Safer Autonomous Systems

European Training Network on

Safer Autonomous Systems (SAS)

Delivarable3.2 - Report on executable safety assurance case models for autonomous drilling use case

Author: Fang Yan[1], Einar Landre[2], Simon Foster[1], Ibrahim Habli[1]

[1]University of York, UK

[2]Equinor, Norway

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788

Abstract

This report is based on the use case study of the executable safety assurance case generation during the virtual secondment of ESR 10 with Equinor from 1st Dec 2020 to 30th Jan 2021. The solution of executable assurance case generation was proposed by ESR 10 prior to the secondment, and has been applied during the secondment to the Equinor underwater AUV system, an autonomous vehicle that provides the underwater maintenance and intervention to the drilling facilities on the seabed.

The planned use case of Equinor was an autonomous drilling system instead of AUV. The reason to switch the use case to undersea AUV is that autonomous drilling application is too far into the future thus not possible to provide sufficient system data for use case study; while AUV Operations has many more applications, and are better aligned with ongoing research and concepts, furtherly can be tested in practice using off-the-shelf hardware.

The report presents the general process of the executable assurance case generation and the application of the solution to the AUV case step by step.

The use case study has shown the effectiveness of the solution, and the further generalization will be considered in future work.

Table of Contents

List of Figures

List of Tables

1. Introduction

Assurance cases (AC) are essential to the operation of safety-critical systems. ACs are defined as compelling arguments, supported by evidence, that systems operate as intended for defined applications in defined environments. It is a systematic way to argue the indispensable properties, such as safety and security. ACs are recommended in some safety standards, such as ISO 26262, and Def Stan 00-55.

For some traditional safety-critical systems whose operational environment is relatively predictable, their ACs evolve at a low rate and, therefore, are regarded as relatively "static" in nature.

However, it is difficult to assure the system safety as the operational environment of the Robotics and Autonomous System (RAS) is not sufficiently predictable during design time compared to the traditional safety-critical systems, causing uncertainty of completeness of environment boundary, the unpredictability of composition of the configurations for system of system (e.g. platoon), validity of safety guarantee for runtime monitoring predefined during design time.

This uncertainty could lead to the need for revalidation and reverification of system design during runtime. Though system updating is not specific to RAS, due to its uncertainty, RAS will face more frequent updates than traditional safety-critical systems.

ACs are constructed along with the system development process. One of the issues for AC generation during design phase is the repeated workload due to the system development iteration. After entering the operation, the RAS system may face a higher frequency of updating which results again in the ACs evolution. That is to say the frequent update of system design and therefore of AC is unavoidable in both design phase and operation phase.

Therefore, an automatic way for AC generation is desired for RAS systems which may involve machine readable AC to realize the automation.

ERS 10's subject of 'From static assurance cases at design-time to executable assurance cases at run-time' is under two Research Hypothesis. 1) Assurance case process is engineering experience oriented, can't be replaced by tool. But the process can be assisted by automation tool for higher efficiency and accuracy. 2) Assurance case process is not a documenting process to only collect inputs data and generate a document, but an analysis process.

In terms of the technical solution, Model-based Engineering (MBE) is explored for the purpose of automatic AC generation and evolution.

MBE has been a well-adopted technique for system development thanks to its efficient tool support. With the success in system design, its applications have expanded into the surrounding aspects including AC generation.

MBE techniques brings the capability to the engineers of syntactical validation, model checking, model simulation, model to text transformations, model to model transformations, and code generation, etc. [1], among which a very interesting advantage is the possible automatic traceability from system models to AC models, and automatic updating of AC input from system models, which may relieve the manual workload of and improve the accuracy of AC updates and its evaluation during the AC evolution.

Part of the ESR 10's work is to propose a technique for generating executable AC models which can facilitate the automatic upgrade of the AC models when the system design changes. The benefit to

generate AC in the form of model is that the AC models then can be manipulated through Model-based Engineering techniques with mature tool support. This work has been achieved by exploiting mainly the model transformation techniques.

During the secondment with Equinor, the techniques of the executable AC generation has been applied to the Equinor use case- Autonomous Underwater Vehicle (AUV). Through the case study, the solution was revised and upgraded.

The work has three research contributions:

- A solution for automatic generation of AC
- An assurance case pattern for transportation robotic system
- A metamodel for unifying the reference data interface

2. Assurance generation process

The process includes three parts as shown Figure 2- 1.

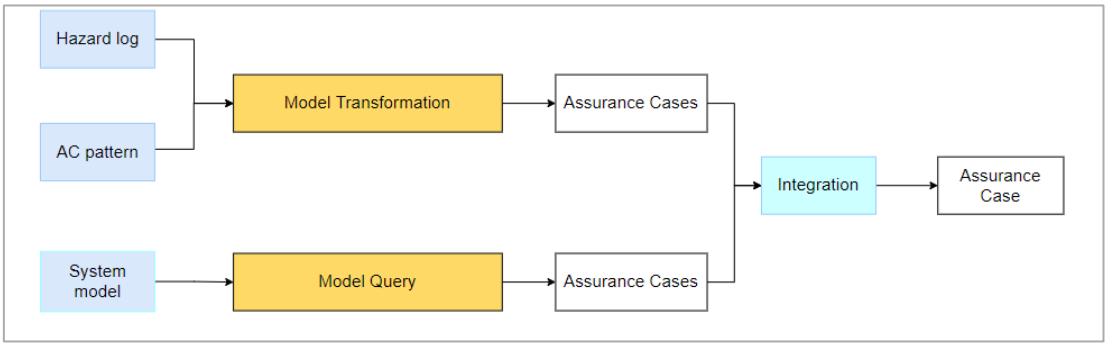

Figure 2- 1 Assurance case generation process

To construct an AC, several sources of data are needed. Here we consider the AC generation from hazard log, and from system models. With a complete hazard log, the AC can be generated in a complete form except for the evidence reference. When the system is developed with Model-based Engineering, the system models can be involved in the AC generation. Since system model is usually the design detail, it may contribute to the sub-claims of the AC. Thus, the AC generated from system models is usually not a complete AC structure, the top-level claims for hazards still need to be generated from hazard log.

The first part includes steps 1-3 for AC model generation from a hazard log.

The second part includes steps 4-6 for AC model generation from system models.

The third part includes steps 7-9 for AC model integration from the first two parts.

More details are explained with the use case application in Section 4.

Step 1-3: AC model generation from hazard log

The relationship among steps 1-3 is shown in Figure 2- 2.

Step 1: generate EMF model for Hazard log

- to create the hazard log in excel for the use case.
- to design the metamodel for hazard log.
- to convert the hazard log in excel format into EMF model by EOL execution.

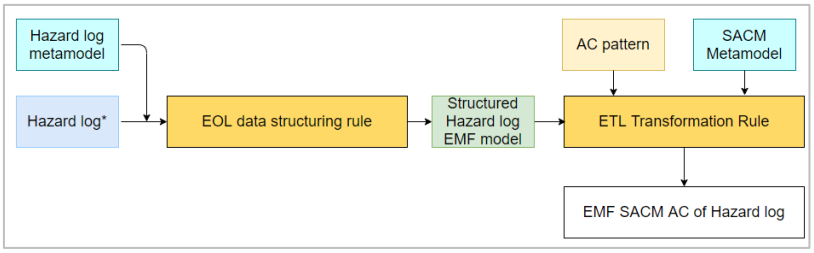

Figure 2- 2 AC model generation from hazard log

Step 2: AC pattern design

Step 3: AC model generation from hazard log

- to build mapping between metamodels of hazard log and SACM through transformation rule according to the AC pattern with the transformation language of ETL
- to execute the ETL transformation to output EMF model of AC

Step 4-6: AC model generation from system models

Step 4: AC pattern design for AC model generation from system model

Step 5: system model query mechanism design by EOL

Step 6: model query execution to output AC models

Step 7-9: AC model integration

Step 7: Hazard log modification to include the keyword “Query” for system model query

Step 8: Hazard log to SACM AC transformation update to incorporate keyword “Query”

Step 9: system model query update to incorporate keyword “Query” to output the integrated AC models

2.2 Process summary

The whole picture of the process is described in Figure 2- 3. The mains steps of three transformation are marked as yellow, including the transformation from Excel hazard log to EMF hazard log, the transformation from EMF hazard log model to SACM AC model, and the transformation from system model to SACM AC model.

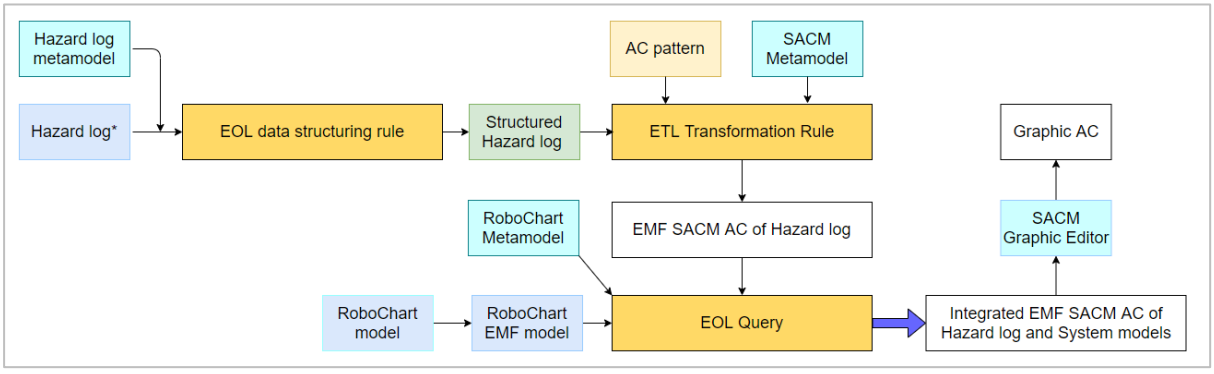

Figure 2- 3 Expanded research process

3. Use case

3.1 System description

The planned use case of Equinor was an autonomous drilling system instead of AUV. The reason to switch the use case to undersea AUV is that autonomous drilling application is too far into the future thus not possible to provide sufficient system data for use case study; while AUV Operations has many more applications, and are better aligned with ongoing research and concepts, furtherly can be tested in practice using off-the-shelf hardware.

According to “TR1231 Appendix D UID Subsea Docking Station requirements”[2], SR-77497 -Level of Autonomy, the AUV can be operated by human, or by system but supervised by human. The mission of the system is the underwater maintenance and intervention tasks. The hazard of the system is the collision of AUV with different types of subsea system components and infrastructure, which can be caused by operator or AUV systems under different control modes.

For the system design details necessary for the AC generation, we refer the system design in [3]. According to [3], we assume that the AUV system is controlled by a safety control unit - LRE whose function is to switch the operation mode of the system based on the safety condition of operation. There are four operational mode: (i) OCM, the operator control mode, (ii) MOM, the automatic mode in safe condition, (iii) HCM, the automatic mode where the collision risk is to be reduced by reduce speed, (iv) CAM, the emergency automatic mode where the collision risk is too high and need to be reduced by maneuver.

3.2 System modelling language

The system modelling language is RoboChart, a DSL developed by RoboStar group, tailored from UML state machines. The language is enriched with facilities to define time properties. It has complete formal semantics in the CSP process algebra. The modelling environment is RoboTool, an Eclipse-based graphical modelling tool. The development of correct RoboChart models can also be supported by Model checking (FDR/PRISM) and theorem proving (Isabelle) tools.

The RoboChart language structure is illustrated in Figure 3- 1. The system is modelled as a module. In each module, there is one platform representing the hardware of the system, such as sensors, actuators, and several controllers representing the software components of the system. The behavior of the controller is defined by state machines.

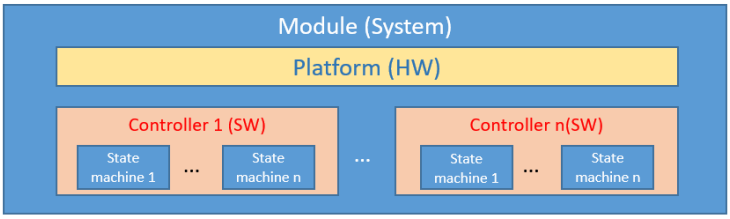

Figure 3- 1 Structure of RoboChart language

The RoboChart models can be in the form of EMF models which is the form needed for model transformation with Epsilon family.

3.3 AUV system models

The main system models are as following. Figure 3- 2 shows the whole system architecture. Figure 3-3 shows the structure of component LRE and the parameters. Figure 3- 4 shows the state machines of LRE. The state machines describe the behavior of LER. The system operation starts from OCM mode which is the manual control state where the system accept operator control inputs and send them to the autopilot. When (i) the AUV powers up; (ii) at the end of a task; or (iii) the operator requests, OCM state shall be entered. State MOM shall be entered from OCM when the following conditions hold: (i) the velocity is less than 0.1 ms^{-1}; (ii) the distance to a static obstacle is > 300mm; (iii) the distance to a dynamic obstacle is greater than 7500mm; or (iv) the operator requests it. When entering MOM, the LRE shall advise Autopilot a maximum velocity of 1 ms^{-1}. When the safety margin is reduced, state HCM shall be entered from MOM when either:(i) the AUV has a horizontal velocity > 0.1 ms^{-1} and is close to a static obstacle horizontally; (ii) the AUV has a vertical velocity > 0.1 ms^{-1} and is close to a static obstacle vertically; (iii) the AUV is close to a static obstacle; or(iv) the operator requests it. When entering HCM, the LRE shall advise a maximum velocity of 0.1 ms^{-1}. Once the risk is cleared, state MOM shall be entered back from HCM. The Emergency state CAM shall be entered if (i) it is not in OCM and (ii) there is an obstacle with an unsafe trajectory.

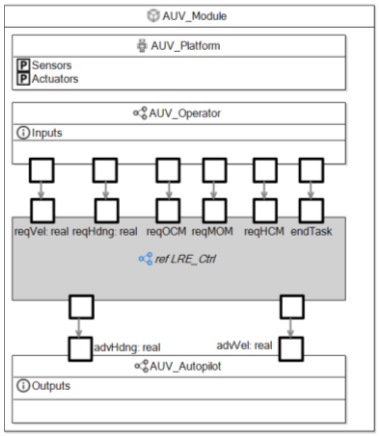

Figure 3- 2 System module model

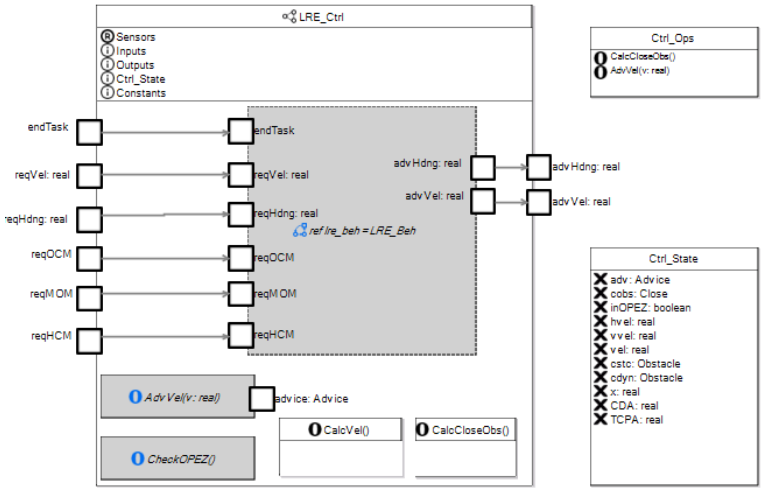

Figure 3- 3 Component LRE models

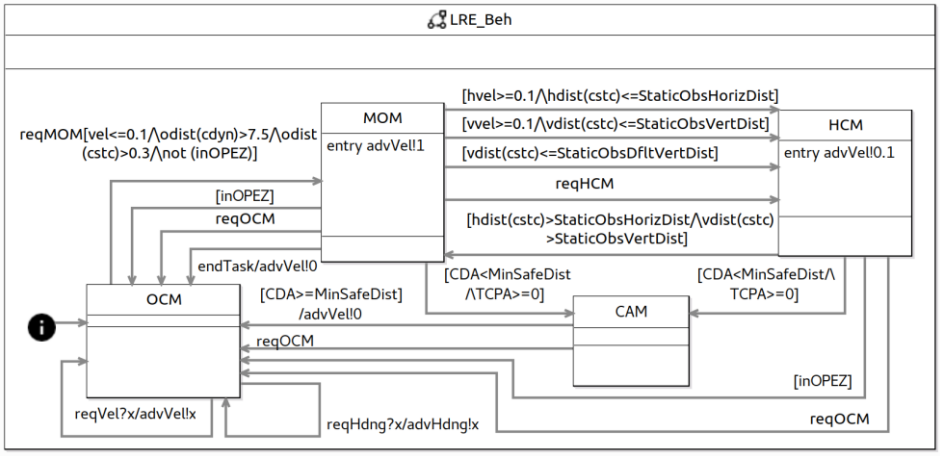

Figure 3- 4 LRE state machine models

4. Case study

Step 1-3: AC model generation from hazard log

Step 1: Generate EMF model for Hazard log

1) To create the hazard log in excel for the use case

The hazard log for hazard H1 and H2 shown in Table 4- 1 is generated through safety analysis. As the safety analysis process is out of the scope of the research, the relative process and method are not discussed here. We use the hazard log as input for the AC model generation.

2) To design the metamodel for hazard log

The metamodel is build in EMF, the code is attached in Attachment 1 EMF code for hazard log metamodel.

In the metamodel, one abstract class NamedElement is defined, all the elements in the hazard log including hazard, cause, safety measure, verification, validation, and together with the hazard log itself are defined as extends classes of class NamedElement.

The class of safetyDecisionRationale will be included in the hazard log later.

Table 4- 1 Hazard log for hazard H1 and H2

Hazard Id	Hazard Description	Cause Id	Cause Description	Sub-Cause Id	Sub-Cause Description	Safety Measure Id	Safety Measure Description	Sub-Safety Measure Id	Sub-Safety Measure Description	Verification Id	Verification Description	Verification Evidence	Validation Id	Validation Description	Validation Evidence
H1	Operator's improper control leads to AUV collides with obstacles	C1-1	wrong procedure in operation manual			M1-1-1	Operator shall follow the correct procedure to control system.			VR1-1-1-1	Operation procedure simulation	procedure simulation report			
										VR1-1-1-2	Operation procedure testing	procedure testing report			
										VR1-1-1-3	Operation procedure peer review	procedure peer review report			
		C 1-2	operator's improper operation			M1-2-1	operator shall receive proper training			VR1-2-1-1	Operator training manual peer review	training manual peer review report			
		C1-3	the operation interface is misleading			M1-3-1	the interface shall be user friendly.			VR1-1-2-1	Operation interface evaluation	Operation interface evaluation report			
H2	AUV system failure leads to AUV collision with obstacles	C2-1	Operator can not obtain the control from the state in which the operator is not in control when requesting			M2-1-1	Operator shall be able to obtain control from any state in which the operator is not in control when requesting.	M2-1-1.1	State **HCM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.1-1	Model query	Model query result : exist transition {} satisfies the safety measure			
								M2-1-1.2	State **MOM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.2-1	Model query	Model query result : exist transition {} satisfies the safety measure			
								M2-1-1.3	State **CAM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.3-1	Model query	Model query result : exist transition {} satisfies the safety measure			
		C2-2	system component malfunction during automatic mode												

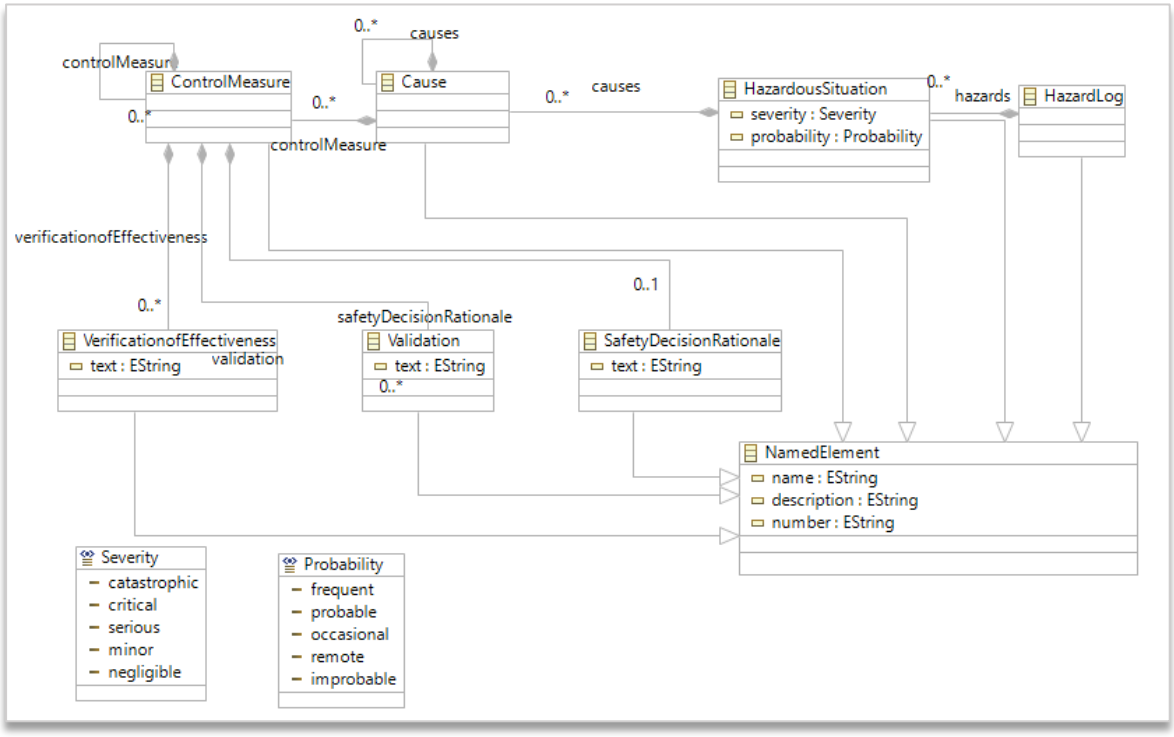

Figure 4- 1 EMF metamodel of hazard log

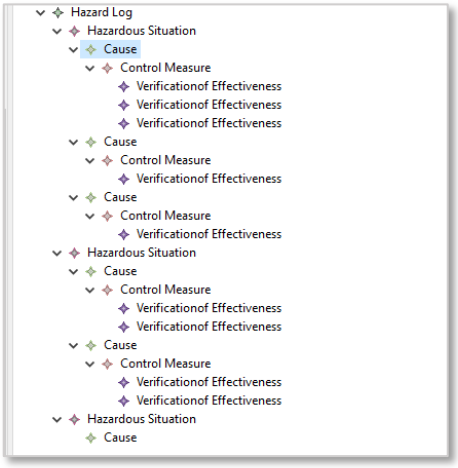

Figure 4- 2 EMF model of hazard log

Two enumerations are also defined as Severity and Probability, both of which are attributes of class hazard.

The Figure 4- 1 below is automatically generated from EMF metamodel of hazard log.

Step 2: AC pattern design

This pattern is designed for the AC that use hazard log in Step 1 as input, and the system architecture is not taken into account yet and will be in step 4. It follows the top-down strategy to decompose the top claim of hazard H1 by arguing over identified cause and sub-causes, then by arguing over safety measures and sub safety measures, then by the relative validation and verification methods to show the evidence. And this process is in compliant with the hazard log metamodel.

As shown in Figure 4- 3 AC pattern for hazard log, the context, assumption, and justification are only present for the top claim for the simplicity of the graph, but can be at any level of the claims and strategies.

In engineering practice, the AC pattern used can be different or modified from the pattern here. But all the hazard log elements in Step 1, the metamodel of hazard log in step 2, and the transformation rule in Step 3 need to be align with the AC pattern.

The GSN AC shown in Figure 4- 4 is manually created with hazard H1 in Step 1 and based on the AC pattern in Figure 4- 3 to help reader have an idea of the AC to be generated. The causes have no sub-level causes, and same as the safety measures.

In step 3, this AC will be generated automatically using the model transformation.

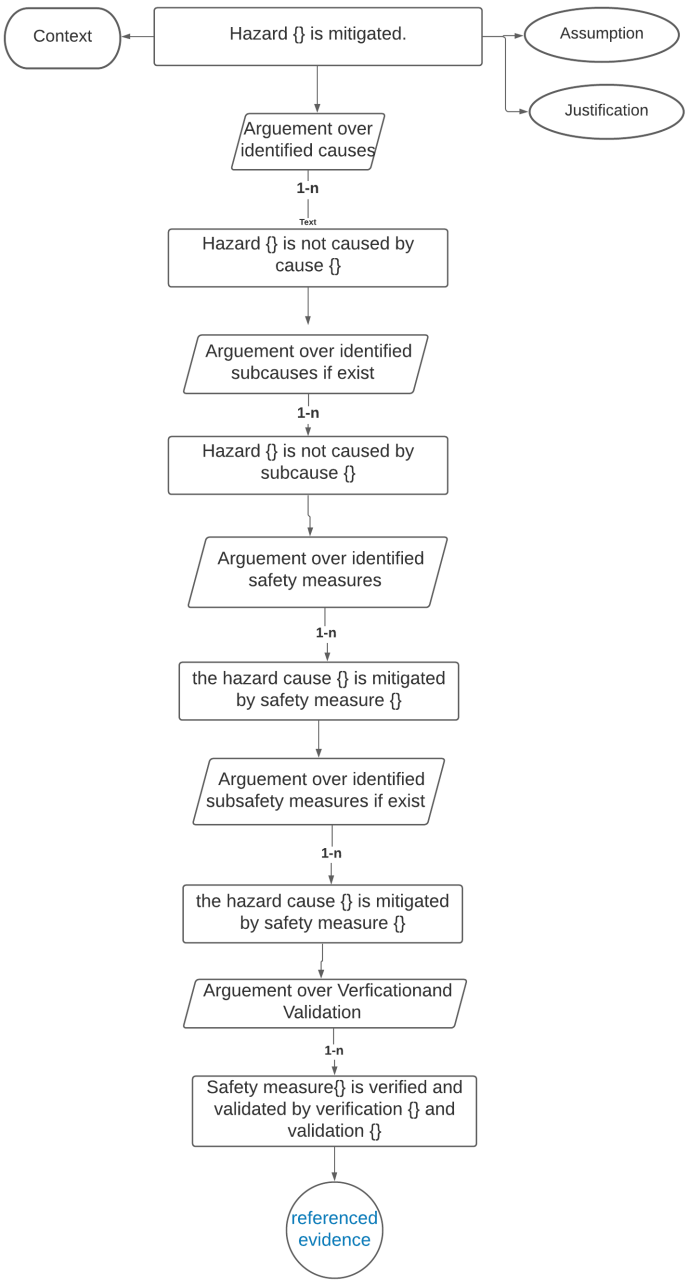

Figure 4- 3 AC pattern for hazard log

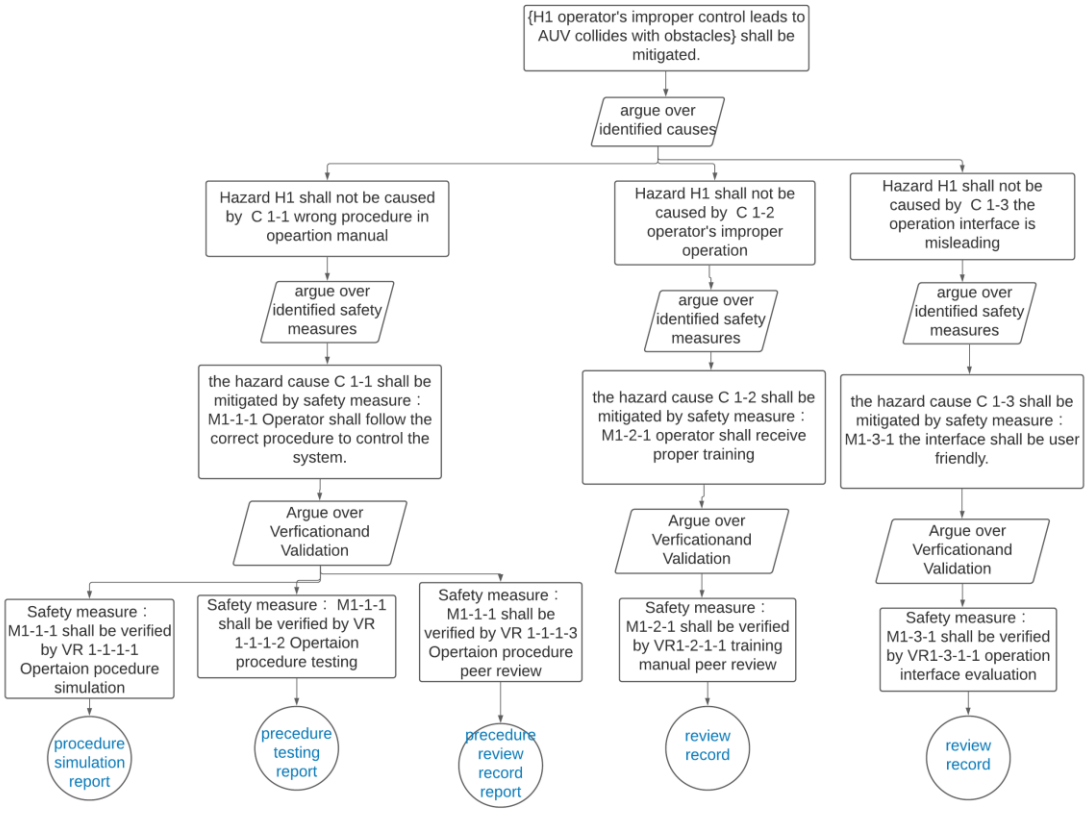

Figure 4- 4 AC for hazard H1

Step 3: AC model generation from hazard log

1) build mapping between metamodels of hazard log and SACM through transformation rule according to the AC pattern.

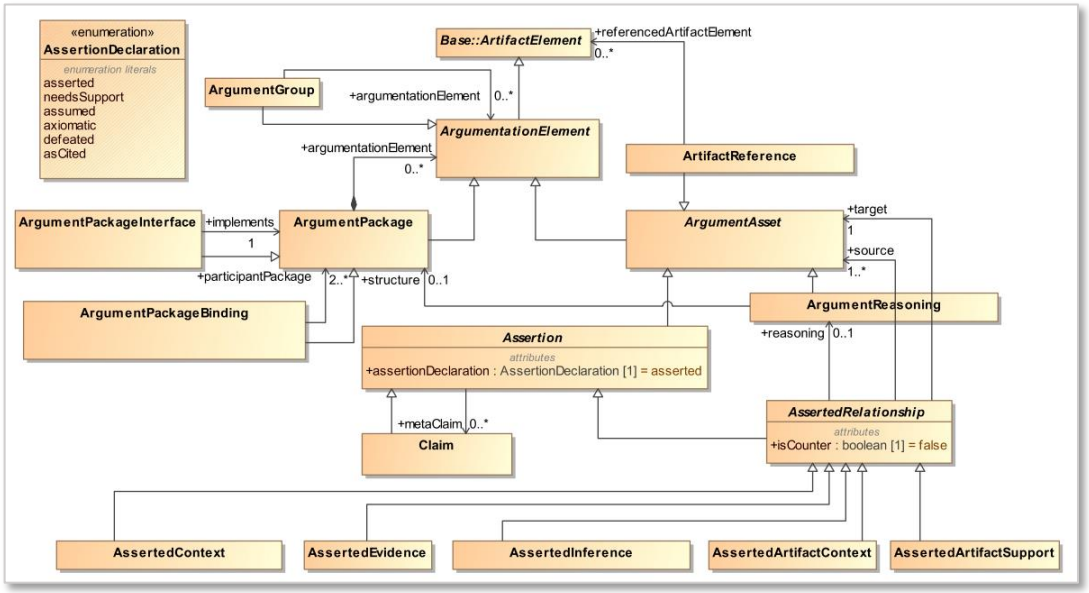

Figure 4- 5 Argumentation Package of SACM metamodel [4]

In the AC pattern in step 2, all the hazard, causes, subcauses, safety measures, and sub-safety measures provide the type for claims. The strategy represents the logical relationship between claims, such as “the claim that “the hazard shall be mitigated” shall be argued over the identified causes”.

SACM metamodel is partially shown in Figure 4- 5. The mapping between SACM and GSN: The claims are goals in GSN, the Argument Reasoning is the strategy in GSN, the Asserted Inference records the logical relationship between goals or goal and strategy. The AssertedEvidence represent the logic relationship between goals or goal and evidence.

The ETL code is recorded in Attachment 3 ETL code for transformation of hazard log model to SACM AC model (not including the content in blue mark).

The main structure of ETL is as following:

- Create claims for each hazard, cause, subcause, safety measure, sub-safety measure, verification, and validation;
- Create ArgumentReasoning between each layer of the claims above;
- Create AssertedInference to link the layers together.

It seems straight forward that the claim for hazard shall be decomposed to claims for causes, causes to sub-causes, sub-cause to safety measure, safety measure to sub-safety measure, sub-safety measure to verification/validation.

Since the hazard log metamodel allows the causes of one tier or multiple tiers, it is possible that the sub-cause is not defined, and the safety measures are linked to causes directly; and same for the safety measure with verification/validation. So if-else structure is frequently used to address this.

2) The ETL transformation rule is executed to output an EMF model of AC. The AC is represented in a tree-based format as shown in Figure 4- 6. Since the classes of claim, asserted inferences, and argument reasoning are parallel objects in the SACM metamodel, the hierarchy among them are not shown in the EMF models, but will be shown in the graphical editor by the link between the elements.

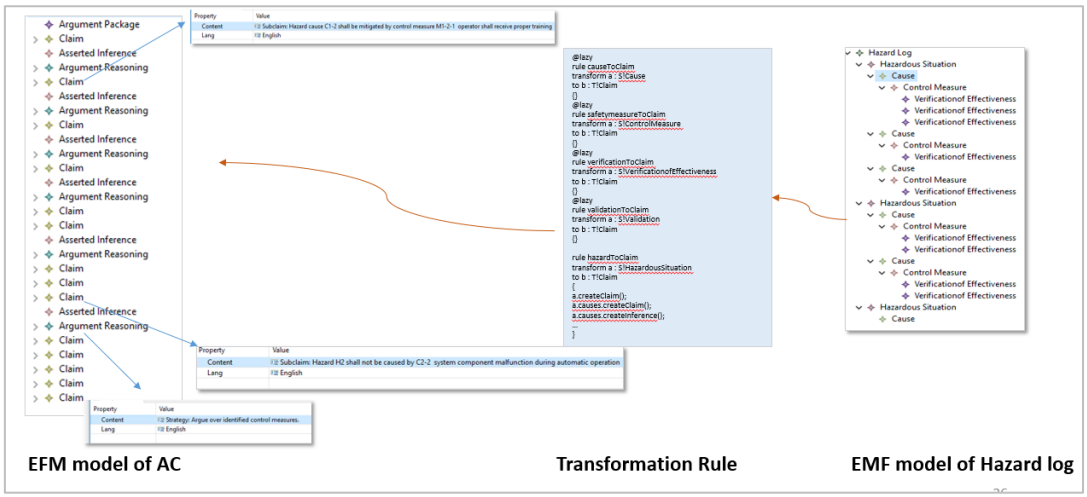

Figure 4- 6 EMF model generation from hazard log models

Step 4-6: AC model generation from system models

Step 4: AC pattern design for AC model generation from system model

For this activity, H2 is used as this hazard argument involves system models.

The hazard can be mitigated by safety measures that “each RoboChart State should have an outgoing transition whose trigger is reqOCM, and target is OCM” (Sub-safety measure M2-1-1.1…n) as shown in Table 4- 2.

As shown in the system model in section 3, three states in the state machine need to be examined as HCM, MOM, and CAM. The state HCM has an outgoing transition to OCM triggered by transition t2, the state MOM has an outgoing transition to OCM triggered by transition t3, the state CAM has an outgoing transition to OCM triggered by transition t17.

The identification of the states to examined and the transitions to satisfy the requirement can be done manually by model review.

The additional validation activity is needed to assure the states are complete.

Table 4- 2 Hazard log for H2

Hazard Id	Hazard Description	Cause Id	Cause Description	Sub-Cause Id	Sub-Cause Description	Safety Measure Id	Safety Measure Description	Sub-Safety Measure Id	Sub-Safety Measure Description	Verification Id	Verification Description	Verification Evidence	Validation Id	Validation Description	Validation Evidence
H2	AUV system failure leads to AUV collision with obstacles	C2-1	Operator can not obtain the control from the state in which the operator is not in control when requesting			M2-1-1	Operator shall be able to obtain control from any state in which the operator is not in control when requesting.	M2-1-1.1	State **HCM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.1-1	Model review	Model review result : exist transition {} satisfies the safety measure			
								M2-1-1.2	State **MOM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.2-1	Model review	Model review result : exist transition {} satisfies the safety measure			
								M2-1-1.3	State **CAM** should have a outgoing transition whose trigger is ReqOCM, and target is OCM	VR2-1-1.3-1	Model review	Model review result : exist transition {} satisfies the safety measure			
		C2-2	system component malfunction during automatic mode												

Then GSN for the whole hazard log of H2 can be created automatically as in Figure 4- 7 with the Steps 1-3.

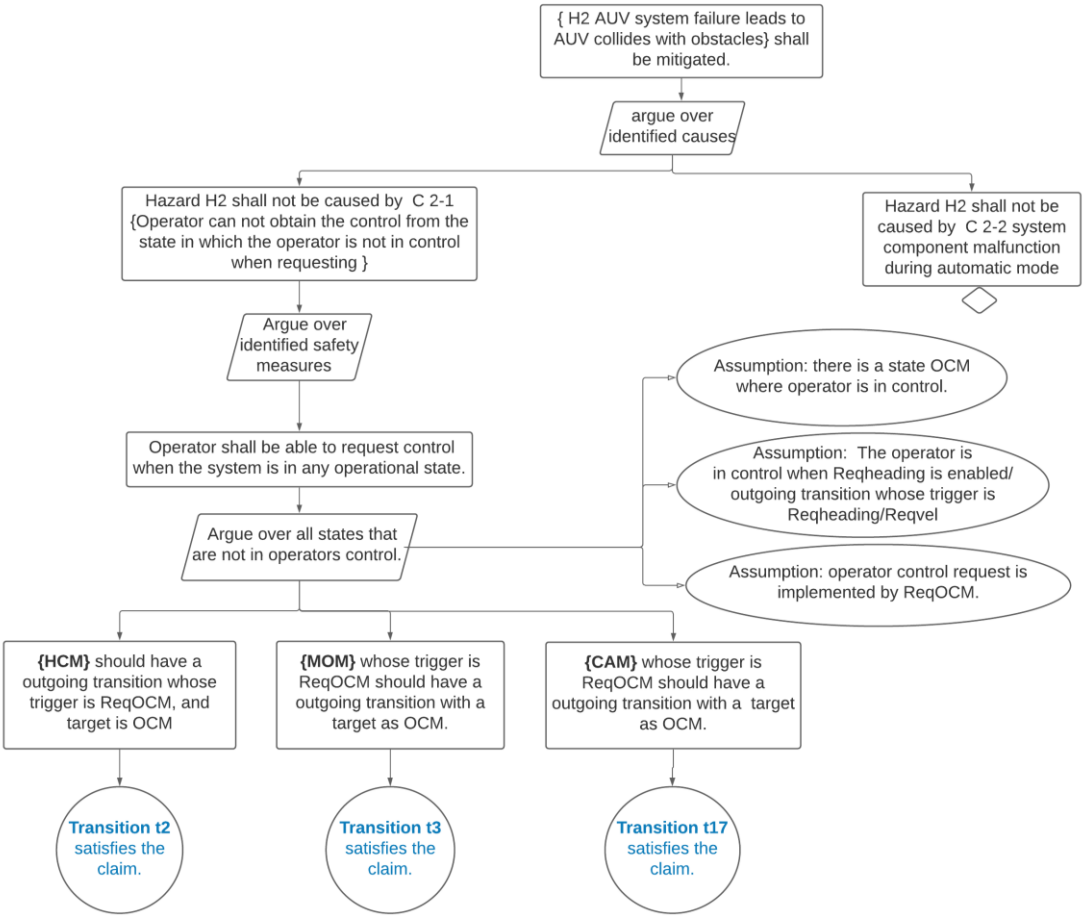

Figure 4- 7 GSN AC of hazard H2

But also, the sub-safety measure can be abstracted as "**each state {RoboChart State}** should have an outgoing transition whose trigger is reqOCM, and target is OCM", and can be filled with concrete state with the returned value from model query. The abstracted form of hazard log and GSN are shown below in Table 4- 3 and Figure 4- 8.

This abstraction is then used in the model query mechanism in Step 5 for automation.

The states to be examined and the transitions to satisfy the requirement then can be identified automatically by model query.

Thus, the AC model for columns of "SubSafety Measure Description", "Verification Id", and "Verification Evidence" could be generated by model query in Step 5 as well.

The additional validation activity (e.g. peer review) to assure the states are complete is not needed as long as the query rule is correct.

Table 4- 3 Abstracted hazard log

Hazard Id	Hazard Description	Cause Id	Cause Description	Sub-Cause Id	Sub-Cause Description	Safety Measure Id	Safety Measure Description	Sub-Safety Measure Id	Sub-Safety Measure Description	Verification Id	Verification Description	Verification Evidence	Validation Id	Validation Description	Validation Evidence
H2	AUV system failure leads to AUV collides with obstacles	C2-1	Operator can not obtain the control from the state in which the operator is not in control when requesting			M2-1-1	Operator shall be able to obtain control from any state in which the operator is not in control when requesting.	M2-1-1.1…n	each state {RoboChart State} should have an outgoing transition whose trigger is reqOCM, and target is OCM	VR2-1-1.1-1…n	Model query	Model query result：exist transition {} satisfies the safety measure			
		C2-2	system component malfunction during automatic mode												

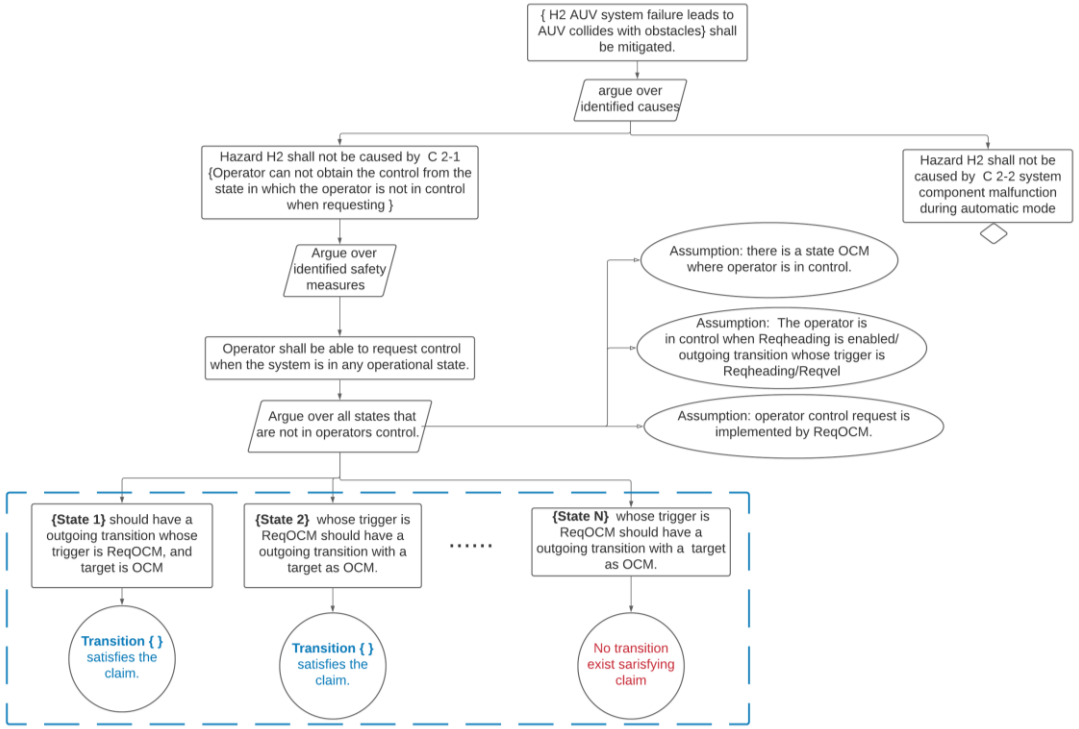

Figure 4- 8 Abstraction of H2 GSN

Step 5: System model query mechanism design by EOL

This step provide the way to generate AC models when model query technique is involved.

When the claim is to be satisfied by all the elements that meet certain conditions in the certain hierarchy of the system architecture, it is possible to search all the system models by model query and return the results. This can reduce the errors by manual search of system models and improve the work efficiency through automation.

The main code structure for the query is abstracted here as follows:

```
For (s in S! State.allInstances)
{
If (s.name != “OCM” and s.name != “f0”)
{
s.creatClaim();
s.getTransitionName();
s.creatEvidence();
s.buildRelationshipToEvidence();
}
}
```

In the use case, the model elements to be examined are the states that are not OCM or initial state, and for each of these states, a claim shall be created as “state {RoboChart State} should have an outgoing transition whose trigger is reqOCM, and target is OCM”.

For each identified state, its transitions are searched to identify the transition that meets the condition that “its trigger is reqOCM, and target is OCM”.

If the transition exists, it will be returned as the evidence that the claim is satisfied; otherwise, the evidence is set as “claim not satisfied.”

The code is in Attachment 4 EOL code for AC model generation from system models (not including the content in blue mark).

Step 6: Model query execution to output AC models

The AC model generated by model query is not a complete AC, as it does not cover the top structure of AC including hazards, causes.

The EMF model of AC is shown below in Figure 4- 9, it represents the GSN structure in Figure 4- 10 in the green frame. The upper structure of GSN framed in purple should be generated from hazard log following Step 1-3.

Until now, these two parts of AC in the green frame and purple frame are generated as two separate AC models instead of an integrated one. The integration will be implemented through Step 7-9.

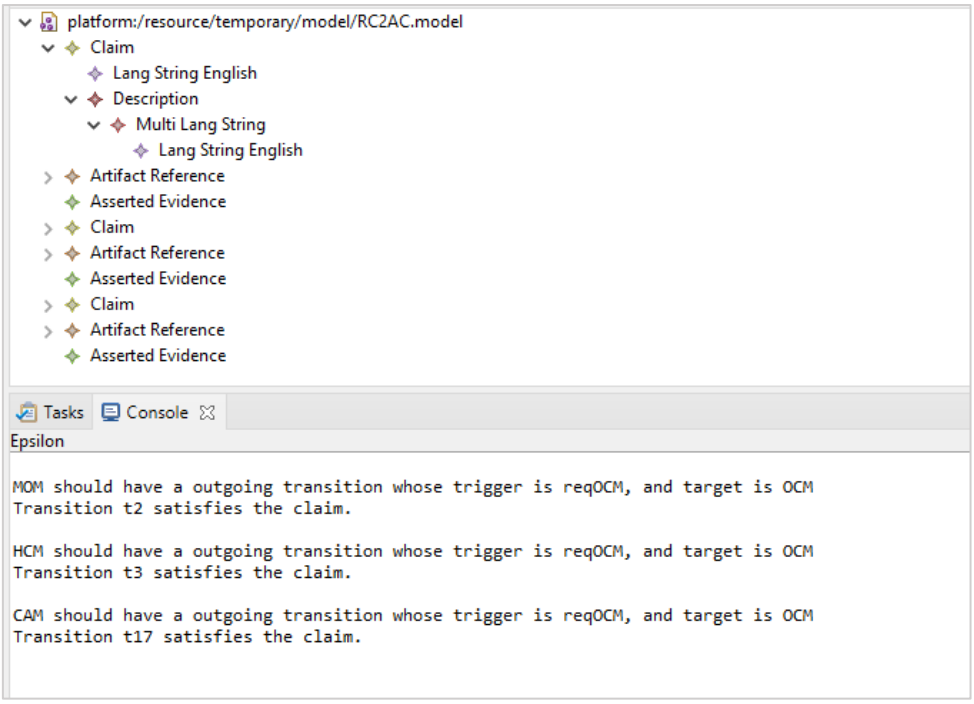

Figure 4- 9 EMF model of AC generated from system model query

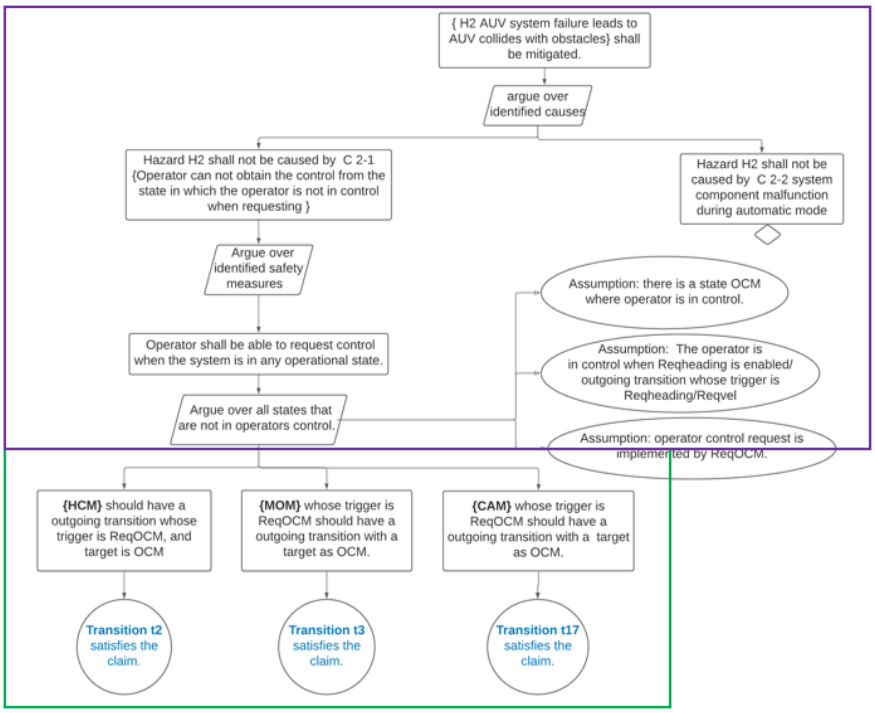

Figure 4- 10 GSN form of AC model from system model query

Step 7-9: AC model integration

Step 7: Hazard log modification to include the keyword “query” for system model query

For hazard H2, we can obtain the upper structure of GSN from hazard log through Step 1-3, and the lower structure from system models through Step 4-6.

Then we would desire a way to generate a complete AC in an integrated way.

The solution is to insert an identifier “Query” in the hazard log when the claim or evidence is to be generated by model query as shown in *Table 4- 4*. During query, the first thing is to look for this identifier.

Step 8: Hazard log to SACM AC transformation update to incorporate keyword “Query”

Based on the new hazard log in Step 7, the ETL code is updated to incorporate the scenario that the identifier exists.

The updated code is shown in Attachment 3 ETL code for transformation of hazard log model to SACM AC model. The blue marked codes are the update for the identifier.

The pseudocode for the update part is as follow:

```
if (m.controlMeasure.isDefined() and (not m.controlMeasure.startstWith("Query")))
{
m.controlMeasure.createClaim();
claim.name= “Query”;
inference.source = claim;
}
```

The idea is to first judge if the identifier exists, if it does then a claim is defined with the name of “Query”, all the following elements after this safety measure do not need further processing. And this claim is linked to the upper claims by set it as the source of the AssertedInference.

If the identifier doesn’t exist, then the procedure is to define a claim for this safety measure with the information in the hazard log, then to define claims and evidence for the elements after this safety measure.

Step 9: System model query update to incorporate keyword “Query” to output the integrated AC models

In order to link the AC model generated from hazard log and the AC model from system models, the **AC model generated from hazard log from Step 8 will be used in this step as both the source model and target model**.

The other source model is RoboChart EMF model.

The update code is shown in Attachment 4 EOL code for AC model generation from system models.

The blue marked codes are the update for the identifier.

The pseudocode for the update part is as follow:

```
for(inference in HazardLogAC!AssertedInference)
{
  if (inference.source = “Query”)
     inference.source.clear();
}
```

This is inserted in front of the original code.

The idea is to find in the AC model generated from hazard log in Step 8 the “Query” claim by searching the `inference.source`, and then to delete this claim. The following codes will define the new claims and link those to the inference as the source.

Figure 4- 11 shows the integrated AC generated from both hazard log and system models. The right figure is the EMF model of AC, and the left figure is the GSN representation. Also the upper part is the AC generated from hazard log, and the lower part is the AC generated from system models.

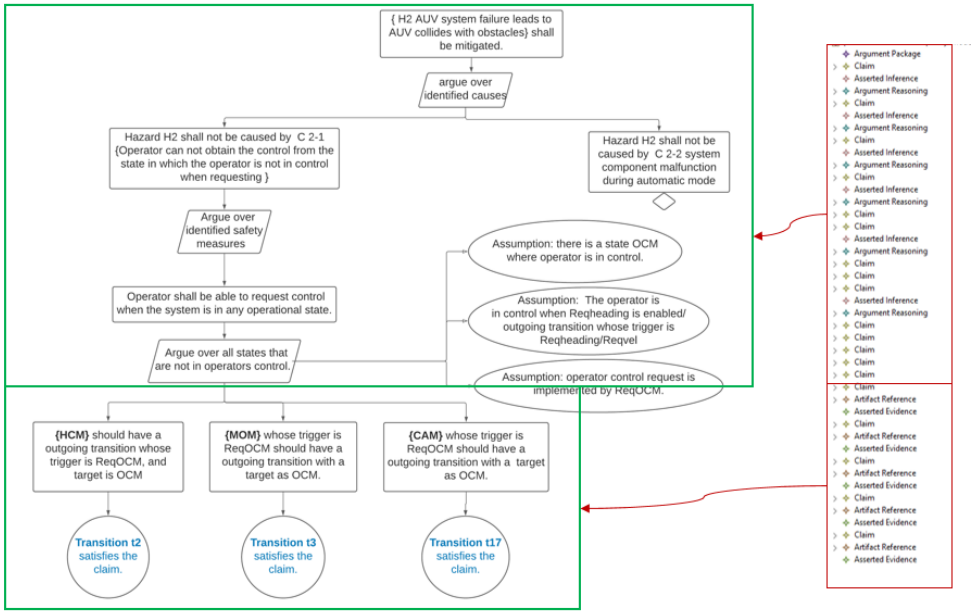

Figure 4- 11 Integrated AC generation from model transformation

Table 4- 4 Hazard log with Identfier

Hazard Id	Hazard Description	CauseId	Cause Description	SubCause Id	SubCause Description	Safety MeasureId	SafetyMeasure Description	SubSafety MeasureId	SubSafety Measure Description	Verification Id	Verification Description	Verification Evidence	ValidationId	Validation Description	Validation Evidence
H2	AUV system failure leads to AUV collides with obstacles	C2-1	Operator can not obtain the control from the state in which the operator is not in control when requesting			M2-1-1	Operator shall be able to obtain control from any state in which the operator is not in control when requesting.	M2-1-1.1…n	**Query**: each state {RoboChart State} should have an outgoing transition whose trigger is reqOCM, and target is OCM	VR2-1-1.1-1…n	Model query	Model query result：exist transition {} satisfies the safety measure			
		C2-2	system component malfunction during automatic mode												

5. Related work

[5] provides an automatic method for AC pattern instantiation. However, each type of instantiable data is labelled manually as a goal, context, strategy, solution in the form of a data table, by which the instantiation tool may automatically recognize the data type correctly. In [6], a similar strategy was exploited with the data form of artefact tree which is not different from the data table in essence.

[7] proposed a new domain-specific language Resolute, which is also a tool for building ACs based on AADL design models. The Resolute AC is generated through querying the AADL model within the system model development environment. This is an integrated way to generation AC and allows the automatic update of AC whenever system models change. However, this method is limited to the AADL language and its environment. Also, the AC is not capable of showing the hazard analysis process.

[8] proposed a formal way to model the assurance case to create an accurate mathematical model of the assurance argument. No common notation e.g. GSN or CAE or standard metamodel e.g. SACM is followed. The work provides a systematic process for AC modelling combined with system development process. However, the special expertise may be required for engineers.

[9] exploits the concept of weaving model for AC generation. The instantiable data can be extracted from the system model automatically. Also, the links between AC elements and system data can be updated automatically when the system design changes because the links are built between the metamodels instead of specific system data. Our work is based on the concept of [9] and made concrete implementation of 1) expanding the model source as multiple models, 2) AC generation from multiple models, 3) expanding the application scope from AADL to Eclipse supported system modelling language, 4) generating AC that is compliant with SACM 2.0.

6. Conclusion and future work

The work implements a framework for automatic generation of executable assurance case that is compliant with SACM 2.0. for the use case AUV, including threefold:

- A MDE solution for automatic instantiation of AC pattern using hazard analysis input
- A MDE solution for generating executable AC from system models
- The implementation of integration of AC generated from hazard analysis and system models

Future work may include the incorporation of other AC element (context, assumption, etc.), the AC evolution process, the further evaluation by Equinor. Also, the solution is based on Epsilon framework, the generalization of the solution will be considered in future work.

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*, vol. 3, no. 1. Morgan & Claypool Publishers, 2017.

[2] Equinor, “TR1231 Appendix D UID Subsea Docking Station requirements.”

[3] S. Foster, Y. Nemouchi, C. O’Halloran, K. Stephenson, and N. Tudor, “Formal model-based assurance cases in Isabelle/SACM: An autonomous underwater vehicle case study,” in *Proceedings of the 8th International Conference on Formal Methods in Software Engineering*, 2020, pp. 11–21.

[4] Object Management Group (OMG), “Structured Assurance Case Metamodel (SACM),” 2020. [Online]. Available: https://www.omg.org/spec/SACM/About-SACM/.

[5] E. Denney and G. Pai, “Tool support for assurance case development,” *Autom. Softw. Eng.*, vol. 25, no. 3, pp. 435–499, 2018.

[6] A. Agrawal, S. Khoshmanesh, M. Vierhauser, M. Rahimi, J. Cleland-Huang, and R. Lutz, “Leveraging Artifact Trees to Evolve and Reuse Safety Cases,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1222–1233.

[7] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, “Resolute: an assurance case language for architecture models,” in *ACM SIGAda Ada Letters*, 2014, vol. 34, no. 3, pp. 19–28.

[8] Z. Diskin, T. Maibaum, A. Wassyng, S. Wynn-Williams, and M. Lawford, “Assurance via model transformations and their hierarchical refinement,” in *Proceedings - 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018*, 2018, pp. 426–436.

[9] R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly, “Weaving an Assurance Case from Design: A Model-Based Approach,” in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 110–117.

Attachment 1 EMF code for hazard log metamodel

```
@namespace(uri="http://org.eclipse.acme/1.0/hazardlog", prefix="hazardlog_")
package hazardlog;

abstract class NamedElement {
  attr String name;
  attr String description;
  attr String number;
}

class HazardLog extends NamedElement {
  val HazardousSituation[*] hazards;
}

class HazardousSituation extends NamedElement {
  attr Severity severity;
  attr Probability probability;
  val Cause[*] causes;
}

class Cause extends NamedElement {
  val ControlMeasure[*] controlMeasure;
  val Cause[*] causes;
}

class ControlMeasure extends NamedElement {
  val SafetyDecisionRationale safetyDecisionRationale;
  val VerificationofEffectiveness[*] verificationofEffectiveness;
  val Validation[*] validation;
  val ControlMeasure[*] controlMeasure;
}

class VerificationofEffectiveness extends NamedElement {
  attr String text;
}

class Validation extends NamedElement {
  attr String text;
}

class SafetyDecisionRationale extends NamedElement {
  attr String text;
}

enum Severity {
  catastrophic = 1;
  critical = 2;
  serious = 3;
  minor = 4;
  negligible = 5;
}
enum Probability {
  frequent = 1;
  probable = 2;
  occasional = 3;
  remote = 4;
  improbable = 5;
}
```

Attachment 2 EOL code for transformation of hazard log from Excel to EMF model

```
var hazard;

var cause;

var subcause;

var safetymeasure;

var subsafetymeasure;

var verification;

var validation;

var hl= new T!HazardLog;

for (t in h2.all)
{
     //t.println();

     //H1 -> C1-1 -> Sub cause ->  SM -> sub SM
     if(t.HazardId.isDefined())
     {

     //create the hazard

     hazard= t.createHazard();

     //create cause
     if (t.CauseId.isDefined())
     {
     cause= t.createCause();
     hazard.causes.add(cause);

      // cause defined, subcause not defined, subcause link to cause,
      //safety measure link to subcause, sub safety measure link to safety measure
      if (t.SubCauseId.isDefined())
     {
     subcause= t.createSubCause();
     cause.causes.add(subcause);



     // create control measure within the if state of cause
     if (t.SafetyMeasureId.isDefined())
     {
     safetymeasure= t.createSafetyMeasure();
     subcause.controlMeasure.add(safetymeasure);

      if (t.SubSafetyMeasureId.isDefined() )
     {
     subsafetymeasure= t.createSubSafetyMeasure();
     safetymeasure.controlMeasure.add(subsafetymeasure);



      // V & V link to subsafety measure
```

```
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
subsafetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
subsafetymeasure.validation.add(validation);
}
}

 // SubSafety Measure not defined, V & V link to safety measure
else {
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
safetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
safetymeasure.validation.add(validation);
}
}
}
}

  // cause defined, subcause not defined, safety measure link to cause
else  if (t.SafetyMeasureId.isDefined())
{
safetymeasure= t.createSafetyMeasure();
cause.controlMeasure.add(safetymeasure);


  if (t.SubSafetyMeasureId.isDefined())
{
subsafetymeasure= t.createSubSafetyMeasure();
safetymeasure.controlMeasure.add(subsafetymeasure);



 // V & V link to subsafety measure
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
subsafetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
subsafetymeasure.validation.add(validation);
```

```
 }
 }

     //SubSafety Measure not defined, V & V link to safety measure
else {
  if  (t.VerificationId.isDefined())
 {
 verification = t.createVerification();
 safetymeasure.verificationofEffectiveness.add(verification);
 }

 // create validation within the if state of safety measure
  if  (t.ValidationId.isDefined())
 {
 validation = createValidation();
 safetymeasure.validation.add(validation);
 }
 }
 }
 }
 }


 else
 // for Hazard not defined, but cause is defined, e.g. C1-2
 //repeat line 33-139
if (t.CauseId.isDefined())
 {
 cause= t.createCause();
 hazard.causes.add(cause);

  // cause defined, subcause not defined, subcause link to cause,
  //safety measure link to subcause, sub safety measure link to safety measure
  if (t.SubCauseId.isDefined())
 {
 subcause= t.createSubCause();
 cause.causes.add(subcause);
 // create control measure within the if state of cause
 if (t.SafetyMeasureId.isDefined())
 {
 safetymeasure= t.createSafetyMeasure();
 subcause.controlMeasure.add(safetymeasure);

  if (t.SubSafetyMeasureId.isDefined() )
 {
 subsafetymeasure= t.createSubSafetyMeasure();
 safetymeasure.controlMeasure.add(subsafetymeasure);



  // V & V link to subsafety measure
  if  (t.VerificationId.isDefined())
 {
 verification = t.createVerification();
 subsafetymeasure.verificationofEffectiveness.add(verification);
 }

 // create validation within the if state of safety measure
```

```
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
subsafetymeasure.validation.add(validation);
}
}

 // SubSafety Measure not defined, V & V link to safety measure
else {
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
safetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
safetymeasure.validation.add(validation);
}
}
}
}

  // cause defined, subcause not defined, safety measure link to cause
else  if (t.SafetyMeasureId.isDefined())
{
safetymeasure= t.createSafetyMeasure();
cause.controlMeasure.add(safetymeasure);


  if (t.SubSafetyMeasureId.isDefined())
{
subsafetymeasure= t.createSubSafetyMeasure();
safetymeasure.controlMeasure.add(subsafetymeasure);

 // V & V link to subsafety measure
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
subsafetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
subsafetymeasure.validation.add(validation);
}
}

    //SubSafety Measure not defined, V & V link to safety measure
else {
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
safetymeasure.verificationofEffectiveness.add(verification);
```

```
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
safetymeasure.validation.add(validation);
}
}
}
}

else
////for hazard not defined, cause is not defined, but subcasue is defined,
//repeat line 40-91

 if (t.SubCauseId.isDefined())
{
subcause= t.createSubCause();
cause.causes.add(subcause);



// create control measure within the if state of cause
if (t.SafetyMeasureId.isDefined())
{
safetymeasure= t.createSafetyMeasure();
subcause.controlMeasure.add(safetymeasure);

 if (t.SubSafetyMeasureId.isDefined())
{
subsafetymeasure= t.createSubSafetyMeasure();
safetymeasure.controlMeasure.add(subsafetymeasure);



 // V & V link to subsafety measure
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
subsafetymeasure.verificationofEffectiveness.add(verification);
}

// create validation within the if state of safety measure
 if  (t.ValidationId.isDefined())
{
validation = createValidation();
subsafetymeasure.validation.add(validation);
}
}

 // SubSafety Measure not defined, V & V link to safety measure
else {
 if  (t.VerificationId.isDefined())
{
verification = t.createVerification();
safetymeasure.verificationofEffectiveness.add(verification);
}
```

```
     // create validation within the if state of safety measure
      if  (t.ValidationId.isDefined())
     {
     validation = createValidation();
     safetymeasure.validation.add(validation);
     }
     }
     }
     }
     else
     //for the row where hazard not defined, cause and subcause is not defined,
but safety measure is defined, e.g. M1-1-2
     //repeat line 48-90, but add two branches as cause and subcause both defined,
or only cause  defined
  if (t.SafetyMeasureId.isDefined())
     {
     //cause and subcause both defined, safety measure link to subcasue
     if(cause.isDefined() and cause.causes.isDefined()){
     safetymeasure= t.createSafetyMeasure();
     subcause.controlMeasure.add(safetymeasure);

      if (t.SubSafetyMeasureId.isDefined())
     {
     subsafetymeasure= t.createSubSafetyMeasure();
     safetymeasure.controlMeasure.add(subsafetymeasure);

      // V & V link to subsafety measure
      if  (t.VerificationId.isDefined())
     {
     verification = t.createVerification();
     subsafetymeasure.verificationofEffectiveness.add(verification);
     }

     // create validation within the if state of safety measure
      if  (t.ValidationId.isDefined())
     {
     validation = createValidation();
     subsafetymeasure.validation.add(validation);
     }
     }

      // SubSafety Measure not defined, V & V link to safety measure
    else {
      if  (t.VerificationId.isDefined())
     {
     verification = t.createVerification();
     safetymeasure.verificationofEffectiveness.add(verification);
     }

     // create validation within the if state of safety measure
      if  (t.ValidationId.isDefined())
     {
     validation = createValidation();
     safetymeasure.validation.add(validation);
     }
     }
     }
```

```
      // cause is defined, not sub cause defined, safety measure link to cause
      else if(cause.isDefined()){
       safetymeasure= t.createSafetyMeasure();
      cause.controlMeasure.add(safetymeasure);

       if (t.SubSafetyMeasureId.isDefined())
      {
      subsafetymeasure= t.createSubSafetyMeasure();
      safetymeasure.controlMeasure.add(subsafetymeasure);
       // V & V link to subsafety measure
       if  (t.VerificationId.isDefined())
      {
      verification = t.createVerification();
      subsafetymeasure.verificationofEffectiveness.add(verification);
      }

      // create validation within the if state of safety measure
       if  (t.ValidationId.isDefined())
      {
      validation = createValidation();
      subsafetymeasure.validation.add(validation);
      }
      }

       // SubSafety Measure not defined, V & V link to safety measure
     else {
       if  (t.VerificationId.isDefined())
      {
      verification = t.createVerification();
      safetymeasure.verificationofEffectiveness.add(verification);
      }

      // create validation within the if state of safety measure
       if  (t.ValidationId.isDefined())
      {
      validation = createValidation();
      safetymeasure.validation.add(validation);
      }
      }
      }
      }
      else
      // for the row where hazard not defined, cause is not defined,safety measure
is not defined, but sub safety measure is defined
     // one branches: sub safety measure link to safety measure
     // repeat line
      if (t.SubSafetyMeasureId.isDefined())
      {
      subsafetymeasure= t.createSubSafetyMeasure();
      safetymeasure.controlMeasure.add(subsafetymeasure);

       // V & V link to subsafety measure
       if  (t.VerificationId.isDefined())
      {
      verification = t.createVerification();
      subsafetymeasure.verificationofEffectiveness.add(verification);
      }
```

```
     // create validation within the if state of safety measure
      if  (t.ValidationId.isDefined())
     {
     validation = createValidation();
     subsafetymeasure.validation.add(validation);
     }
     }
     else
     // for the row where hazard not defined, cause is not defined,safety measure
is not defined, but verification is defined, e.g. V1-1-1-2
    // two branches: both safety measure and sub SM are defined, V&V link to sub,
or only safety measure defined, V&V link to safety measure
     if (safetymeasure.isDefined() and safetymeasure.controlMeasure.isDefined()){
     if(t.VerificationId.isDefined())
     {
     verification = t.createVerification();
     subsafetymeasure.verificationofEffectiveness.add(verification);
     }

    // for hazard not defined, cause is not defined,safety measure is not defined,
but validation is defined, e.g. V1-1-1-1-2
      if  (t.ValidationId.isDefined())
     {
       validation = createValidation();
     subsafetymeasure.validation.add(validation);
     }
     }

     else if (safetymeasure.isDefined())
     {
     if(t.VerificationId.isDefined())
     {
     verification = t.createVerification();
     subsafetymeasure.verificationofEffectiveness.add(verification);
     }

    // for hazard not defined, cause is not defined,safety measure is not defined,
but validation is defined, e.g. V1-1-1-1-2
      if  (t.ValidationId.isDefined())
     {
       validation = createValidation();
     subsafetymeasure.validation.add(validation);
     }
     }


     hl.hazards.add(hazard);
     hazard.causes.println();
     }

   operation h2 createHazard() : T!HazardousSituation{
 var h = new T!HazardousSituation;
     h.number = self.HazardId;
     h.description = self.HazardDescription;
     return h;
}
```

```
operation h2 createCause(): T!Cause{
  var c = new T!Cause;
     c.number = self.CauseId;
     c.description = self.CauseDescription;
    // h.causes.add(c);
     return c;
   }

operation h2 createSubCause(): T!Cause{
  var c = new T!Cause;
     c.number = self.SubCauseId;
     c.description = self.SubCauseDescription;
        return c;
   }

operation h2 createSafetyMeasure(): T!ControlMeasure{
  var m = new T!ControlMeasure;
     m.number = self.SafetyMeasureId;
     m.description = self.SafetyMeasureDescription;
     return m;
   }

operation h2 createSubSafetyMeasure(): T!ControlMeasure{
  var m = new T!ControlMeasure;
     m.number = self.SubSafetyMeasureId;
     m.description = self.SubSafetyMeasureDescription;
     return m;
   }

operation h2 createVerification(): T!VerificationofEffectiveness{
var vr = new T!VerificationofEffectiveness;
     vr.number = self.VerificationId;
     vr.description = self.VerificationDescription;
     return vr;
}

operation h2 createValidation(): T!Validation{
var vl = new T!Validation;
     vl.number = self.ValidationId;
     vl.description = self.ValidationDescription;
     return vl;
     }
```

Attachment 3 ETL code for transformation of hazard log model to SACM AC model

```
// added "Query" constraints in line 243 and 455, to remove the transformation for
safety measure including "Query"
//if ( m.controlMeasure.isDefined() and (not
m.controlMeasure.startstWith("Query")))// so inference.target = safety measure
// inference.reasoning = argur over sub safety measure
// but define inference.source = queryclain in line 301-312, and line 510-521, so
this queryclaim with a claim.name= Query can be queried in OEL later
//and will be redefined in the RC2AC.eol


rule hazardlogToArgumentPackage
transform a : S!HazardLog
to b : T!ArgumentPackage
{
}
@lazy
rule causeToClaim
transform a : S!Cause
to b : T!Claim
{
var name = new T!LangString;
b.name = name;
b.name.lang = "English";
b.name.content ="Claim for " + a.number ;

var description = new T!Description;
var content = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content ="Cause " + a.number + " {" +  a.description +" } shall be
avoided." ;

b.description = description;
b.description.content = content;
// content.value is Collection
b.description.content.value.add(value);
}

@lazy
rule safetymeasureToClaim
transform a : S!ControlMeasure
to b : T!Claim
{
var name = new T!LangString;
b.name = name;
b.name.lang = "English";
b.name.content ="Claim for " + a.number ;

var description = new T!Description;
var content = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Safety measure " + a.number + " {" +  a.description +" } shall be
implemented." ;

b.description = description;
b.description.content = content;
```

```
// content.value is Collection
b.description.content.value.add(value);
}

@lazy
rule verificationToClaim
transform a : S!VerificationofEffectiveness
to b : T!Claim
{
var name = new T!LangString;
b.name = name;
b.name.lang = "English";
b.name.content ="Claim for " + a.number ;

var description = new T!Description;
var content = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Safety measure shall be verified by {"  +  a.description + " }";

b.description = description;
b.description.content = content;
// content.value is Collection
b.description.content.value.add(value);
}


@lazy
rule validationToClaim
transform a : S!Validation
to b : T!Claim
{
var name = new T!LangString;
b.name = name;
b.name.lang = "English";
b.name.content ="Claim for " + a.number ;

var description = new T!Description;
var content = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Safety measure shall be validated by {"  +  a.description + " }";

b.description = description;
b.description.content = content;
// content.value is Collection
b.description.content.value.add(value);
}



rule hazardToClaim
transform a : S!HazardousSituation
to b : T!Claim
{
var name = new T!LangString;
b.name = name;
b.name.lang = "English";
```

```
b.name.content ="Claim for " + a.number ;

var description = new T!Description;
var content = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Hazard " + a.number + " {" +  a.description + "} shall be
avoided." ;

b.description = description;
b.description.content = content;
// content.value is Collection
b.description.content.value.add(value);


if(a.causes.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = claim = transition,i.e. upper claim
inference.target.add(b);
" ".println();
inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argument over identified causes.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();



//inferenec.source = causes
for (c in a.causes)
{
//c.number.println();
inference.source.add(c.equivalent());

//link subcauses to cause
//this "if" cover line 154- 364
if(c.causes.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = claim = transition,i.e. upper claim
inference.target.add(c.equivalent());
" ".println();
inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning
```

```
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argument over identified subcauses.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();


for (subcause in c.causes)
{
inference.source.add(subcause.equivalent());

//then to link safety measure(source) to subcasue(target)
if(subcause.controlMeasure.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = cause
inference.target.add(subcause.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified control measure
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over identified control measures.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

//inferenec.source = control measure
for (m in subcause.controlMeasure)
{
//c.number.println();
inference.source.add(m.equivalent());


//sub measure to measure
if(m.controlMeasure.notEmpty() )
{
var inference = new T!AssertedInference;

//inference.target = cause
inference.target.add(m.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();
```

```
//strategy = arugmentreasoning: Argue over identified control measure
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over identified subcontrol measures.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

for(submeasure in m.controlMeasure)
{

if(not submeasure.description.startsWith("Query"))
{
inference.source.add(submeasure.equivalent());

// then link sub measure to V & V
if(submeasure.verificationofEffectiveness.notEmpty() or
submeasure.validation.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = control measure
inference.target.add(submeasure.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified V & V
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over V & V.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

//inferenec.source = Verification
if(submeasure.verificationofEffectiveness.notEmpty())
{
for (vr in submeasure.verificationofEffectiveness)
{
//c.number.println();
inference.source.add(vr.equivalent());
}
}
```

```
//inferenec.source = validation
if(submeasure.validation.notEmpty())
{
for (vl in submeasure.validation)
{
//c.number.println();
inference.source.add(vl.equivalent());
}
}

inference.source.description.content.value.first().content.first().println();
}

}

else
// here to generate a special claim with name "Query" as source of inference
// and in the next step in RC2AC.eol, EOL needs to find this inference with source
as "Query"
//and clear this source , then redefine source as RC states claims
{
var queryclaim = new T!Claim;
var name = new T!LangString;
queryclaim.name = name;
queryclaim.name.lang = "English";
queryclaim.name.content ="Query";
inference.source.add(queryclaim);
}
}
}

else
// safety measure defined, sub measure not defined
//link safety measure to VV
{
if(m.verificationofEffectiveness.notEmpty() or m.validation.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = control measure
inference.target.add(m.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified V & V
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over V & V.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
```

```
"".println();

//inferenec.source = Verification
if(m.verificationofEffectiveness.notEmpty())
{
for (vr in m.verificationofEffectiveness)
{
//c.number.println();
inference.source.add(vr.equivalent());
}
}
//inferenec.source = validation
if(m.validation.notEmpty())
{
for (vl in m.validation)
{
//c.number.println();
inference.source.add(vl.equivalent());
}
}
inference.source.description.content.value.first().content.first().println();
}
}
}
}
}
}


// the 2nd branch opposite to line 154 if(c.causes.notEmpty())
//here is the case if(c.causes.Empty()), i.e. no subcauses
// then link cause to safety measure
// this if scope line 191-277
else
if(c.controlMeasure.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = cause
inference.target.add(c.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified control measure
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over identified control measures.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();
```

```
//below line 408-558 repeat line 216-360
//but change subcause to cause
for (m in c.controlMeasure)
{
//c.number.println();
inference.source.add(m.equivalent());


//sub measure to measure
if(m.controlMeasure.notEmpty() )
{
var inference = new T!AssertedInference;

//inference.target = cause
inference.target.add(m.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified control measure
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over identified subcontrol measures.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

for(submeasure in m.controlMeausre)
{
if(not submeasure.description.startsWith("Query"))
{
inference.source.add(submeasure.equivalent());

// then link sub measure to V & V
if(submeasure.verificationofEffectiveness.notEmpty() or
submeasure.validation.notEmpty())
{
var inference = new T!AssertedInference;

//inference.target = control measure
inference.target.add(submeasure.equivalent());
" ".println();

inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified V & V
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
```

```
value.lang = "English";
value.content = "Strategy: Argue over V & V.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

//inferenec.source = Verification
if(submeasure.verificationofEffectiveness.notEmpty())
{
for (vr in submeasure.verificationofEffectiveness)
{
//c.number.println();
inference.source.add(vr.equivalent());
}
}

//inferenec.source = validation
if(submeasure.validation.notEmpty())
{
for (vl in submeasure.validation)
{
//c.number.println();
inference.source.add(vl.equivalent());
}
}

inference.source.description.content.value.first().content.first().println();
}

}

else
// here to generate a special claim with name "Query" as source of inference
// and in the next step in RC2AC.eol, EOL needs to find this inference with source
as "Query"
//and clear this source , then redefine source as RC states claims
{
var queryclaim = new T!Claim;
var name = new T!LangString;
queryclaim.name = name;
queryclaim.name.lang = "English";
queryclaim.name.content ="Query";
inference.source.add(queryclaim);
}
}
}


else
// safety measure defined, sub measure not defined
//link safety measure to VV
{
if(m.verificationofEffectiveness.notEmpty() or m.validation.notEmpty())
{
var inference = new T!AssertedInference;
```

```
//inference.target = control measure
inference.target.add(m.equivalent());
" ".println();
inference.target.first().name.content.print();
"- ".print();
inference.target.first().description.content.value.content.first().println();


//strategy = arugmentreasoning: Argue over identified V & V
var reasoning = new T!ArgumentReasoning;
var reasoningcontent = new T!MultiLangString;
var value = new T!LangString;
value.lang = "English";
value.content = "Strategy: Argue over V & V.";

inference.reasoning = reasoning;
inference.reasoning.content = reasoningcontent;
inference.reasoning.content.value.add(value);
inference.reasoning.content.value.first().content.println();
"".println();

//inferenec.source = Verification
if(m.verificationofEffectiveness.notEmpty())
{
for (vr in m.verificationofEffectiveness)
{
//c.number.println();
inference.source.add(vr.equivalent());
}
}


//inferenec.source = validation
if(m.validation.notEmpty())
{
for (vl in m.validation)
{
//c.number.println();
inference.source.add(vl.equivalent());
}
}

inference.source.description.content.value.first().content.first().println();
}
}
}
}
}
}
}
```

Attachment 4 EOL code for AC model generation from system models

```
for(inference in
HAC!AssertedInference.allInstances.select(i|i.source.first().name.content =
"Query"))
{
//inference.target.first().description.content.value.content.first().println();
inference.source.clear();

for ( s in S!State.allInstances().select(s|s.name != "OCM"))
{
//create the claim for this state
var c= new HAC!Claim;

var name = new HAC!LangString;
c.name = name;
c.name.lang = "English";
c.name.content ="Subclaim for State " + s.name;
//c.name.content.println();

var description = new HAC!Description;
var content = new HAC!MultiLangString;
var value = new HAC!LangString;
value.lang = "English";
value.content = s.name + " should have an outgoing transition whose trigger is
reqOCM, and target is OCM";
c.description = description;
c.description.content = content;
// content.value is Collection
c.description.content.value.add(value);

//here to link the claim generated in this eol to the inference generated in
hazardEMF2SACM.etl
//the bridge is the "Query"
inference.source.add(c);

//to verify if this state meets the claim, create evidence. evidence =
ArtifaceReference

var trName;
for (t in S!Transition.allInstances.select(t| t.source.name = s.name and
t.target.name = "OCM" and t.trigger.event.isDefined()= true and
t.trigger.event.name = "reqOCM"))
{
trName = t.name;
// ("Transition " + trName +" satisfies the claim.").println();

var ar = new HAC!ArtifactReference;
var name = new HAC!LangString;
ar.name = name;
ar.name.lang = "English";
ar.name.content = "Evidence for Claim of State " + s.name;


var description = new HAC!Description;
var content = new HAC!MultiLangString;
var value = new HAC!LangString;
value.lang = "English";
```

```
value.content = "Transition " + trName +" satisfies the claim.";
ar.description = description;
ar.description.content = content;
// content.value is Collection
ar.description.content.value.add(value);

ar.createAssertedEvidence(c);
}

if (trName.isUndefined())
{
//(" no transition exists for state " + s.name).println();
//then need to create the solution for fail
var ar = new HAC!ArtifactReference;
var name = new HAC!LangString;
ar.name = name;
ar.name.lang = "English";
ar.name.content = "Evidence for Claim of State " + s.name;


var description = new HAC!Description;
var content = new HAC!MultiLangString;
var value = new HAC!LangString;
value.lang = "English";

value.content = "No transition exists satisfing the claim." ;
ar.description = description;
ar.description.content = content;
// content.value is Collection
ar.description.content.value.add(value);
ar.createAssertedEvidence(c);
}
}
}
 //S!Transition.allInstances.select(t| t.source.name = s.name and t.target.name =
"OCM" and t.trigger).isDefined())

//S!Transition.allInstances.select(t| t.source.name = s.name and t.target.name =
"OCM"  and t.trigger.event.name = "reqOCM").isDefined().println();
operation HAC!ArtifactReference createAssertedEvidence(a: HAC!Claim){
//build relationship between claim and evidence, relationship = Asserted Evidence
var evidencerelation = new HAC!AssertedEvidence;

//inference.target = hazard claim
evidencerelation.target.add(a);
" ".println();
//inference.target.first().name.content.print();
//" ".print();
evidencerelation.target.first().description.content.value.content.first().println(
);

evidencerelation.source.add(self);
//evidencerelation.source.first().name.content.println();
evidencerelation.source.first().description.content.value.content.first().println(
);
}
```