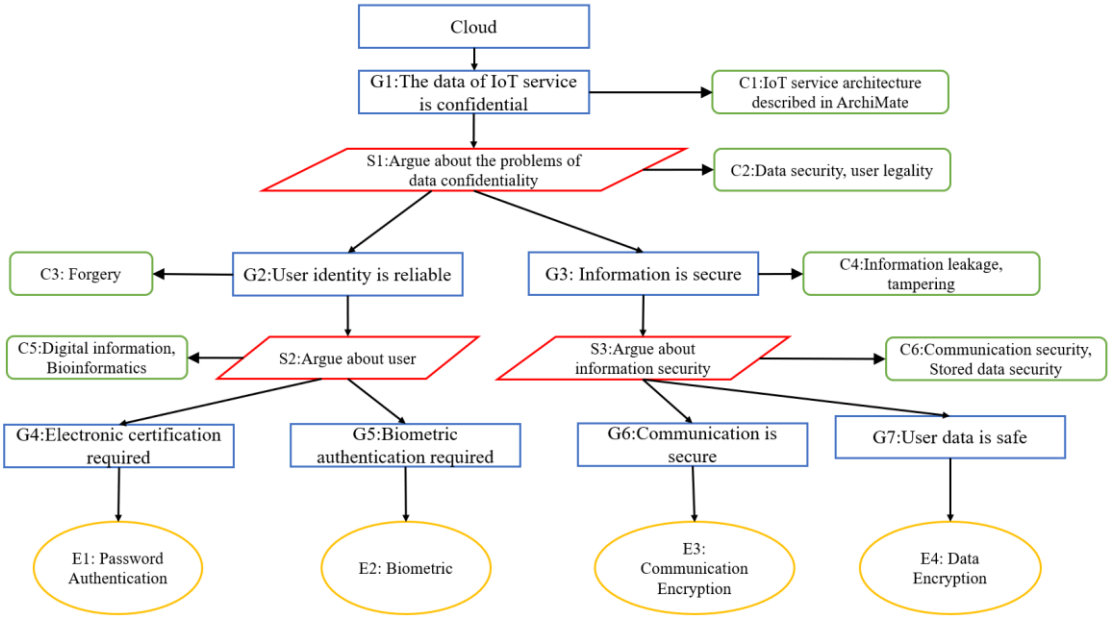

Figure 2 0 Assurance case of a healthcare system in the d*framework

The results are summarized in Table 3. For this case, the assurance case nodes number in the proposed method is 16 and the number of nodes in the d*framework is 21. The number of assurance case relationships in the proposed method is 17, while it is 20 in d*framework.

In summary, for the description in Actor, Dependability Claim, Relationship, and System Configuration, ArchiMate is superior to d*framework, and ArchiMate is more suitable in terms of assuring composite dependability. However, the number of relationships and nodes in ArchiMate is less than that of the D-case according to the example in the appendix. This result shows that, to some extent, the proposed method is more concise than the d*framework.

Table 10 Number of nodes and relationships in ArchiMate and d*framework

	Number of Nodes	**Number of relationships**
ArchiMate	16	17
d*framework	21	20

4.3.2 Effectiveness

As previously mentioned, the ArchiMate diagram is effective for dependability assurance. The effectiveness is summarized as follows.

In Section 4.1, assurance case methods based on system architecture diagrams were

clarified. When compared to the d*framework, ArchiMate is more effective for element diversification. The assurance case can be concretized by relating the elements in the application or technology layer. The assurance case can also be simply abstracted to represent the relationship between the actors in the business layer.

Because actors can be directly defined with system architecture in ArchiMate, the relationships between actors and the relationships between actors and system components are further clarified. In addition, the arguments of the assurance case can be easily defined using the motivational elements and the relations in ArchiMate.

Finally, from the visualization perspective, using ArchiMate to describe dependability relationships has advantages over the previous methods.

4.3.3 Limitations

Although the digital signature process was well evaluated in this study, it is necessary to evaluate some other systems to confirm the effectiveness of this proposal. Moreover, because the dependability assurance in the business layer is a depend-on relationship and is different from the description of the application or technology layers, it may be necessary to separately evaluate the method for the business layer.

5 Composite Safety Assurance for Healthcare Devices

Some classes of system are critical systems where system failure may result in injury to people, damage to the environment, or extensive economic losses. Examples of critical systems include embedded systems in medical devices, such as an insulin pump. As system failure may lead to user injury, the development of medical device often requires safety assurance.

Interdependency should be clarified for managing healthcare devices. It is necessary to describe interdependence of system actors for clarifying the safety assurance. That is to say, the interdependency of the system components, and the internal dependability of the system components should be proved to assure the dependability of a system. Although Yu [7] showed that the network of intentions among the actors could be represented using the i* framework, the problem of how to treat the dependability of systems has not been solved.

Some other methods were proposed to achieve dependability assurance. The purpose of developing safety case is to ensure the safety of a system. As previously mentioned, the Goal Structuring Notation (GSN) was proposed and widely used to develop assurance and safety cases. The argument patterns had been proposed to help engineers develop assurance cases. Besides, a security argument pattern and security case based on common criteria [15] has been proposed for assuring security. In the absence of any clearly organized guidelines concerning the approach to be taken in decomposing claims using strategies and the decomposition sequence, engineers often do not know how to develop their arguments. For this, assurance cases were summarized and prospected by Bloomfield and Bishop [30]. Besides, d* framework, which is a derivative of GSN, is used to assure system dependability. In d*framework, an actor node is used to relate the assurance case. The premise of development using the d*framework is the existence of a collaboration diagram. Therefore, the scope of the d*framework is limited when assuring dependability between system components. Moreover, in terms of visualization, the performance of d* framework is not outstanding.

Goal-oriented approaches [31] are applied to analyze healthcare processes [32] [33]. Enterprise Architecture (EA) [17] can be used to model medical systems. For example, Eldein [34] discussed EA for the cloud service of mobile healthcare. Ahsan [35] designed and provided the insight of an EA approach to process architecture for healthcare management. Yamamoto proposed an ArchiMate pattern to analyze e-health business model [36]. Zhi proposed a method to assure the dependability between business actors [37], but the assurance between system components is not clarified.

5.1 Composite safety assurance

In this section, the metamodel for composite dependability is proposed. We have proposed an intra model security assurance (IMSA) [27] method to describe security assurance, but the dependability relationships between system components cannot be treated, for the interdependence between system components, we propose the metamodel as shown in Figure 21, which shows the metamodel of the composite goal concept. The depender component depends on the composite goal. The dependee component achieves the composite requirements that realize the composite goal and countermeasure through composite requirements.

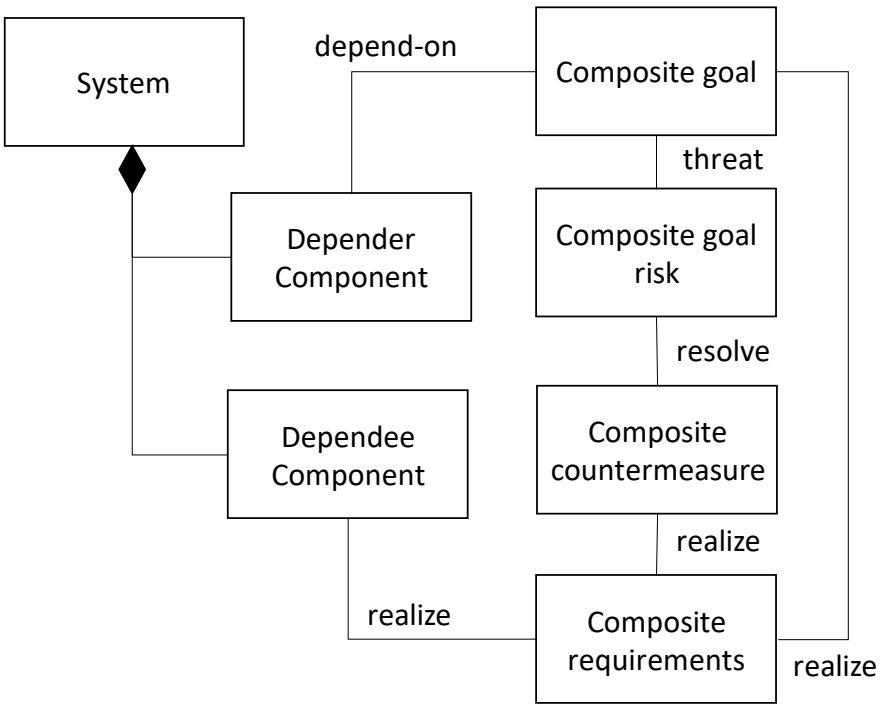

Figure 2 1 Metamodel of composite goal

Next, we use ArchiMate to describe the composite goal metamodel. Table 11 shows the mapping relationships between the metamodel and ArchiMate elements.

Table 11 The mapping between composite goal metamodel and ArchiMate elements

Composite goal meta model	ArchiMate elements
Stakeholder,	Actor,
System, Component	Application component, Node
Composite Goal	Driver

Composite goal risk	Assessment
Composite countermeasure	Goal
Composite requirements	Requirements

Figure 22 shows the definition of safety case in ArchiMate. The interrelationship between elements is described by the influence and realization relationship in ArchiMate. The realization relationship is used between countermeasures and requirements. The influence relationship between safety goal, risk, and countermeasure is negative.

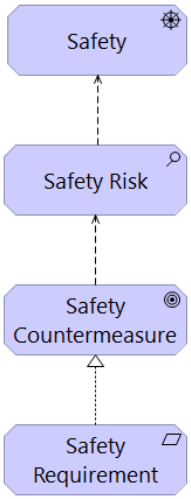

Figure 2 2 Example of Safety Case in ArchiMate

To describe the dependability goals in d*framework, a method for mapping the depend-on relationship between actors has been proposed. Suppose that an actor X depends on another actor Y for the safety goal. The safety goal which realized by the actor Y will support the safety of the Actor X. In this section, the depend-on relationship is defined by the association and realization relationships in ArchiMate. Figure. 23 shows an example of the depend-on relationship using ArchiMate. In this figure, a patient is associated with the safety goal expressed as “Blood sugar is balanced,” and the insulin pump realizes this goal.

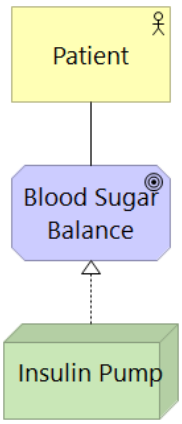

Figure 2 3 Example of composite safety relationship

According to the above approach, the composite safety between patient and insulin pump is defined in ArchiMate as follows in Figure 24.

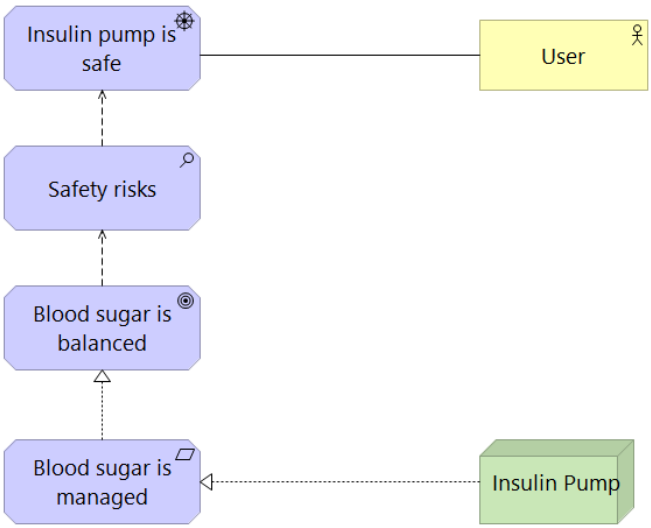

Figure 2 4 Using ArchiMate to describe Composite safety between Patient and Insulin pump

5.2 Composite safety assurance method

We introduce the steps of the composite safety assurance method in this section.

[STEP1] Describe system architecture with ArchiMate

System architecture is a generic discipline to handle systems, and it is the study of early

decision making in complex systems [38]. Systems modeling language (SysML) and Unified Modeling Language (UML) are applied to model system architecture. However, these modeling languages only focus on modeling software and system architecture. In order to achieve composite safety assurance, we use ArchiMate to describe system architecture and assurance case.

[STEP2] Identify composite safety goals between components

Complexity theory implies that system components are interdependent to the extent where changes in one component may affect another, or result in failure of interconnected systems [39]. Identifying, understanding, and analyzing critical architecture interdependencies are essential [40]. Thus, it is necessary to identify the interdependency between system components. But what safety goals should be set between system components is not within the scope of this section. We have introduced how to use ArchiMate to define the composite safety assurance relationships, and will further clarify this method based on a case study of insulin pump in the next section.

[STEP3] Safety goals elicitation

In order to ensure the safety of a critical system, safety goals should be extracted after confirming the relationships between system components. For goals elicitation, the risks of a system should be grasped. Safety goals can be derived from the corresponding risks. Figure 4-3 showed an example of safety goal, which is “Blood sugar balance”.

[STEP4] Requirements elicitation for safety goals

In safety assurance, the requirements are necessary for the realization of a goal. Here, the requirement intended to finally support an elaborated goal, such as the verification results of tests or techniques. Namely, the requirements should be met by a software system in order for that system safe and stable [41]. In Figure 4-4, the requirement for the safety goal is “Blood sugar is managed”.

[STEP5] Safety goals assurance using composite requirements

A safety goal should be realized by requirements. In general, each requirement should have corresponding evidence. In Figure 4, the corresponding evidence for the requirement is “Insulin pump”, which is a system actor in the system architecture.

5.3 Case study of the proposed method

In this section, to illustrate the proposed approach, we use the 5-steps method described

in the previous section to analyze an insulin pump, which is a medical device. The insulin pump in this case study is for personal use. In recent years, insulin pump is gradually accepted, and its safety has also widely received attention [42] [43]. Zhang analyzed the hazards for the insulin pump [44].

[STEP1] Describe insulin pump system architecture with ArchiMate

We modeled the insulin pump system architecture using ArchiMate as shown in Figure 25. It is an ArchiMate model to illustrate how the insulin pump software transforms an input blood sugar level to a sequence of commands that drive the insulin pump. This is an embedded system, which collects the information from a sensor and controls a pump that delivers a controlled dose of insulin to a patient. In this section, we discussed only the software related safety issues. The safety issues related to hardware and environment, such as battery and extreme environment will not be discussed.

In the figure, the patient interacts with the insulin pump through the user interface. The patient can receive the information from the output device and input commands through the input device using the user interface.

The insulin is administered to the patient by the insulin pump via a delivery path, which composed of the insulin reservoir, the insulin delivery mechanism, and the blood sensor. The insulin reservoir and the insulin delivery mechanism are monitored and administered by the insulin pump actuator and controller. The pump delivery mechanism can make insulin delivered from the pump to the patient at a prescribed time or rate. The insulin pump control component includes blood calculation function, insulin dose computation function, compute pump command function and exception handling function. For the exception handling, if the software fails, the safe dose of insulin will be set and insulin pump will alerts.

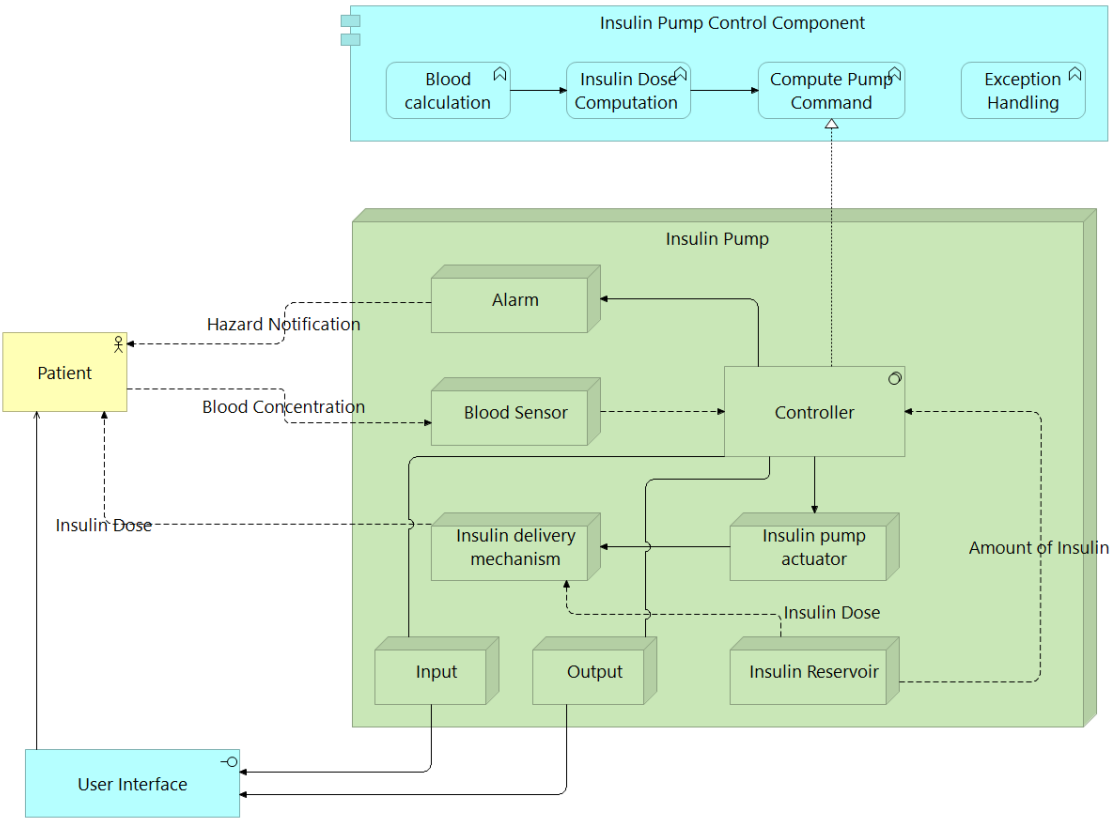

Figure 2 5 Insulin pump system in ArchiMate

A software-controlled insulin delivery system might work by using a microsensor embedded in the patient to measure some blood parameter that is proportional to the sugar level. Then the blood parameter will be sent to the pump controller. This controller computes the sugar level and the amount of insulin that is needed. At last, it sends signals to a miniaturized pump to deliver the insulin via a permanently attached needle.

[STEP2] Identify composite safety goals between insulin pump system components

Next, we would like to explain the safety issues about the insulin pump. Obviously, the insulin pump system is a safety-critical system. Safety assurance is necessary for the development process. If the pump fails to operate or does not operate correctly, then the patient's health may be damaged or they may fall into a coma because their blood sugar levels are too high or too low. Therefore, the system must meet two essential requirements as follows.

1. The system should provide insulin when insulin is required.

2. The system should perform reliably and deliver the correct amount of insulin to offset current blood glucose levels.

Here, we analyze the insulin pump working process as follows.

1. “Blood sensor” measures blood sugar level.
2. “Blood calculation function” analyzes the result of blood sugar level.
3. “Insulin dose computation function” calculates the insulin dose.
4. “Compute pump command function” conducts based on calculation results.
5. “Insulin delivery mechanism” delivers insulin through infusion set according to the command.

[STEP3] Safety goals elicitation in insulin pump architecture

For this architecture, we analyze the 5 depend-on relationships among these actors as follows.

1. “Blood sensor” depends on “The sensor is dependable” for “Blood calculation function”.
2. “Blood calculation function” depends on “The blood calculation is dependable” for “Insulin dose computation function”.
3. “Insulin dose computation function” depends on “The insulin computation is dependable” for “Compute pump command function”.
4. “Compute pump command function” depends on “The pump command is dependable” for “Insulin pump actuator”.
5. “Insulin delivery mechanism” depends on “The insulin delivery is dependable” for “Patient”.

We add these depend-on relationships as safety goals into the system architecture by using the method previously mentioned. Figure 26 shows the depend-on relationship on the insulin pump system in ArchiMate.

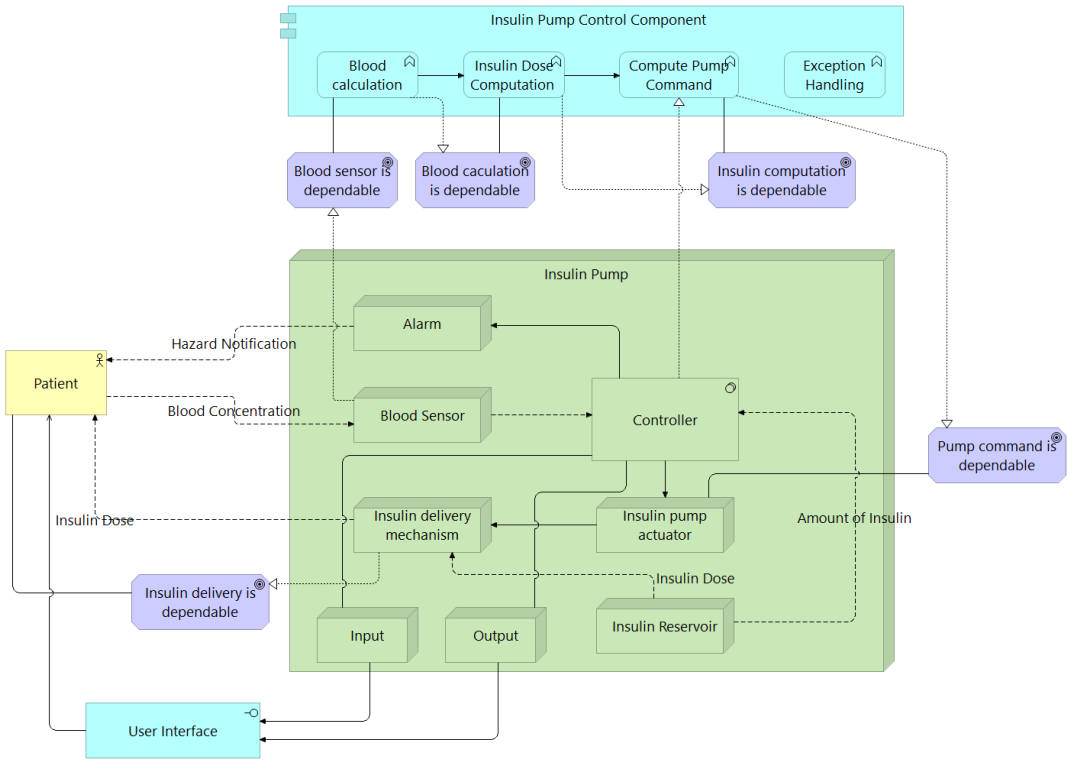

Figure 2 6 Depend-on relationship on the insulin pump system in ArchiMate

[STEP4] Requirements elicitation for insulin pump safety goals

As previously mentioned, the corresponding requirements are required for safety goals. For this, there should be requirements correspond to the depend-on relationships mentioned above. We analyze these requirements as follows.

1. For “Blood sensor”, the value range of blood sugar should be defined. If the measured blood sugar level is outside this range, it should stop working or deliver safe insulin dose, and alerts at the same time.
2. For “Blood calculation function”, the analysis algorithm should be defined. If the data is abnormal, it should stop working or deliver safe insulin dose, and alerts at the same time.
3. For “Insulin dose computation function”, the method of calculating insulin dose based on the blood sugar level should be defined, if the computation result is abnormal, it should stop working or deliver safe insulin dose, and alerts at the same time.
4. For “Compute pump command function”, the pump command and exception handling should be defined.
5. For “Insulin pump delivery mechanism”, the control of insulin dose should be

defined. If the insulin dose is abnormal, it should stop working or deliver safe insulin dose, and alerts at the same time.

[STEP5] Insulin pump safety goals assurance using composite requirements

We implement the requirements into the insulin pump system architecture as shown in Figure 27.

1. For safety goal “The sensor is dependable”, the corresponding requirements are “Exception handling” , “Alert” and “Measurement rule is defined”. The evidences that support the requirements are “Exception handling function”, “Alarm device” and “Blood sensor”.
2. For safety goal “The blood calculation is dependable”, the corresponding requirements are “Exception handling”, “ Alert” and “The calculation rule is defined”. The evidences that support the requirements are “Exception handling function”, “Alarm device” and “Blood calculation function”.
3. For safety goal “Insulin computation is dependable”, the corresponding requirements are “Exception handling”, “Alert” and “Computation rule is defined”. The evidences that support the requirements are “Exception handling function”, “Alarm device” and “Insulin dose computation function”.
4. For safety goal “Pump command is dependable”, the corresponding requirements are “Exception handling”, “Alert” and “Pump command is defined”. The evidences that support the requirements are “Exception handling function”, “Alarm device” and “Compute pump commands function”.
5. For safety goal “Insulin delivery is dependable”, the corresponding requirements are “Exception handling”, “Alert” and “Insulin dose is defined”. The evidences that support the requirements are “Exception handling function”, “Alarm” and “Insulin delivery mechanism”.

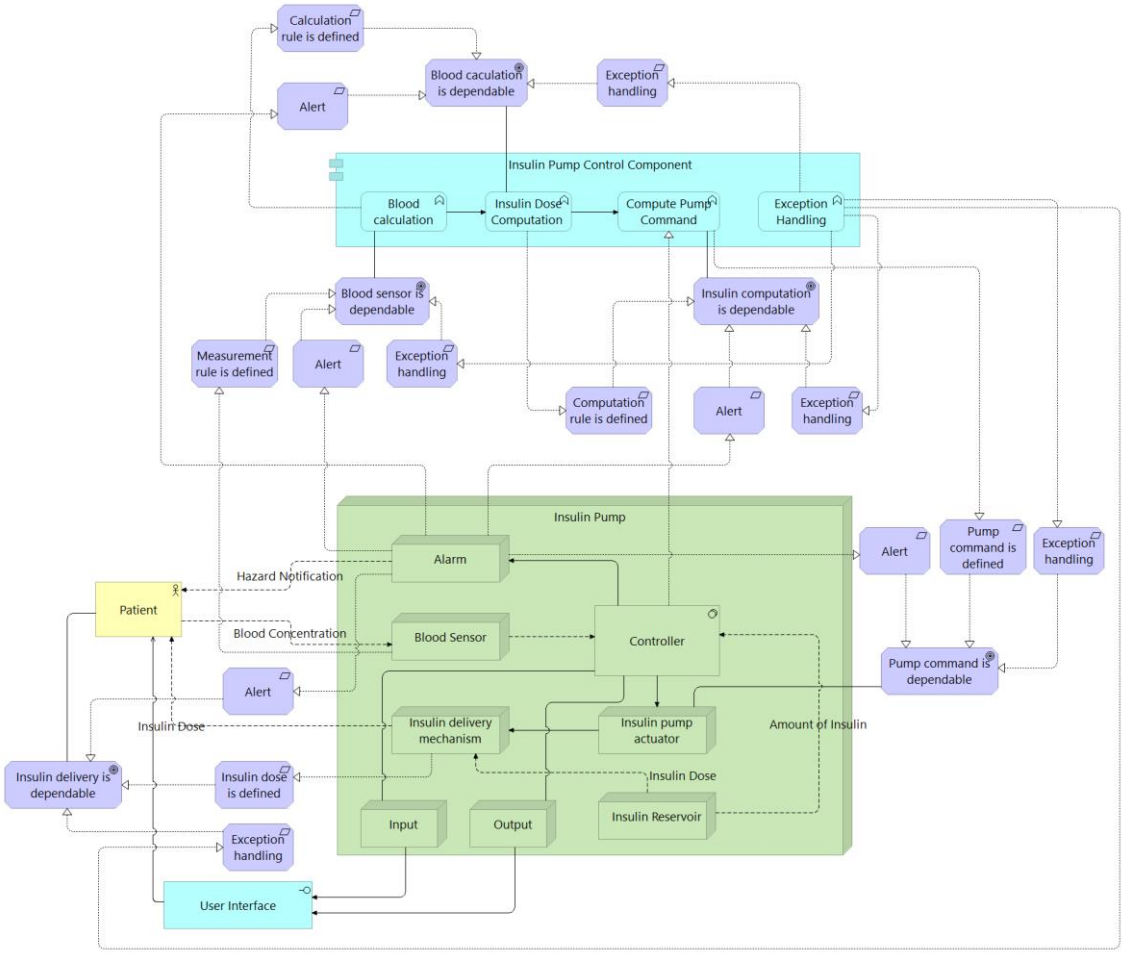

Figure 2 7 Composite safety on the insulin pump system in ArchiMate

According to the above steps, we have achieved safety assurance for the insulin pump system architecture.

5.4 Discussion

In previous sections, we proposed an approach to develop composite safety assurance through ArchiMate. Moreover, a case study of insulin pump safety was carried out to illustrate this approach. To verify the effectiveness of the proposed approach, we compare it with d*framework, which is a method to assure the composite dependability.

Table 12 compares the proposed method to the d*framework at system components, safety claim, and relationship. For the system components, d*framework only uses the module node or actor. In the proposed method, system components were represented by the nodes of business architecture layer, application architecture layer and technology architecture layer in ArchiMate as shown in Figure 4-4, Figure 4-5, Figure 4-6 and Figure 4-7, such as Component Business actor, Interface, Function, Device, and System software.

System components can be more vividly described in ArchiMate. Besides, in the proposed method, we defined the safety assurance rules by using the nodes of Driver, Assessment, Goal, and Requirement in ArchiMate. In d* framework, safety claim consists of Context, Strategy and Evidence [45]. Moreover, in the proposed method, we defined components relationships by using Realize, Association, Influence, and Serving relationships in ArchiMate. However, the relationship between components in d* framework is the depend-on relationship. The proposed method can more clearly describe the relationship between system components.

Table 12 Comparison of Proposed method and d*framework

Items	ArchiMate	d*framework
System components	Component	Module Node
	Business actor	Actor
	Interface	-
	Function	-
	Device	-
	System software	-
Composite Safety Claim	Driver	Goal
	Assessment	Context
	Goal	Strategy
	Requirement	Evidence
Relationship	Realize	-
	Association	-
	Influence	Depend-on
	Serving	-

In summary, the proposed method is superior to the d* framework in describing the system components and the relationships. For the safety assurance of healthcare systems of medical devices, the proposed approach is effectively applicable.

5.5 Effectiveness

As previously mentioned, the ArchiMate diagram is effective in safety assurance for safety-critical systems. The effectiveness is summarized as follows.

Because system components can be directly defined with system architecture in ArchiMate, the relationships between system components are further clarified. In addition, the arguments of the assurance case can be easily defined using the motivational elements and the relations in ArchiMate as previously mentioned. Besides, from the visualization perspective, using ArchiMate to describe dependability relationships has advantages over the previous method.

5.6 Limitations

In this section, only one case study of the insulin pump was carried out, and only partial safety issues of software were analyzed. Besides, we did not consider the quantitative comparison with the previous approach. The comparative experiment to quantitatively evaluate the productivity and quality should be carried out to verify the effectiveness.

6 Composite Safety Assurance Using Model Checking

6.1 Definition of the 4-Steps of Composite Safety Assurance

To improve confidence in system assurance, a composite safety assurance approach is proposed in this chapter based on the 4-steps as follows.

1. Interaction visualization.
2. Processes model checking.
3. Dependability-case creation.
4. Composite safety assurance.

This section will define each step in detail.

6.1.1 Visualization of the Component Interaction

Traditional system architectures with state transitions are typically developed in UML, and a limited state transition process can be described. When safety assurance is required for the system architecture, it is often necessary to develop an additional assurance case diagram, such as GSN diagram. However, it cannot intuitively express the inter-dependency relationships between system components, and the management of system diagrams and assurance case diagrams becomes complicated. Intra-model security assurance (IMSA) [27] approach has been proposed for integrating system architecture and assurance case, this method is based on ArchiMate, because ArchiMate can clearly describe the interactions and state transitions between system processes. However, this method does not take into account the rationality of the interaction and state transfer between system processes. Figure 28 shows the metamodel of composite safety assurance.

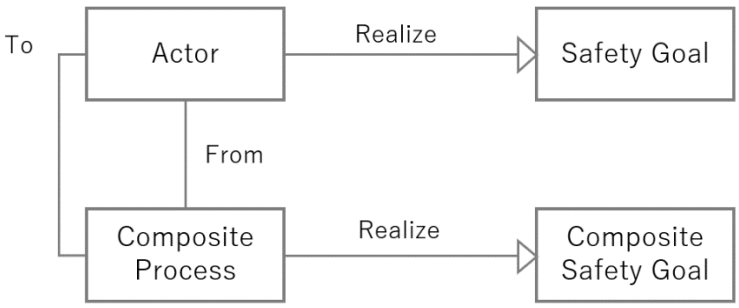

Figure 2 8 The metamodel of composite safety assurance

Here, an example of automatic driving system (ADS) is carried out to illustrate the relationships among actor, process, and safety goal as shown in Figure 29.

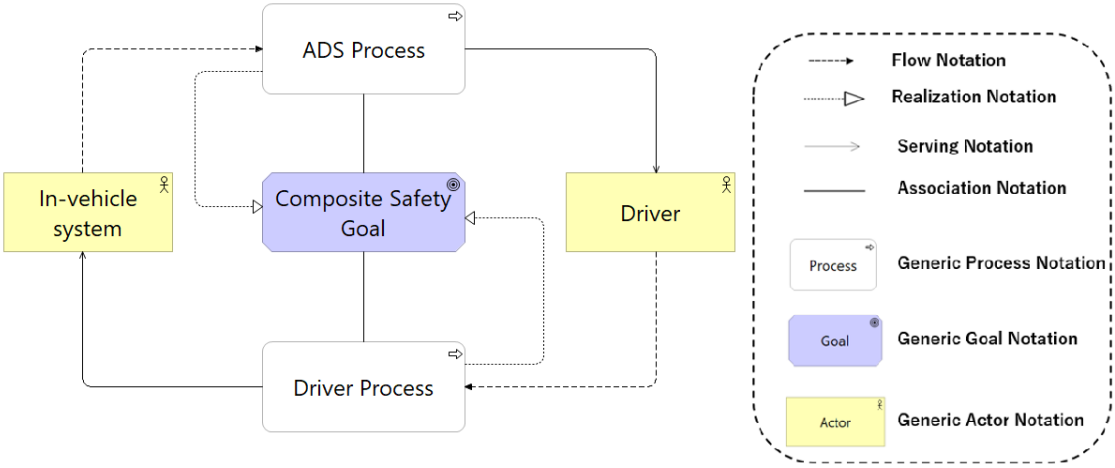

Figure 2 9 The relationships in ADS

This figure is developed with ArchiMate, it consists of process, actor, and safety goal. The ADS process and the Driver Process are composite process, there should be complex interactions both inside of the ADS Process and the Driver Process. Besides, there are also complex interactions between ADS Process and Driver Process. We aim to assure the dependability of the composite process in this paper. The relationship between processes is trigger relation or flow relation. It is noteworthy that in this paper, the flow relation defined as the corresponding channel in CSP model, and the trigger relation defined as the status transition in CSP model. The relationship between the safety goal and the process is association relation. The relationship between the process and the actor is serving relation or flow relation.

6.1.2 Processes Model Checking

Model checking is a method for algorithmic verification of formal systems. It is used to verify that models derived from hardware and software designs meet formal specifications as specifications are often described in the form of logical expressions of temporal logic. There are types of model checking tools, and their functionalities are different from each other. In this section, PAT [50] is used for processes model checking to check the processes for deadlocks. PAT is a self-contained framework for composing, simulating and reasoning of concurrent, real-time systems and other possible domains. The modeling language is Communicating Sequential Processes (CSP) [51]. In this chapter, the purpose of process model checking is to detect if there are logical errors between system processes, then the result of model checking is regarded as the evidence in step 3

Here, the purpose of process model checking is to detect if there are logical errors between system processes. The CSP syntax defines a “legitimate” way of combining

processes and events. Let e be an event and X be a set of events. Then the basic syntax of CSP in PAT is defined as follows.

Definition:

P* :: = *Stop
|***Skip***
|***e{prog}→P***
|***P1;P2***
|***P1* [] *P2***
|***P1*|||*P2***
|***P1*‖*P2***
|***[b]P***
|***atomic{P}***
|***P1* △ *P2***
|***ch!exp→P***
|***ch?xP***
|***case{b1:P1;b2:P2;⋯;default:P}***
|***e* ::= *name(.exp)****

Here, P1 and P2 are process, e is the name of an event with an optional sequential program prog. b, b1, and b2 are boolean expressions. Stop represents the deadlock process, and Skip represents the normal termination of the process. e → P represents a process that behaves as process P after event e occurs. If e has a sequential program, it behaves as process P after executing the program. P1; P2 represents a process that behaves as process P1 and then behaves as P2. P1 [] P2 represents a process with non-deterministic choice, and its behavior is determined by the event as either process P1 or P2. P1|| P2 is a parallel synthesis process that synchronizes with events that appear in both processes. P1 ||| P2 represents the interleaving synthesis of concurrent processes. [b] P represents a guarded process and behaves as P when the Boolean expression b holds. P1 △ P2 is a process that represents an interrupt. It behaves as P1 until the first event of P2 occurs, and after that event, it behaves as P2.

Figure 30 shows a general example, the state transition between processes is triggered by e.

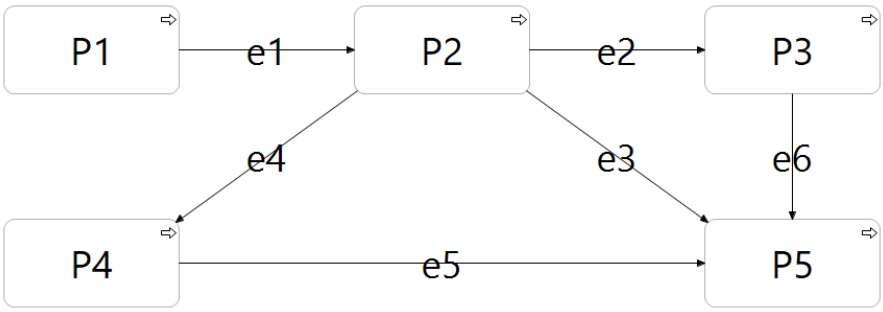

Figure 3 0 Conditional state transition in ArchiMate

6.1.3 Dependability Case Creation

In contemporary society, objective and reasonable evidence and explanations are necessary for the safety of a product. Dependability Case (D-Case) is a method for showing the dependability of the systems, to make third parties understand the dependability of a system. In this section, the results of model checking are proposed as evidence for the D-Case.

A formal example of D-Case is shown in Figure 31. The top goal G1, the system is safe, is decomposed into G2, G3, and G4 by strategy S1. Similarly, G2 is decomposed into G5 and G8 by S2, G3 is decomposed into G6 and G7 by S3. Finally, G1 is considered to be decomposed into G4, G5, G6, G7, and G8. These sub-goals are supported by evidence E5, E2, E3, E4, and E1. The evidence should be supported by the result of the model checking in step 2.

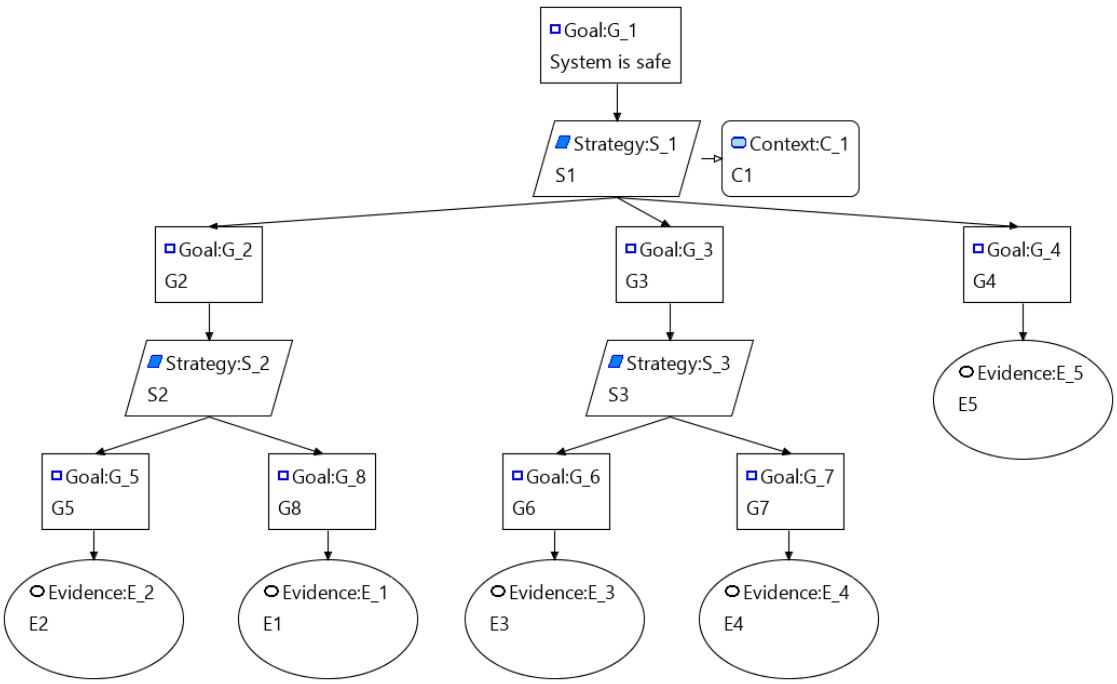

Figure 3 1 D-Case Creation

6.1.4 Composite Safety Assurance

In Figure 31, the top goal is decomposed into 5 sub-goals. Apply these safety goals to Figure 30, suppose that the safety goal for P2 is G4, and G4 is supported by P1; The safety goal for P3 is G5, and G5 is supported P2; The safety goals for P4 is G6, and G6 is supported by P5; The safety goals for P5 is G7, and G7 is supported by P3; The safety goals for P1 is G8, and G8 is supported by P4. In this part, the associate relationship is represented by “Association Relation” in ArchiMate, the support relationship is represented by “Realize Relation” in ArchiMate. A general example of composite safety assurance can be shown in Figure 31.

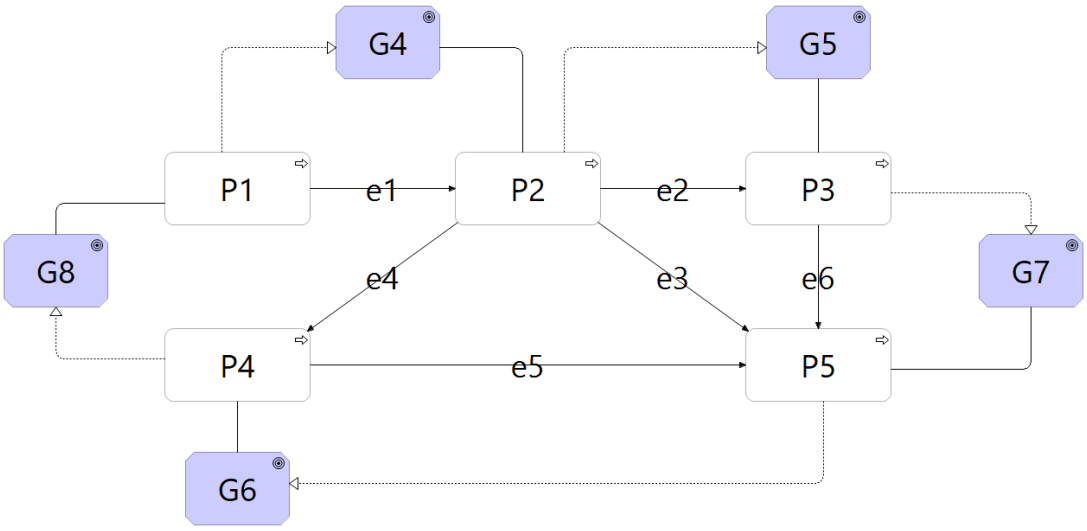

Figure 3 2 General example of composite safety assurance

6.2. Application of the Proposed Method

In this section, for the application of the proposed method, an automatic driving system is carried out.

Recently, the research of automatic driving is in great demand. Automatic driving system is a complex combination of components. It can be defined as a system of vehicle perception, decision, and operation through electronic equipment and machinery, rather than human drivers, as well as a system that automatically introduces road traffic. Since the automatic driving system can help people to correct driving errors or replace people with driving responsibilities, traffic accidents caused by driving are expected to be reduced [46].

SAE International and NHTSA (National Highway Traffic Safety Administration) define automation level [47], "driving mode" means "a type of driving scenario with characteristic dynamic driving task requirements [48]. In this part, the application for the proposed method will be conducted based on level 3 driving automation, which is “The driver can safely turn their attention away from the driving tasks, for example, the driver can text or watch a movie. The vehicle will handle situations that call for an immediate response, like emergency braking. The driver must still be prepared to intervene within some limited time, specified by the manufacturer, when called upon by the vehicle to do so”. Kinoshita proposed a CSP model for the level 3 driving automation [49].

Next, the discussion will focus on the safety assurance for the interaction between driver and automatic driving system (ADS) in level 3 automatic driving. As the previously

mentioned approach, the safety assurance could be implemented as follows.

6.2.1 Visualization of Interactions between Driver and ADS

For the interactions between driver and ADS, ADS provides driving information to drivers, or the driver sets the ADS, and then the ADS issues an operation command to the vehicle based on the setting. In order to model the behavior of a driver in the autonomous vehicle, it should be considered from the basic driving operations, which are recognition, judgment, and operation. Parasuraman [50] proposed a model for a driver in automatic driving, which includes Sensory, Perception, Decision Making, and Response Selection. This part follows this model, but it must also consider whether ADS can hand over the driving authority to the driver, because the driver may not be able to enter the driving state immediately for various reasons. If the driver is unable to enter the driving state immediately, the transfer of driving authority will fail. Besides, the driver's status is not static, if the ADS gives the driver a warning, the driver may return to the state where he can immediately enter the driving. Of course, the real situation may be more complicated, but it is beyond the scope of this chapter. Figure 33 shows the model of interactions between driver and ADS in automatic driving.

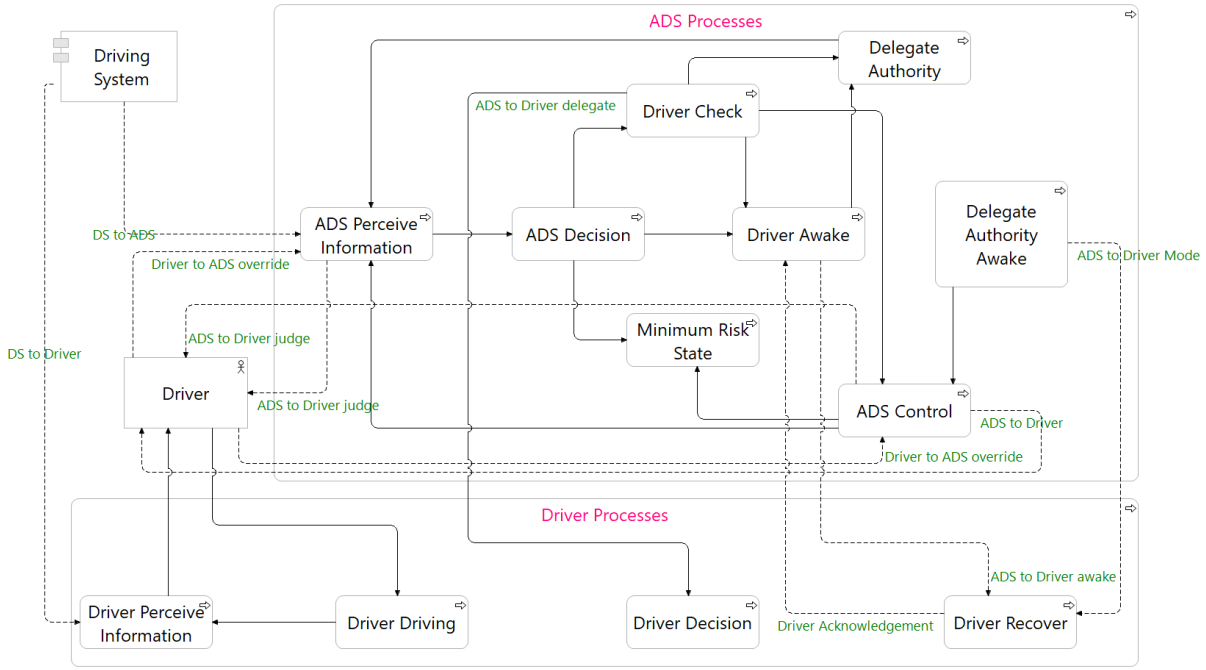

Figure 3 3 Automatic driving processes in ArchiMate

In this figure, there are Driving System, Driver Processes, and ADS processes. For the

ADS processes, in the state of automatic driving, ADS will constantly confirm whether the driver's state can participate in driving. If the driver fails to respond, or fails to perform proper operations within the necessary time, Minimum Risk State (MRS) process will be conducted to avoid serious safety problems. The arrows in the figure represent the process of possible state migration. To confirm the interactions between ADS and driver, this model will be checked to ensure that the interactions do not result in deadlock.

6.2.2 Model Checking for the Processes between Driver and ADS

In Figure 34, the solid lines denote the transfer of states and dotted lines denote the channels. For example, ADS and Driver perceive system information through channel “DS to ADS” and “DS to Driver”, then, the states of Driver Perceive Information and ADS Perceive Information vary according to channel information.

It is worth noting that in order to meet the safety requirements of automatic driving, reasonable ADS and Driver state variables should be set up. For example, for the driver state, a normal state means that the driver can drive immediately, an abnormal state means that the driver cannot drive immediately, the driver may be drowsy or inattentive. For the ADS state, there should be an automatic driving state and a limited state. Limited state means that it is difficult to automate in the current environment. In Figure 5-6, ADS first perceives environmental information through the ADS Perceive Information process, then determines the next behavior through the ADS Decision process. The Driver Check process detects whether the driver is in a normal state, if the driver is not in a normal state, the ADS will try to restore the driver state through the Driver Awake process. Also, the Minimum Risk State process will be conducted if necessary. For the safety, the transfer of driving authority between driver and ADS is very important in the level 3 driving automation, therefore, the transfer of driving rights should not be only determined by thee process ADS Control, but also by the Delegate Authority Awake process in this chapter. On the other hand, manual driving is conducted through the Driver Driving process when the driver judges that manual driving is necessary, the driver's intention is indicated to ADS through the Driver Decision process. The Driver Recover process responses the driver state to ADS. Figure 34 shows the CSP model of ADS processes. The CSP model of the driver is omitted in this research.

```
ADS_Perceive_Information = (DS_to_ADS?x -> ([x==0](adsstatechangeOK {ADS_state = automated;} -> ADS_Decision)
                [] [x!=0] (adsstatechangeNG {ADS_state = limited;} -> ADS_Decision)));
                [] (Driver_to_ADSoverride?x -> ([evdstate == adnormal] (ADS_to_Driver_judge!0 -> Driver_Awake)
                [] [evdstate == normal] (ADS_to_Driver_judge!1 - > ADS_Perceive_Information)));

ADS_Decision = [drivingMode == driverDriving && driverState == normal && ADS_state == automated](ADSdecision01 -> ADS_Perceive_Information)
              [] [drivingMode == driverDriving && driverState == normal && ADS_state == limited](ADSdecision06 -> Driver_Check)
              [] [drivingMode == driverDriving && driverState == normal && ADS_state == automated](ADSdecision05 -> Driver_Check)
              [] [drivingMode == driverDriving && driverState == abnormal && ADS_state == automated](ADSdecision07 -> Driver_Check)
              [] [drivingMode == driverDriving && driverState == normal && ADS_state ==limited] (ADSdecision02 -> ADS_Perceive_Information)
              [] [drivingMode == driverDriving && driverState == abnormal && ADS_state == automated](ADSdecision03 -> Driver_Awake)
              [] [drivingMode == MRS_driving](ADSdecision09 -> Minimum_Risk_State)
              [] [drivingMode == driverDriving && driverState == abnormal && ADS_state == limited](ADSdecision04 -> Minimum_Risk_State)
              [] [drivingMode == driverDriving && driverState == abnormal && ADS_state == limited](atomic{ADSdecision08 -> Minimum_Risk_State});

Driver_Check = [driverState == normal && ADS_state == automated] (ADS_Control)
              [] [driverState == abnormal] (Driver_Awake)
              [] [driverState == normal && ADS_state == limited] (ADS_to_Driver_delegate ! 1 -> Delegate_Authority);
ADS_Control = [drivingMode == ADS_Driving] (ADS_to_Driver!1 -> ((ADS_to_Driver!1 -> ADS_Perceive_Information)
              [] (Driver_to_ADS_override?x -> ([driverState == abnormal] (ADS_to_Driver_judge!0 -> Driver_Awake)
              [] [driverState == normal] (ADS_to_Driver_judge!1 -> ADS_Perceive_Information)))))
              [] [drivingMode == driverDriving && driverState == normal] (ADS_Perceive_Information)
              [] [drivingMode == driverDriving && driverState == abnormal] (Driver_Awake)
              [] [drivingMode == MRS_driving] (Minimum_Risk_State);

Delegate_Authority = Driver_to_ADS?x ->
                    if(x==1)
                    {drivingModechevd {drivingMode = driverDriving;} ->
                    ADS_Perceive_Information}
                    else {Minimum_Risk_State};

Delegate_Authority_Awake = ADS_to_Driver_Mode!0 -> ADS_to_Driver_Mode?x -> ADS_Control;

Minimum_Risk_State = mrs {drivingMode = MRS_driving;} -> Minimum_Risk_State;
```

Figure 3 4 CSP model describing ADS process in level 3 driving automation

The deadlocks in CSP model can be verified by defining "assert" function. For the ADS process in level 3 driving automation, the verification result is shown in Figure 35. The verification result can be used as evidence of D-Case in the next step.

```
Output
********Verification Result********
The Assertion (ADS() deadlockfree) is VALID.

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:40
Total Transitions:82
Time Used:0.0032506s
Estimated Memory Used:9594.976KB
```

Figure 3 5 Verification result of ADS process

6.2.3 Dependability Case Creation for ADS

The interactions between driver and ADS have been discussed, and the CSP model describing ADS processes also has been proposed in the previous section. According to the CSP model of ADS, the dependability case for the ADS is discussed as shown in Figure 36. It is worth noting that only the dependability case of ADS is discussed here based on the CSP model of the ADS, the other dependability cases such as driver or

driving environment will be omitted in this research.

In Figure 35, the top goal is that ADS can work with a driver to achieve safe driving. The top goal can be decomposed into sub-goals based on the actions of ADS. For the sub-goal G_2 “ADS can identify the driving environment”, is supported by the Evidence E_1, which is “ADS Perceive Information process is deadlock-free”. For the sub-goal G_5 “When the driver is in an abnormal state, correct measures should be taken to reduce risk”, is supported by the Evidence E_4, which is “Minimum Risk State is deadlock-free”. Likewise, so are other sub-goals. In this case study, each of the evidence corresponds to the process in step 1 and step 2. Step 1 visualized the interactions between these processes, and step 2 made model checking for these processes.

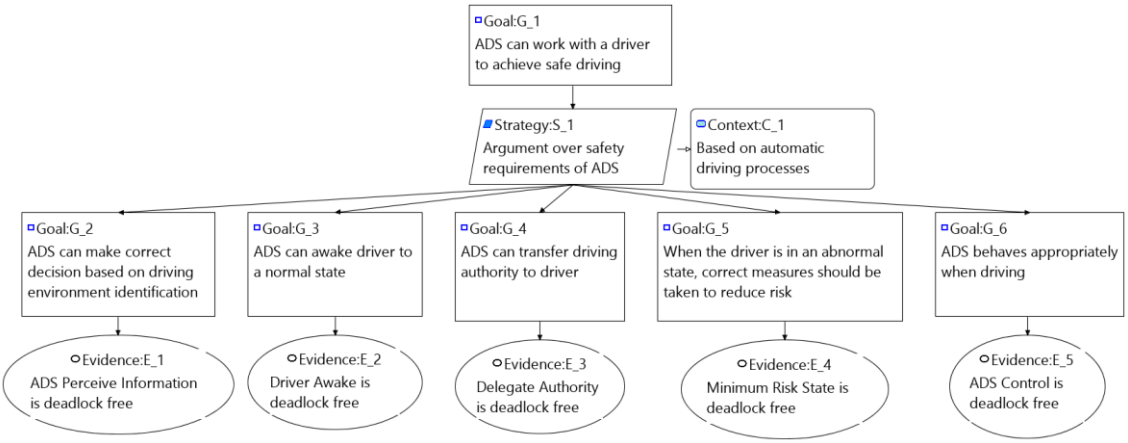

Figure 3 6 D-Case for the ADS process

6.2.4 Composite Safety Assurance for ADS

For the level 3 driving automation system, the interactions between a driver and ADS have been clarified as previously discussed. The content of Goal 2 is that the decision of ADS is correct, it is realized by ADS Perceive Information, which is verified in step 3, and associated with ADS Decision. The content of Goal 3 is that ADS can awake driver. Goal 3 is realized by Driver Awake, which is verified in step 3, and associated with Driver Recover. The content of Goal 4 is that ADS can transfer driving right to driver. Goal 4 is realized by Delegate Authority, which is verified in step 3, and associated with Driver Awake. The content of Goal 5 is that MRS could be conducted at the right time. Goal 5 is realized by Minimum Risk State, which is verified in step 3, and associated with ADS Decision. For Goal 6, ADS behaves appropriately when driving, is realized by ADS Control, which is verified in step 3, and associated with Driving System. The safety

requirements of ADS were also explained in Figure 36. The composite safety assurance for the target system is shown in Figure 37 as follows. Although not all dependability cases are considered, the methods of safety assurance are introduced in detail.

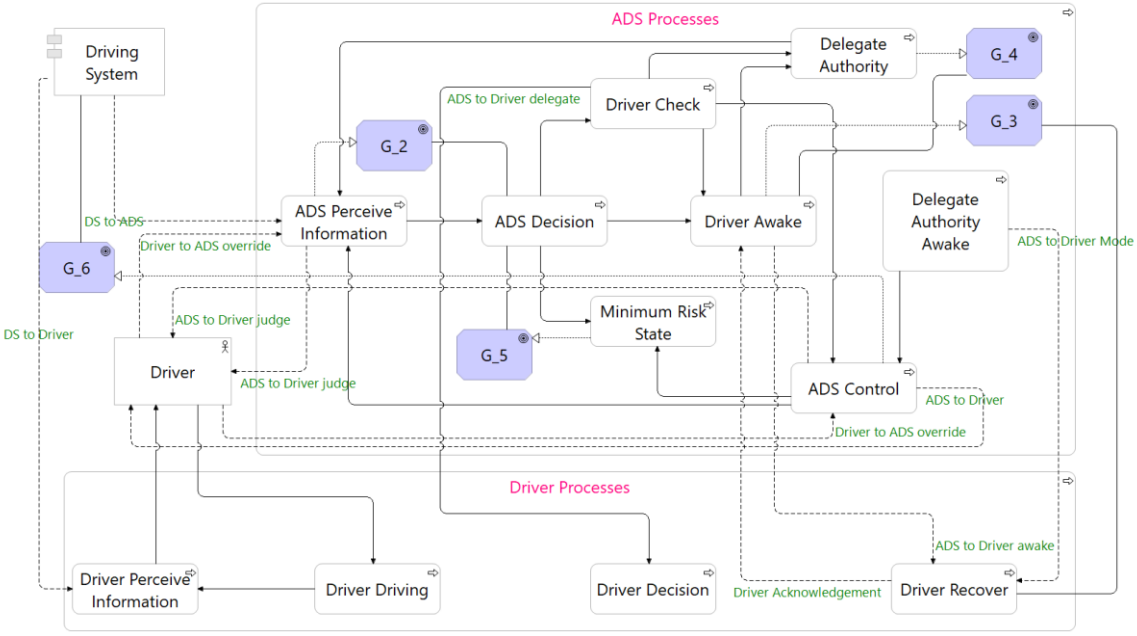

Figure 3 7 Composite safety assurance for ADS in ArchiMate

6.3 Discussion

6.3.1 Effectiveness

In the previous part, the proposed safety assurance approach was applied to a case study of automatic driving, this part makes a discussion on the case study.

Firstly, the interactions between a driver and ADS in autonomous driving were analyzed, and these relationships were visualized through ArchiMate. In the ArchiMate diagram, triggering relation is used to represent process direction, flow relation is used to represent messages passing between processes. Then model checking based on the ArchiMate diagram is carried out to verify whether there are logical errors or deadlocks in the processes. For the level 3 driving automation system, all the relation names and process names in the ArchiMate diagram are highly consistent with the CSP model. Also, D-Case is created based on the safety requirements of the driving automation system. It is noted that in this section, only the ADS was discussed through D-Case. The evidence for the safety goals in the D-Case could be supported by the result of the model checking.

Table 13 makes a comparison between the previous method d* framework and the

proposed method for system safety assurance. For d* framework, it can describe the safety goals between system components, but the visualization of the interactions for systems is insufficient. Besides, in d*framework, no consideration was given to the correctness of the process transition. For the proposed method, in the first step, the interactions between system actors are visualized through ArchiMate, state transition between processes is clarified. The second step conducts a model checking for the interactions based on the diagram in the first step to make sure if there are logical errors or deadlocks. Then step 3 discusses the safety requirements for the target system by using D-Case, the evidence for the D-Case could be supported by the result of step 1 and step 2, and the evidence is more clear and credible. At the last step, actors, processes, safety goals, and interactions are integrated by using the composite safety assurance method.

In summary, the proposed composite safety assurance method is superior to the previous method. It is more reliable and logically traceable. For the safety assurance of level 3 driving automation system, the proposed approach is effectively applicable.

Table 13 Comparison of d*framework and the proposed method

Comparison items	d* framework	Proposed method
Interaction visualization	Collaboration Diagram	ArchiMate
Logical check of interaction	-	Model Checking
Explicit argument based on evidence	Assurance Case	Assurance Case

6.3.2 Limitations

This method is suitable for systems with complex state transitions and interactions. In this section, although a case study of level 3 driving automation system was carried out, only partial safety issues of the target system were analyzed. Besides, the quantitative comparison with the previous approach is not considered. The comparative experiment to quantitatively evaluate the productivity and quality should be carried out to verify the effectiveness.

7 Conclusion

In this paper, we conducted a study on dependability assurance in system modeling. We analyzed the previous methods to find out the shortcomings, and proposed new methods to improve these shortcomings. For the previous method, the system architecture and assurance case need to be developed separately. We proposed a method named Intra Model Security Assurance to integrate system architecture and assurance case. So far assurance case is described by using a specific goal oriented diagram notation. The goal oriented diagram is different from the architectural diagrams, therefore engineers have the problem to manage two different diagrams for developing architecture and assuring security. The contribution of this method is that it clarified the security assurance method with only one EA modeling language. To integrate different diagrams, we clarified the relationship and notation method for system architecture and assurance case. Then we developed a mapping from the metamodel to EA modeling language, ArchiMate. To clarify the effectiveness of the proposed approach, we compared it with D-Case which is a derivative of GSN. The comparison consists of a case study and a quantitative experiment. The case study for a secure retrieval on cloud storage service showed that the proposed approach reduced the number of diagram nodes and relationships for those of the previous approach. The reduction percentages for nodes and relationships were about 44% and 36%, respectively. The experiment on Healthcare device and Smart house systems showed that the proposed approach improved the effort and correctness of investigation works on assuring security for those of the previous approach.

In Chapter 4, an approach was proposed to assure dependability in Enterprise Architecture which includes business layer, application layer, and technology layer. In this chapter, firstly, a model of the relationship for the business layer and assurance case was proposed; then the mapping relationships between the assurance case and actor were defined using ArchiMate based on this model. The presentation of the dependability between actors was also clarified. A case study of a digital signature process was carried out to explain this approach, and the study showed that the composite dependability for the business, application, and technology layers could be well treated using ArchiMate. Finally, a comparison between ArchiMate and d*framework was conducted, and the effectiveness and superiority of the proposed method were proven by analyzing the actors, dependability claim, relationship, and system configuration. Moreover, a comparison between the proposed method and d*framework regarding the number of nodes and relationships was conducted from a data perspective. The proposed method is more concise; and easier to understand than d*framework. The significance of the proposed method in terms of the enterprise architecture, is that it can directly assure the system

architecture. However, because d*framework uses UML, it cannot directly assure models of enterprise architecture.

For the modeling of safety-critical medical equipment, we proposed a composite safety assurance method, and gave a detailed description in Chapter 5. This method consists of five steps: Describe system architecture with ArchiMate, identify composite safety goals between components, safety goals elicitation, requirements elicitation for safety goals, safety goals assurance using composite requirements. In this section, a composite safety assurance method was proposed for safety-critical system architecture. First, the safety assurance model between system components was explained, then the mapping relationships were defined using ArchiMate based on this model. A case study with the insulin pump system was carried out to explain this approach, and the study showed that the composite safety assurance between system components could be well treated using ArchiMate. Finally, a comparison between ArchiMate and d*framework was conducted. The effectiveness and superiority of the proposed method were also proved by analyzing the system components, composite safety claim, and relationship.

In Chapter 6, for generalization, we have reduced this method to 4 steps, and added model checking. The purpose of model checking is to verify whether there is a logical error in the interaction between the process in the system architecture. This chapter proposed a composite safety assurance approach, and applied to level 3 driving automation system. The proposed approach consists of 4 steps: Interaction visualization, model checking, D-Case creation, and composite safety assurance. Compare with the previous method, the proposed method realizes interaction visualization and logical verification for process state transitions, which can make dependability assurance more reliable. For example, “Driver Awake” process in the case study try to wake up the driver and determine if the driver is in a state where he can drive, if not, “Minimum Risk State” process will be executed. Here, the logic of state transition needs to be checked. Especially for safety-critical systems, there is an urgent need for a more reliable way of safety assurance, because the argument of safety assurance would be insufficient or unconvincing in the absence of interaction visualization and process logical checking. In the proposed method, all steps are mutually linked. The proposed method is no longer a linguistic discussion, it is supported by verified logical evidence. The significance of this method is that it provides a formalized procedure for safety assurance, which boosts the confidence of system safety.

Throughout this paper, this work gives a new way to assure the dependability of system architecture. Compared with the previous methods, the proposed method has a comparative advantage in terms of visualization and operability. However, the

effectiveness of this method needs to be verified through practical applications.

Acknowledgment

I would like to express my gratitude to all those who helped me during my Ph.D. study at Nagoya University.

I gratefully acknowledge the help of my supervisor, Prof. Shuichiro Yamamoto, who has offered me valuable suggestions in the academic studies and whose useful suggestions incisive comments and constructive criticism have contributed greatly to the completion of this thesis. In the preparation of the thesis, his willingness to give me his time so generously has been much appreciated. His tremendous assistance in analyzing the proposed method and in having gone through the draft versions of this thesis several times as well as his great care in life deserve more thanks than I can find words to express. Truly, without his painstaking efforts in revising and polishing my drafts, the completion of the present thesis would not have been possible. I do appreciate his patience, encouragement, and professional instructions during my whole study and making suggestions for further revisions.

My sincere thanks are extended to Prof. Mutsunori Banbara and Prof. Shuji Morisaki. I greatly appreciate their feedback, suggestions and insightful perspective on my study. I am also greatly indebted to all my teachers in the graduate school of informatics of Nagoya University who have helped me directly and indirectly in my studies, from whose devoted teaching and enlightening lectures I have benefited a lot and academically prepared for the thesis. Any progress that I have made is the result of their profound concern and selfless devotion.

Thanks also go to our team members in Yamamoto Lab, especially: Zhengshu Zhou, Chengchen Xia, Koyo Kanamori, and Takashi Kaneko, who have given me the warm help and precious time to work out my problems.

Last but not least, my gratitude also extends to my family who has been assisting, supporting and caring for me all of my life.

Reference

[1] I. Sommerville, Software Engineering (10th Edition), Pearson, 2015.

[2] J. Jacobson, I. Booch and G. Rumbaugh, Unified Modeling Language Reference Manual, Addison-Wesley Professional, 2004.

[3] S. Friedenthal, A. Moore and R. Stenier, A Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufmann, 2014.

[4] International Organization for Standardization 15026-1:2019, Systems and software engineering -- Systems and software assurance, 2019.

[5] E. M. Clarke Jr, O. Grumberg and D. Peleg, Model Checking, The MIT Press, 1999.

[6] NATO, AEP-67:2017 Engineering For System Assurance In Nato Programmes, NATO Standardization Agreemens, 2010.

[7] E. Yu, Social modeling for requirements engineering, The MIT Press, 2011.

[8] S. Yamamoto and Y. Matsuno, "d* framework: Inter-Dependency Model for Dependability," in *International Conference on Dependable Systems and Networks*, Boston, USA, 2012.

[9] K. Tim and R. Weaver, "The Goal Structuring Notation – A Safety Argument Notation," in *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[10] The Open Group, *Dependability through Assuredness™ (O-DA) Framework,* 2013.

[11] Y. Matsuno, J. Nakazawa, M. Takeyama, M. Sugaya and Y. Ishikawa, "Toward a Language for Communication among Stakeholders," in *IEEE Pacific Rim International Symposium on Dependable Computing*, Tokyo, Japan, 2010.

[12] T. Saruwatari, T. Hishino and S. Yamamoto, "Method to Share Responsibility Knowledge of Dependability Cases," in *International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Kitakyushu, Japan, 2013.

[13] T. Kelly, Concepts and Principles of Compositional Safety Case Construction., COMSA, 2001.

[14] S. Yamamoto and M. Yutaka, "An Evaluation of Argument Patterns to Reduce Pitfalls of Applying Assurance Case," in *Proceedings of ASSURE*, San Francisco, USA, 2013.

[15] S. Yamamoto, T. Kaneko and H. Tanaka, "A Proposal on Security Case Based on Common Criteria," in *Information and Communicatiaon Technology - International Conference*, Yogyakarta, Indonesia, 2013.

[16] S. Yamamoto, "Assuring Security through Attribute GSN," in *International Conference on IT Convergence and Security*, Kuala Lumpur, Malaysia, 2015.

[17] M. Lankhorst, Enterprise Architecture at Work, Springer-Verlag Berlin Heidelberg, 2013.

[18] The Open Group, ArchiMate 3.0 Specification, Van Haren Publishing, 2016.

[19] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, Non-Functional Requirements In Software Engineering, Kluwer Academic Publishers, 2000.

[20] E. Grandry , C. Feltus and E. Dubois, "Conceptual Integration of Enterprise Architecture Management and Security Risk Management," in *17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, Vancouver, British Columbia, Canada, 2013.

[21] M. Korman, T. Sommestad , J. Hallberg, J. Bengtsson and M. Ekstedt, "Overview of Enterprise Information Needs in Information Security Risk Assessment," in *IEEE 18th International Enterprise Distributed Object Computing Conference*, Ulm, Germany, 2014.

[22] I. Band, Modeling enterprise risk management and security with the ArchiMate language, A White Paper Published by The Open Group, 2015.

[23] W. Abbass, A. Baina and M. Bellafkih, "Improvement of information system security risk management," in *4th IEEE International Colloquium on Information Science and Technology*, Tangier, Morocco., 2016.

[24] N. Mayer and C. Feltus, "Evaluation of the risk and security overlay of ArchiMate to model information system security risks," in *IEEE 21st International Enterprise Distributed Object Computing Conference*, Quebec City, Canada, 2017.

[25] D. Moody, "The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 35, no. 6, pp. 756-779, 2009.

[26] S. Yamamoto, "An approach to assure Dependability through ArchiMate," in *International Conference on Computer Safety, Reliability, and Security*, Trondheim, Norway, 2015.

[27] Q. Zhi, S. Yamamoto and S. Morisaki, "IMSA - Intra Model Security Assurance," *Journal of Internet Services and Information Security,* vol. 8, no. 2, pp. 18-32, 2018.

[28] S. Yamamoto, Q. Zhi and S. Morisaki, "A Composite Dependability for Enterprise Architecture," in *International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Belgrade, Serbia, 2018.

[29] DEOS, "D-CASE PROCESS," DEOS, [Online]. Available: http://deos.or.jp/technology/dcase-j.html.

[30] P. Bishop and R. Bloomfield, "A Methodology for Safety Case Development," *Industrial Perspectives of Safety-critical Systems,* pp. 194-203, 1998.

[31] A. V. Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 2001.

[32] M. Hagglund, "A new approach for goal-oriented analysis of healthcare processes," *Studies in Health Technology and Informatics,* pp. 1251-1255, 2010.

[33] Y. An, P. W. Dalrymple, M. Rogers, P. Gerrity and E. Yu, "Collaborative social modeling for designing a patient wellness tracking system in a nurse-managed health care center," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, Philadelphia, Pennsylvania, 2009.

[34] A. Eldein, H. Ammar and D. Dzielski, "Enterprise architecture of mobile healthcare for large crowd events," in *International Conference on Information and Communication Technology and Accessibility*, Muscat, Oman, 2017.

[35] K. Ahsan, H. Shah and P. Kingston, "Healthcare Modelling through Enterprise Architecture: A Hospital Case," in *Seventh International Conference on Information Technology: New Generations*, Las Vegas, NV, USA, 2010.

[36] S. Yamamoto, I. O. Nada and S. Morisaki, "Using ArchiMate to Design e-Health Business Models," *ACTA SCIENTIFIC MEDICAL SCIENCES,* vol. 2, no. 7, 2018.

[37] Q. Zhi, Z. Zhou and S. Yamamoto, "Visualized Assurance Approach for Enterprise Architecture," *Journal of Information and Communication Convergence Engineering,* vol. 17, no. 2, pp. 117-128, 2019.

[38] B. Cameron, E. Crawley and D. Selva, System Architecture, Pearson Education Limited, 2015.

[39] R. F. Stapelberg, "Infrastructure Systems Interdependencies and Risk Informed Decision Making (RIDM): Impact Scenario Analysis of Infrastructure Risks Induced by Natural, Technological and Intentional Hazards," *SYSTEMICS, CYBERNETICS AND INFORMATICS,* vol. 6, no. 5, 2008.

[40] S. M. Rinaldi, J. P. Peerenboom and T. K. Kelly, "Identifying, understanding, and analyzing critical infrastructure interdependencies," *IEEE Control Systems Magazine,* vol. 21, no. 6, pp. 11-25, 2001.

[41] A. V. Lamsweerde, Requirements Engineering: From System Goals to UML Models to

Software Specifications, Wiley, 2009.

[42] L. P. Plotnick, L. M. Clark, F. L. Brancati and T. Erlinger, "Safety and Effectiveness of Insulin Pump Therapy in Children and Adolescents With Type 1 Diabetes," *Diabetes Care,* vol. 26, no. 4, pp. 1142-1146, 2003.

[43] L. Henimann, A. Fleming, J. R. Petrie, R. Holl, R. M. Bergenstal and A. L. Peters, "Insulin Pump Risks and Benefits: A Clinical Appraisal of Pump Safety Standards, Adverse Event Reporting, and Research Needs A Joint Statement of the European Association for the Study of Diabetes and the American Diabetes Association Diabetes Technology W," *Diabetologia,* vol. 38, no. 4, pp. 716-722, 2015.

[44] Y. Zhang, P. L. Jones and R. Jetley, "A Hazard Analysis for a Generic Insulin Infusion Pump," *Journal of Diabetes Science and Technology,* vol. 4, no. 2, pp. 263-283, 2010.

[45] Y. Matsuno, H. Takamura and Y. Ishikawa, "A Dependability Case Editor with Pattern Library," in *International Symposium on High Assurance Systems Engineering*, San Jose, CA, USA, 2010.

[46] U.S. Departement of Tranportation, "Federal Automated Vehicles Policy-Accelerating the Next Revolution In Roadway Safety," 2016.

[47] SAE International, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," 2016.

[48] SAE International, "Automated Driving-Levels Of Driving Automation," 2017.

[49] S. Kinoshita, S. Yun, N. Kitamura and H. Nishimura, "Analysis of a Driver and Automated Driving System Interaction Using a Communicating Sequential Process," in *IEEE International Sysmposium of Systems Engineering*, 2015.

[50] R. Parasuraman, T. B. Sheridan and C. D. Wickens, "A Model for Types and Levels of Human Interaction with Automation," *IEEE TRANSACTIONS ON SYSTEMS,* vol. 30, no. 3, pp. 286-297, 2000.

[51] T. Saruwatari and S. Yamamoto, "Definition and application of an assurance case development method (d*)," *SpringerPlus,* vol. 2, no. 224, 2013.

[52] S. Yamamoto, "An Approach for Evaluating Softgoals Using Weight," in *Information & Communication Technology-Eurasia Conference*, Daejeon, Korea, 2015.

[53] G. Wierda , A Serious Introduction to the Archimate(r) Enterprise Architecture Modeling Language, R&a, 2014.

[54] S. Yamamoto and N. Kobayashi, "Mobile Security Assurance through ArchiMate," in *International Symposium on Mobile Internet Security*, Taichung, Taiwan, 2016.

[55] Y. Liu, J. Sun and J. Dong, "PAT 3: An Extensible Architecture for Building Multi-

domain Model Checkers," in *IEEE 22nd International Symposium on Software Reliability Engineering*, Hiroshima, Japan, 2011.

[56] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM,* vol. 21, no. 8, pp. 666-677, 1978.

Paper List

I. Journal

Qiang Zhi, Shuichiro Yamamoto, Shuji Morisaki, "IMSA -Intra Model Security Assurance," Journal of Internet Services and Information Security, vol. 8, no. 2, pp. 18-32, 2018.

Qiang Zhi, Zhengshu Zhou, Shuichiro Yamamoto, "Composite Safety Assurance for Healthcare Devices“, ACTA SCIENTIFIC MEDICAL SCIENCES, Vol 3, Issue 9, pp. 14-22, 2019

Qiang Zhi, Zhengshu Zhou, Shuichiro Yamamoto, “Visualized Assurance Approach for Enterprise Architecture”, Journal of Information and Communication Convergence Engineering, vol.17, no.2, pp. 117-127, 2019.

Zhengshu Zhou, Qiang Zhi, Shuji Morisaki, Shuichiro Yamamoto, “IMAF: A Visual Innovation Methodology Based on ArchiMate Framework”, International Journal of Enterprise Information Systems, Vol 16, pp. 31-52, 2020.

II. International Conference

Qiang Zhi, Yamamoto Shuichiro, Shuji Morisaki, “Quantitative Evaluation in Security Assurance”, IEEE International Conference on Computer and Communications,Chengdu, pp. 2477-2483, China, 2018.

Qiang Zhi, Zhengshu Zhou, Shuji Morisaki, Shuichiro Yamamoto,"An Approach for Requirements Elicitation using Goal, Question, and Answer", in International Congress on Advanced Applied Informatics, pp. 847-852, Toyama, Japan, 2019

Zhengshu Zhou, Qiang Zhi, Shuichiro Yamamoto, “A Proposal for Developing EA Models toward Innovation”, International Congress on Advanced Applied Informatics, pp. 853-858, Toyama, Japan, 2019.

Xia Chengchen, Qiang Zhi, Zhengshu Zhou, Shuichiro Yamamoto, “An Approach to

Transform Enterprise Architecture Models from Systemigrams", International Conference on Software Engineering and Service Science, pp. 571-574, Beijing, China, 2019.

Zhengshu Zhou, Qiang Zhi, Shuichiro Yamamoto, Zilong Liang, "Automated Reasoning towards Quantitative Security Assurance", International Conference on Software Engineering and Service Science, pp. 394-399, Beijing, China, 2019.

Shuichiro Yamamoto, Qiang Zhi, Shuji Morisaki, "A Composite Dependability for Enterprise Architecture", International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, pp. 1130-1137, Belgrade, Serbia. 2019.

Shuichiro Yamamoto, Qiang Zhi, "ArchiMate Business Model Patterns to e-Healthcare", Innovation in Medicine and Healthcare Systems, and Multimedia, vol 145, pp. 11-20, Singapore.

Shuichiro Yamamoto, Qiang Zhi, Zhengshu Zhou, "Aspect Analysis towards ArchiMate Diagrams", Procedia Computer Science, vol. 159, pp. 973-980, 2019.