You are an assistant who assist in developing an assurance case in a tree structure using Goal Structuring Notation (GSN) based on an existing assurance case pattern. Your role is to instantiate an assurance case pattern to create an assurance case. I will provide you with context information on assurance case and assurance case pattern. The context information for assurance case begins with the delimiter *“@Context\_AC”* and ends with the delimiter *“@End\_Context\_AC”* while the context information for assurance case pattern begins with the delimiter *"@Context\_ACP”* and ends with the delimiter *“@End\_Context\_ACP”*

*@Context\_AC*

An assurance case, such as a safety case or security case, can be represented using Goal Structuring Notation (GSN), a visual representation that presents the elements of an assurance case in a tree structure. The main elements of a GSN assurance case include Goals, Strategies, Solutions (evidence), Contexts, Assumptions, and Justifications.

Additionally, an assurance case in GSN may include an undeveloped element decorator, represented as a hollow diamond placed at the bottom center of a goal or strategy element. This indicates that a particular line of argument for the goal or strategy has not been fully developed and needs to be further developed.

I will explain each element of an assurance case in GSN so you can generate it efficiently.

1. Goal – A goal is represented by a rectangle and denoted as G. It represents the claims made in the argument. Goals should contain only claims. For the top-level claim, it should contain the most fundamental objective of the entire assurance case.
2. Strategy – A strategy is represented by a parallelogram and denoted as S. It describes the reasoning that connects the parent goals and their supporting goals. A Strategy should only summarize the argument approach. The text in a strategy element is usually preceded by phrases such as “Argument by appeal to…”, “Argument by …”, “Argument across …” etc.
3. Solution – A solution is represented by a circle and denoted as Sn. A solution element makes no claims but are simply references to evidence that provides support to a claim.
4. Context (Rounded rectangles) – In GSN, context is represented by a rounded rectangle and denoted as C. The context element provides additional background information for an argument and the scope for a goal or strategy within an assurance case.
5. Assumption – An assumption element is represented by an oval with the letter ‘A’ at the top- or bottom-right. It presents an intentionally unsubstantiated statement accepted as true within an assurance case. It is denoted by A
6. Justification (Ovals) – A justification element is represented by an oval with the letter ‘J’ at the top- or bottom-right. It presents a statement of reasoning or rationale within an assurance case. It is denoted by J.

*@End\_Context\_AC*

*@Context\_ACP*

Assurance case patterns in GSN (Goal Structuring Notation) are templates that can be re-used to create an assurance case. Assurance case patterns encapsulate common structures of argumentation that have been found effective for addressing recurrent safety, reliability, or security concerns. An assurance case pattern can be instantiated to develop an assurance case by replacing generic information in placeholder decorator with concrete or system specific information.

To represent assurance case patterns in GSN format, additional decorators have been provided to support assurance case patterns. These additional decorators are used together with the elements of an assurance case to represent assurance case pattern. I will explain each additional decorator below to support assurance case pattern in GSN.

1. Uninstantiated - This decorator denotes that a GSN element remains to be instantiated, i.e. at some later stage, the generic information in placeholders within a GSN element needs to be replaced (instantiated) with a more concrete or system specific information. This decorator can be applied to any GSN element.
2. Uninstantiated and Undeveloped – Both decorators of undeveloped and uninstantiated are overlaid to form this decorator. This decorator denotes that a GSN element requires both further development and instantiation.
3. Placeholders – This is represented as curly brackets “{}” within the description of an element to allow for customization. The placeholder "{}" should be directly inserted within the description of elements for which the predicate "HasPlaceholder (X)" returns true. The placeholder "{}" can sometimes be empty or contain generic information that will need to be replaced when an assurance case pattern is instantiated.
4. Choice - A solid diamond is the symbol for Choice. A GSN choice can be used to denote alternatives in satisfying a relationship or represent alternative lines of argument used to support a particular goal.
5. Multiplicity - A solid ball is the symbol for multiple instantiations. It represents generalized n-ary relationships between GSN elements. Multiplicity symbols can be used to describe how many instances of one element-type relate to another element.
6. Optionality - A hollow ball indicates ‘optional’ instantiation. Optionality represents optional and alternative relationships between GSN elements.

The following steps is used to create an assurance case from an Assurance cases pattern.

1. Create the assurance case using only elements and decorators defined for assurance cases.
2. Remove all additional assurance case pattern decorators such as (Uninstantiated, Placeholders, Choice, Multiplicity, Optionality, and the combined Uninstantiated and Undeveloped decorator)
3. Remove the placeholder symbol "{}" and replace all generic information in placeholders “{}” with system specific or concrete information.

*@End\_Context\_ACP*

We have defined the following predicate rules for the elements and decorator used in an assurance case to ease understanding of an assurance case. The predicate rules for the elements and decorator of an assurance case begins with the delimiter *“@Predicate\_AC”* and ends with the delimiter *"@End\_Predicate\_AC”*

*@Predicate\_AC*

1. Goal(G): True if G is a goal within the assurance case. This predicate is represented as Goal (ID, Description) where ID is the unique identifier for the goal, and description is the textual information of the goal.
2. Strategy(S): True if S is a strategy within the assurance case. This predicate is represented as Strategy (ID, Description) where ID is the unique identifier for the strategy and description is the textual information of the Strategy.
3. Solution (Sn): True if Sn is evidence within the assurance case. This predicate is represented as Solution (ID, Description) where ID is the unique identifier for the evidence or solution and description is the textual information of the evidence.
4. Context(C): True if C is a context within the assurance case. This predicate is represented as Context (ID, Description) where ID is the unique identifier for the context and description is the textual information of the context.
5. Assumption (A): True if A is an assumption within the assurance case. This predicate is represented as Assumption (ID, Description) where ID is the unique identifier for the assumption and description is the textual information of the assumption.
6. Justification (J): True if J is a justification within the assurance case. This predicate is represented as Justification (ID, Description) where ID is the unique identifier for the justification and description is the textual information of the justification.
7. Undeveloped(X): True if X is either a Goal(G) or Strategy(S) marked as undeveloped. This predicate is represented as Undeveloped(X), where X can be either a goal or strategy.

*@End\_Predicate\_AC*

We have defined the following predicate rules for the additional decorators used to support assurance case patterns to ease understanding. The predicate rules for the additional decorators to support assurance case pattern begins with the delimiter *“@Predicate\_ACP”* and ends with the delimiter *"@End\_Predicate\_ACP”*

*@Predicate\_ACP*

1. Uninstantiated (X): True if element X (can be any GSN element) is marked as uninstantiated.
2. UndevelopStantiated (X): True if element X is either a Goal(G) or Strategy(S) and is marked both as uninstantiated and undeveloped.
3. HasPlaceholder (X): True if element ‘X’ (can be any GSN element) contains a placeholder ‘{}’ within its description that needs instantiation.
4. HasChoice (X, [Y], Label): True if an element ‘X’ (either a Goal(G) or Strategy(S)) can be supported by selecting among any number of elements in [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between ‘X’ and ‘Y’. A label is of the general form “m of n” (e.g. a label given as “1 of 3” implies an element ‘X’ can be supported by any one of three possible supporting elements in [Y])
5. HasMultiplicity (X, [Y], Label): True if multiple instances of an element X (either a Goal(G) or Strategy(S)) relate to multiple instances of another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y. (i.e., how many instances of an element in X relates with how many instances of an element in [Y]. e.g. m of n implies m instances of an element in X must be supported by n instances of an element in Y)
6. IsOptional (X, [Y], Label): True if an element X (either a Goal(G) or Strategy(S)) can be optionally supported by another element [Y] (where Y can be any GSN element) according to the cardinality specified by an optional Label. The label specifies the cardinality of the relationship between X and Y. (i.e. an instance of an element in X may be supported by another instance of an element in [Y], but it is not required)

*@End\_Predicate\_ACP*

To represent an assurance case or assurance case pattern in GSN is equivalent to depicting in a hierarchical tree structure. To achieve this hierarchical tree structure, the below predicates have been defined to ease understanding of this structure. The predicate rules to support the structure of an assurance case or assurance case pattern begins with the delimiter *“@Predicate\_Structure”* and ends with the delimiter *“@End\_Predicate\_Structure”*

*@Predicate\_Structure*

1. IncontextOf (X, [N], D): True if element X at depth D has a neighbour [N] to the left or right at depth D, where ‘[N]’ can be an Assumption (A), Justification (J), or Context (C), ‘X’ can be a Goal (G), or Strategy (S) and ‘D’ represents the height or depth of the goal or strategy element and its neighbours in the GSN hierarchical structure.
2. SupportedBy (X, [C], D): True if element X at depth D has children [C] directly below it, where [C] can include Goal (G), Strategy (S), or Solution (Sn) and ‘X’ can be a Goal (G), or Strategy (S).

* If X is Strategy (S), [C] can only be Goal (G).
* If X is Goal (G), [C] can be either Goal (G), Strategy(S), or Solution (Sn).

*@End\_Predicate\_Structure*

Now, I will provide you with an example of an assurance case pattern in its predicate form and the corresponding assurance case derived from this pattern so that you can understand the process of instantiating an assurance case pattern to create an assurance case.

For example, a security case pattern for threat identification for ACAS Xu (Airborne Collision Avoidance System Xu) and the derived security case from the pattern is given below. The security case pattern begins with the delimiter *"@Pattern"* and ends with the delimiter *"@End\_Pattern"* while the derived security case begins with the delimiter *"@Assurance\_case"* and ends with the delimiter *"@End\_Assurance\_case"*

*@Pattern*

Goal (G0, {System} satisfies security requirements)

Goal (G1, {System} satisfies the asset protection requirements)

Goal (G2, {System} satisfies secure development requirements)

Goal (G3, Asset protection requirements are met during the architecture design phase)

Goal (G4, Asset protection requirements are met during other phases)

Goal (G5, {System} architecture is protected against identified security threats (STs))

Goal (G6, {System} architecture is validated)

Goal (G0.X, {System} architecture is protected against STX)

Strategy (S0, Argue through asset protection and secure development requirements)

Strategy (S1, Argue through the different stages of the system development life cycle)

Strategy (S2, Argue through derivating security threats from SRs)

Strategy (S3, Argue over each security threat)

Context (C0, Description of {system})

Context (C1, SR are requirements about protecting the system from malicious entities)

Context (C2, Description of the {architecture})

Context (C3, Description of {system} architecture model)

Justification (J0, The argumentation is based on satisfaction of SRs)

Justification (J1, Detection and mitigation of threats fulfill SRs)

Assumption (A0, System SRS are complete, adequate, and consistent)

Assumption (A1, Asset inventory is established)

Assumption (A2, All relevant threats have been identified)

Assumption (A3, {System} architecture model is well defined in {formal method})

SupportedBy (G0, S0, 1)

SupportedBy (S0, [G1, G2], 2)

SupportedBy (G1, S1, 3)

SupportedBy (S1, [G3, G4], 4)

SupportedBy (G3, S2, 5)

SupportedBy (S2, [G5, G6], 6)

SupportedBy (G5, S3, 7)

SupportedBy (S3, G0.X, 8)

IncontextOf (G0, [C0, C1, J0, A0], 1)

IncontextOf (G1, A1, 3)

IncontextOf (G3, C2, 5)

IncontextOf (S2, J1, 6)

IncontextOf (G5, A2, 7)

IncontextOf (G6, [C3, A3], 7)

HasPlaceholder (G0)

HasPlaceholder (C0)

HasPlaceholder (G1)

HasPlaceholder (G2)

HasPlaceholder (C2)

HasPlaceholder (G5)

HasPlaceholder (G6)

HasPlaceholder (C3)

HasPlaceholder (A3)

HasPlaceholder (G0.X)

Uninstantiated (G0)

Uninstantiated (C0)

Uninstantiated (G1)

Uninstantiated (C2)

Uninstantiated (G5)

Uninstantiated (C3)

Uninstantiated (A3)

Undeveloped (G4)

UndevelopStantiated (G2)

UndevelopStantiated (G6)

UndevelopStantiated (G0.X)

*@End\_Pattern*

*@Assurance\_case*

G0: ACAS Xu satisfies security requirements

C0: Description of ACAS Xu

C1: SR are requirements about protecting the system from malicious entities

J0: The argumentation is based on satisfaction of SRs

A0: System SRs are complete, adequate, and consistent

S0: Argue through asset protection and secure development requirements

G1: ACAS Xu satisfies the asset protection requirements

A1: Asset inventory is established

S1: Argue through the different stages of the system development life cycle

G3: Asset protection requirements are met during the architecture design phase

C2: The architecture is based on components and connectors style. The communication paradigm is message-passing communication

S2: Argue through derivating security threats from SRs

J1: Detection and mitigation of threats fulfill SRs.

G5: ACAS Xu architecture is protected against identified security threats (STs)

A2: All relevant threats have been identified

S3: Argue over each security threat

G0.1: ACAS Xu architecture is protected against ST1 (undeveloped)

G0.2: ACAS Xu architecture is protected against ST2 (undeveloped)

G0.3: ACAS Xu architecture is protected against ST3 (undeveloped)

G6: ACAS Xu architecture is validated (undeveloped)

C3: ACAS Xu architecture formal model from work[24]

A3: ACAS Xu architecture model is well defined in Alloy

G4: Asset protection requirements are met during other phases (undeveloped)

G2: ACAS Xu satisfies secure development requirements (undeveloped)

*@End\_Assurance\_case*

Now, I would provide you with domain information about the Instant messaging (IM) software for which you would create a security case from a given security case pattern. The domain information begins with the delimiter *“@Domain\_Information”* and ends with the delimiter *“@End\_Domain\_Information”.*

*@Domain\_Information*

Instant messaging (IM) software is typically used for information exchange, with the data within the software forming the basis of user interaction.

Software can be characterized by its independent behavioral features, known as its internal structure, as well as by its interactive relationships with external components, referred to as the external manifestation (EM). The EM encompasses the overall interaction of software with the outside world, including the set of external environment entities, the interaction set between these entities and the software, and the direction of these interactions.

The internal structure (IS) of software focuses on the internal functional processes and data transmission within the software. This includes the set of software functional processes, data storage, and internal interaction sets.

Software can be considered a unification of the external manifestation and the internal structure. When attacking software, an attacker can target either the EM or the IS. Therefore, ensuring software security requires full protection of the core elements of both aspects. These core elements are summarized in the PEDD classification, which includes four types of assets: Process, Environment Entity, Data Exchange, and Data Storage. The Process, Data Storage, and Data Exchange are associated with the internal structure of the software, while the Environment Entity and Data Exchange are linked to the external manifestation of the software.

In IM software, the critical assets are:

* Registration information
* Registration process
* User account information
* Authentication information
* Login process
* Chat logs.

To ensure that IM software is acceptably secure, it is essential to protect all critical assets by establishing a security goal for each one.

*@End\_Domain\_Information*