Managing the Evolution of Dependability Cases for Systems of Systems

Georgios Despotou

Submitted for the degree of PhD

Department of Computer Science

University of York

April 2007

In memory of my grandfather

"Strength accompanied by wisdom has benefited the one who possesses it, but without wisdom it harms more its possessors, and while it laureates the bodies of those who cultivate it, yet it obscures the nurture of the soul."

Isocrates

Abstract

Dependability is a composite property consisting of attributes such as reliability, availability, safety and security. The achievement of these attributes is often essential for the operational success of systems undertaking critical and complex tasks. Assurance that the final system will demonstrate the required dependability qualities, can be crucial to the acceptance of the system into service.

Safety cases are a well established concept used to establish assurance about the safety properties of a system. However, safety cases focus only on one attribute of dependability. The principles and processes of creating an integrated *dependability* case – that assures *all* aspects of dependable system behaviour – are less well understood. A number of challenges are faced when attempting to support dependability case development. These include the systematic elicitation of dependability goals, the management and justification of trade-offs, and the evolution of multi-attribute arguments in step with the design process.

This thesis addresses these challenges by defining a rigorous framework, accompanied by a set of methods, for establishing dependability cases. Firstly, a method for eliciting dependability requirements is defined by extending existing safety deviational analysis techniques. Secondly, a method for systematically identifying and managing justified trade-offs is presented. Thirdly, the thesis describes the co-evolution of dependability case arguments alongside system development – using a dependability case architecture that corresponds to system structures. Finally, the thesis unifies these contributions by defining a metamodel that captures and interrelates the concepts underlying the proposed methods. Evaluation of the work is presented by means of peer review, pilot studies and industrial examples.

Contents

List of Figures

List of Tables

Acknowledgements

This research is carried out under the High Integrity Real Time Systems Defence and Aerospace Research Partnership (HIRTS DARP), funded by the MoD, DTI and EPSRC. The members of the HIRTS DARP are BAE SYSTEMS, Rolls-Royce plc, QinetiQ and the University of York.

I would like to thank my supervisor, Dr. Tim Kelly, whose advice, encouragement and patience have been invaluable.

I would also like to thank my friend Dimitrios Kolovos for his expert input on modelling and metamodelling.

Finally I would like to thank the DARP Working Group members Colin o'Halloran and Andy Ward, Glen Callow and Jane Fenn for their comments and feedback, and in particular Jane Fenn, Glenn Callow and BAE SYSTEMS for providing this research with an appropriate industrial case study.

Author's Declaration

Some of the material presented in this thesis has previously been published as conference, workshop and journal papers:

(presented in reverse chronological order)

- G. Despotou, T. Kelly. Design and Development of Dependability Case Architecture during System Development. In proceedings of the 25th International System Safety Conference (ISSC), Baltimore, MD USA. August 2007.

- G. Despotou, T. Kelly. An Argument Based Approach for Assessing Design Alternatives and Facilitating Trade-offs in Critical Systems. Journal of System Safety Vol.43 No.2 March-April 2007, System Safety Society. ISSN-0743-8826

- G. Despotou, D. Kolovos, R. Paige, F. Polack, T. Kelly. Towards a Metamodel for Dependability Cases, presented at the Object Management Group (OMG) 1st Software Assurance Workshop, Washington D.C. USA, March 2007.

- G. Despotou, T. Kelly. An Argument Based Approach for Assessing Design Alternatives and Facilitating Trade-offs in Critical Systems. In proceedings of the 24th International System Safety Conference (ISSC), Albuquerque, NM USA. Proceedings by the System Safety Society, August 2006.

- G. Despotou, T. Kelly. Extending Safety Deviation Analysis Techniques to Elicit Flexible Dependability Requirements. In proceedings of the 1st IET International Conference on System Safety Engineering, London, UK, June 2006. Proceedings by the Institute of Engineering and Technology (IET). ISSN 0537-9989.

- G. Despotou, M. Hall-May, T. Kelly. Eliciting Safety Policy and Balancing with Operational Fitness in Systems of Systems. In proceedings of the 1st IEEE International Conference on Systems of Systems Engineering (SoSE), Los

Angeles, CA USA, April 2006. Proceedings by IEEE SMC. ISBN 1-4244-0188-7.

- G. Despotou, T. Kelly. The Need for Flexible Requirements in Dependable Systems. In proceedings of the 4th International Workshop on Requirements for High Assurance Systems (RHAS), published by the Software Engineering Institute (2005). 13th IEEE International Requirements Engineering Conference, Paris, France, August 2005.

- G. Despotou, J. McDermid, T. Kelly. Using Scenarios to Identify and Trade-off Dependability Objectives in Design. In proceedings of the 23rd International System Safety Conference (ISSC), San Diego, CA USA, proceedings by System Safety Society August 2005.

- R. Weaver, G. Despotou, T. Kelly, J. McDermid. Combining Software Evidence – Arguments and Assurance. Workshop in Realising Evidence Based Software Engineering (REBSE), 25th International Conference on Software Engineering, Saint Louis, MO USA, ACM 2005. ACM SIGSOFT Software Engineering Notes, Volume 30, Issue 4 (July 2005), ISBN: 1-59593-121-X.

- R. Alexander, M. Hall-May, G. Despotou, T. Kelly. Using Simulation to Evaluate Safety Policy for Systems of Systems. 2nd International Workshop on Safety and Security of Multi Agent Systems (SASEMAS), 4th International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, Netherlands, July 2005.

- G. Despotou, T. Kelly. Extending the Concept of Safety Cases to Address Dependability. In proceedings of the 22nd International System Safety Conference, Providence, RI USA, proceedings by System Safety Society 2004.

Except where stated, all the work contained within this thesis represents the original contribution of the author.

Intentionally Blank

Chapter 1

Introduction

Dependability is a composite property consisting of attributes such as reliability, availability, safety and security. The achievement of these attributes is often essential for the operational success of systems undertaking critical and complex tasks. Systems of Systems (SoS) is a term increasingly used to represent large complex systems consisting of many independent elements. Systems of Systems are designed to solve particularly complex and critical problems. Due to the criticality of the problems that the SoS solve, system stakeholders need to have confidence that the system will operate in an acceptable way. This entails examining the operational behaviour of the system from the perspectives of a number of attributes crucial to the success of the operation, such as performance, availability and safety. Dependability is an umbrella term used to encompass these attributes.

1.1 Systems of Systems

An example of an SoS can be found in the integration of the air traffic control regions providing automated air traffic management functions for both aircraft and flight controllers; another is the concept of Network Centric Warfare which entails individual platforms collaborating and sharing awareness to achieve the mission objectives more effectively.

1.1.1 Air Traffic Control (ATC)

Air traffic control consists of a set of services aiming to direct aircraft through airspace, overseeing adherence to separation limits between aircraft [1]. The airspace management functions provided by ATC need to cover all phases of a flight. ATC functions involve a number of geographically dispersed systems and users, such as radars, controllers' terminals, airport towers, and meteorological stations, which need to collaborate to achieve the safe and efficient passage of aircraft.

In particular, airspace consists of Flight Information Region (FIR) sectors over which an Area Control Centre is responsible for controlling all the flights. The FIR can be further divided into sectors. During take-off (and initial climb), landing (and initial approach) and whilst on the ground, aircraft are controlled by the local tower. When moving from one region to another, responsibility for an aircraft is handed over to the corresponding controllers. For example, shortly after take-off, the local tower will hand responsibility over to the appropriate FIR control centre. Although pilots have ultimate responsibility for the aircraft's safety, controllers may require an aircraft to change flight level (altitude), reduce or increase speed in order to maintain separation, or to request manoeuvres such as joining a holding pattern.

Controllers must be able to see information in real time. Precision of navigation data is another important requirement, which allows aircraft to cruise with reduced separation between them. This has already led to airspace with reduced minimum vertical separation between aircraft [2], increasing the capacity of an area for aircraft. The air traffic management system has to predict the trajectories of aircraft in airspace over time (i.e. 4D trajectories), search for any conflicts between aircraft, and help a flight to avoid areas with turbulence or bad weather.

Operation of the ATC should be both safe and efficient. Not being able to achieve either of these two attributes the system will not be fit to operate for its intended purpose. The ATC example was used throughout the research to better understand the challenges in acquiring confidence for this class of systems.

1.1.2 Network Centric Warfare (NCW)

The concept of NCW was introduced as a means of using computing and systems engineering technology to achieve more effective military missions. The US Department of Defense (DoD) in [3] suggests that "*Network Centric Warfare provides a valuable perspective for achieving success in a target-oriented warfare situation, where timely, relevant, accurate and precise information is required to automatically engage targets expeditiously with the most effective weapons and forces available*".

NCW involves integration and coordination between all battle, control and decision making systems, so that the appropriate tactics and resources are used to efficiently achieve mission goals. Each element consists of a battle platform that has a perception of its environment, which then is shared among other platforms [4]. Each element is capable of independent operation – making its own decisions in order to achieve mission objectives. Elements should share their view of the environment in order to enrich the overall intelligence 'picture'. Thereby one element may benefit from another element's knowledge allowing it to make more accurate and more confident decisions.

When a mission goal is defined the system has to plan the steps required to achieve the goal, the elements that will participate in carrying out these steps, along with the contribution and responsibility of each element. After receiving the goals and its tasked responsibilities, each element has to decide upon the resources that it needs, and to plan how it will acquire those efficiently. NCW is a paradigm vividly showing the need of achievement of dependability attributes. The safety of the participating units, security of sensitive information (e.g. communication codes) and availability when a unit is required to operate are three attributes of importance in NCW. Moreover there can be interaction between these attributes. For example in a NCW system the safety of the system's elements is closely dependent to the security of the system, as security vulnerabilities can be exploited by the enemy.

1.2 General Characteristics of Systems of Systems

The phrase "Systems of Systems (SoS)" has been introduced to describe systems such as those described in the previous section. Although there are variations depending on the particular domain in which the term System of Systems is defined, in this thesis the following essential characteristics have been defined in order to classify a system as System of Systems:

- **Overall objectives:** A System of Systems has a set of high level goals of interest to its stakeholders, such as provision of air traffic management (for ATC) or accomplishment of a mission (for NCW).

- **Complexity of problem:** SoS are typically used in problems of high complexity (e.g. aircraft route planning or targeting).

- **Multiple elements:** A System of Systems consists of many elements which are systems in their own right and can or have been developed independently from the SoS (e.g. a radar or an unmanned aircraft).

- **Autonomy:** Elements are able to make their own decisions with varying degrees of autonomy (e.g. autonomous vehicles).

- **Geographical dispersion:** SoS deployed in the real world often involve elements that are geographically dispersed and mobile – changing their position according to the overall SoS objectives.

- **Collaboration:** SoS elements need to collaborate offering some of their functions to achieve the overall SoS objectives (e.g. sharing of intelligence requires elements sensing and others analysing data).

- **Communication:** Collaboration between the SoS elements results in exchanges of information.

- **Elements are heterogeneous:** Elements have been developed independently from each other, potentially with different technologies and from different developers.

This thesis defines System of Systems in the following terms:

> *A System of Systems is an organised complex unity assembled from dispersed, highly cooperating, autonomous systems – each of which is capable of operating independently.*

1.3 Assurance of Operation

In section 1 we briefly described two examples of Systems of Systems, as well as their characteristics. Development and use of such systems can be very beneficial; however SoS exhibit emergent behaviour resulting from the interactions between components. This behaviour cannot be predicted from observation of any single component. To illustrate this problem, the following sections present a number of historical situations in which SoS did not operate as expected.

1.3.1 Examples of Accidents in NCW and ATC

Friendly Fire Accident in NCW

After the gulf war, on April 14th 1994, two F-15 fighter aircraft operated by the US Air Force shot down two Black Hawk transport helicopters operated by the US Army. The helicopters were flying in the no fly zone, and as a result of the accident, everyone onboard the helicopters was killed. This occurred despite the presence of the US Air Force Airborne Warning and Control System (AWACS) which was in control of coordinating NATO activity in the region. AWACS fitted aircraft are able to carry out airborne surveillance and battle management functions. Leveson cites [5] that as many as 130 mistakes could be identified as contributing factors to this accident. In this case, failure of individual elements to operate as expected had an impact on the overall SoS operation, resulting in compromising its safety levels. Alexander et al provide in [6] an analysis from the viewpoint of the SoS characteristics, and how individual failures propagated through the different SoS elements and contributed to the accident.

Air Traffic Control Accident

After an accident in 1978 in which two aircraft were involved in a mid-air collision in airspace over the airport from which they had just taken off, all aircraft were required to install a collision avoidance system – TCAS (Traffic Collision Avoidance System). TCAS continuously monitors the traffic around an aircraft enhancing the crew's perception of the environment. It detects and resolves situations in which two aircraft are on course for a collision.

Although the primary responsibility of resolving such conflicts lies with the air traffic controller, TCAS becomes essential when two aircraft are in such close proximity that

immediate action is necessary to avoid collision. In such situations, TCAS systems on both involved aircraft, after communicating with each other, issue resolution advisory messages. These messages ideally will result in opposing (vertical) manoeuvres increasing the aircraft separation. On the 1st of July 2002 a cargo Boeing 757 and a passenger Tu-154 aircraft collided inside Swiss airspace over Lake Constance near Überlingen [7]. Although the two aircraft were the only aircraft in that vicinity, a number of circumstances resulted in the controller failing to notice the conflict until the aircraft were in close proximity. The controller issued (with substantial delay) advice to the aircraft to change altitude. At that time TCAS had also issued resolution advisory messages, requesting the ascent of the Tu-154 and the descent of the B-757. One aircraft followed the controller's advice whereas the other followed TCAS. This resulted in the descent of both aircraft and – ultimately – a collision. The accident claimed 69 fatalities on board both aircraft.

The Problem of Multiple, Interacting, Causes

Analysing the accidents, one cannot conclusively pinpoint a single cause. In both cases there were many factors that contributed to the accidents. The safeguards that had been designed to contain the effects of possible failures failed to operate properly themselves. Furthermore one can also notice different types of failures. For example had the ATC controller completely failed to notice the conflict between the aircraft, the crews would only have had the (safe) advisory of the TCAS. Instead, the untimely response of the controller resulted in conflicting advice, which contributed to the accident.

1.3.2 Assurance as Obligatory Requirement

It is now commonplace that developers of safety critical systems are required to produce a corresponding safety case – communicating an argument, supported by evidence, that a system is acceptably safe to operate. Examples of application domains for such systems include the defence and railway sectors. In such domains the description of requirements for the safety case product, as well as the processes for development, are described in detail by the respective safety standards such as the U.K. Defence Standard 00-56 [8].

1.3.3 System Dependability Assurance

Over the years there has been significant research in the safety domain and in particular regarding safety cases. However safety is only one of a number of system attributes that are potentially of interest. Other system attributes can be crucial to the stakeholders of a system. For example, consider the case of NCW – conceived with the sole purpose of improving the effectiveness of force elements in the battlefield. In such a system, safety can be of similar importance with other attributes such as performance or availability. For example, a defensive system being unavailable in the presence of an enemy threat could be considered to be of utmost importance. The importance of assuring the achievement of dependability attributes (other than safety) is readily apparent in many Systems of Systems examples. Despite the fact that there are standards explicitly requiring assurance about safety and the maintainability and availability of a system, there are no standards explicitly asking for assurance about the overall dependable behaviour of a system.

1.4 Dependability

Whilst there is no overall consensus on the exact definition of dependability, many agree that it can be described as the "*the system's characteristic that justifies placing one's reliance on it*" [9], entailing such attributes as reliability, safety, security and maintainability. Prasad similarly defines Dependability as, "*... a variable sized vector of attributes describing overlapping desiderata, chosen subjectively, in accordance to the stakeholders' particular requirements*" [10]. Furthermore, Prasad highlighted that despite the fact that dependability attributes can be interrelated, they are not orthogonal to each other, and can be in conflict or in harmony. Overall, dependability is a composite system property consisting of a number of different heterogeneous attributes.

1.5 Dependability Cases

In this thesis the following definition of a dependability case has been adopted:

> *A dependability case is a clear, defensible, and traceable argument that a system is acceptably dependable to operate in a given operational context.*

- **Argument:** A dependability case must communicate an argument about the achievement of the dependability attributes of a system, providing assurance to the developers.

- **Context:** Context describes the system's intended operation; it is unrealistic to attempt to create a dependability case without capturing the envisioned operation and operational context of the system.

- **Dependable:** A dependability case should provide confidence in all dependability attributes that are of interest to the system's stakeholders.

- **Acceptable:** Achieving all the required attributes fully is a utopian goal. Stakeholders must trade-off the (sufficient) achievement of multiple competing attributes. Justification of this trade-off is an essential element of any dependability case.

- **Traceable:** It should be possible to clearly trace between the objectives, arguments, evidence and trade-offs of the dependability case. Such traceability enables systematic review and evaluation of the acceptability of the system design.

1.6 A Roadmap for Systems of Systems Dependability Cases

The research presented in this thesis was performed under the initiative of the Defence and Aerospace Research Partnership (DARP) for High Integrity Real-Time Systems (HIRTS) – in collaboration with BAE Systems, QinetiQ and Rolls-Royce plc. As part of the HIRTS DARP work the participating companies helped define the main research challenges in the field of assuring the dependable operation of Systems of Systems. Identification of the challenges took place during workshops, in which participants from the partner companies and the University of York discussed academic and industrial experience and the state of the art related to the subject. Part of the output of the workshops is discussed in [11]. A compiled list of challenges, focusing on

dependability cases, was created during these activities. The following challenges were identified:

- Multiple dependability attributes
- Allocation and apportionment of requirements
- Conflicting requirements
- Changing requirements
- Traceability
- Interaction of case and design
- Ownership of the dependability case

The identified issues are discussed in more detail in the following sections.

1.6.1 Multiple Dependability Attributes

Dependability is a multi-attribute system characteristic. Arguing about achievement of dependability will include references to achievement of its constituent attributes. There are several domains in which there are examples of the creation of arguments for individual dependability attributes. One such domain is safety. The practice of safety case development and acceptance is relatively mature, and extensively used both in military and civil industry. Maintainability and reliability cases are examples of 'cases' communicating arguments about dependability attributes. However their use is not as widespread as that of safety cases.

Although the concept of creating cases for the individual attributes of dependability is not new, 'simply integrating' all different attribute arguments will introduce challenges that are difficult to overcome. A dependability case will need to record the relationships between attributes as well as how they were affected by decisions taken during system development. For example, the *availability* of a protection function onboard an aircraft (e.g. availability of TCAS) also (positively) affects the *safety* of the aircraft. In contrast, availability and safety can also be at odds with each other. An example of such conflict can be found in the development and use of an aircraft's Minimum Equipment List (MEL). A MEL provides a detailed description of the minimum systems required to be

operational for an aircraft to be airworthy. For example, an aircraft may be allowed to be dispatched (for some time) with one processing unit inoperative. Hence, in principle, a reduction of safety (due to reduced redundancy) is acceptable, in order for the aircraft to perform its mission.

When designing to achieve multiple dependability requirements certain (design) decisions may introduce conflicts between the attributes, which will eventually result in trade-offs needing to be made. The dependability case should be able to capture the conflicts that occur during system development and the trade-off process – providing justification that the design decisions taken constitute the most optimal choice.

1.6.2 Allocation and Apportionment of Requirements

The operation of a System of Systems consists of the combined operation of each of its individual elements. Identification of requirements with regard to the operation of a System of Systems occurs by considering the envisioned operation of the SoS. For example, in NCW initially the system stakeholders will specify the overall objectives that the system is required to achieve, and not the individual contribution of each SoS element. Consequently, the development of a case initially takes place in the context of the overall concept of operation.

In terms of dependability, identification of the context includes identification of the criteria of acceptable operation with respect to the dependability attributes of interest to the stakeholders. The initial high level objectives constitute the stakeholders' dependability requirements that need to be addressed by the proposed system design. According to how the system is designed, the system elements can variably affect the achievement of the stakeholders' overall dependability goals. For example, the overall safety of a system may require certain system elements to achieve a particular reliability requirement.

Following the initial stages of the case development (which regard the SoS in the large), the system design process must consider apportionment of the overall SoS requirements to individual elements of the system. Consequently, the focus of the case will shift towards the individual elements of the SoS. Apportionment of requirements relies on a

very clear understanding of the contribution of each system to the overall SoS functionality. However, the characteristics of a System of Systems are such that do not always favour a clear apportionment of requirements. Collaboration between the elements of a SoS and dynamic reconfiguration of its operation, often result in emergent behaviour which cannot be revealed by merely analysing the behaviour of each element individually. The dependability case should be able to demonstrate assurance regarding the contribution of the underlying behaviour of SoS elements, in satisfying the overall dependability requirements of the SoS stakeholders.

1.6.3 Conflicting Requirements

Dependability attributes are heterogeneous and non-orthogonal to each other. They can variably be in conflict or in harmony with each other, according to the design of the system. The more complex the system and the larger its scale, the more unlikely it is for its requirements to be met without any conflicts arising between them. This is further exacerbated by the multiple attributes that a SoS needs to satisfy and their interrelationships.

For example, consider an ATC system, similar to that described earlier. Increasing the number of aircraft that an ATC (services) area can support is considered to be at odds with safety. A 'busy' sky increases the safety risk. In order to achieve acceptable levels of safety, the number of aircraft served at any time may need to be limited – eventually resulting in delays. Performance and safety need to be balanced achieving a system that is able to handle a satisfactory number of aircraft without increasing the safety risk to intolerable levels. However, stakeholders will inevitably need to re-evaluate and trade their initial requirements identifying the most optimum solution for both (in this example) performance and safety. Re-evaluation of requirements has to result ultimately in an acceptable system, satisfying all the system stakeholders and fulfilling its overall purpose.

1.6.4 Changing Requirements

The complexity and the time span of the problems that a (typical) SoS addresses, are two of the reasons that cause SoS operational requirements will change. SoS requirements may change in order to provide the same functionality in a more effective

way, or to provide slightly different functionality. This can be achieved through the reconfiguration of SoS elements, or the addition of new SoS elements in response to capability enhancements of the SoS.

Two examples regarding NCW and air traffic control demonstrate the reconfiguration and capability enhancement characteristics of a SoS. Consider an existing NCW into which UAVs carrying additional sensors are introduced to enhance intelligence collection, or the dynamic allocation of responsibility to ATC centres according to air traffic volume in a given time period. During reconfiguration the role of certain SoS elements may change depending on the operation. For example, a UAV could have intelligence collection function as well as suppression of enemy capabilities. According to the role the system elements certain dependability requirements may differ, to reflect the needs of the particular role.

Another issue that can result in requirements changes is the change of the operational context of a SoS. The most typical example of such a situation is the change from peacetime to wartime operations. The relative importance of requirements may change to reflect the needs of the operation. For example, in wartime operation some reduction in the safety (hence increasing the associated risk) levels of the SoS may be tolerated in favour of increased operational effectiveness.

There is an obvious challenge when attempting to establish a dependability case that will communicate assurance about the satisfaction of dependability requirements. The dependability case should be able to provide justification for the elicitation of the dependability requirements. Different scenarios for the operational context of the SoS may result in different sets of requirements. Simply taking the union of the resulting requirement sets may result in an impasse or in a suboptimal SoS. Requirements elicited for a specific dependability attribute in the context of a specific scenario, may not be in line with the requirements elicited for the same attribute in the context of a different scenario.

In order to overcome the problem of ending up with a suboptimum SoS which satisfies requirements for many scenarios, limitations may be imposed on the number of scenarios that will be 'targeted' by the SoS. Hence, the resulting system will be

optimised for a specific number of scenarios. Alternatively, restrictions on the operation of the SoS may be specified in the form of policy. Irrespective of how a potential impasse is overcome, the dependability case needs to provide justification in a clear and understandable manner. The dependability case could be used to 'inform' the reconfiguration process of a SoS, whilst maintaining acceptable assurance about the satisfaction of the overall dependability requirements.

1.6.5 Traceability

The dependability case of a system can be a part of the evolution of that system. It can be used as a means of recording the requirements and their subsequent decomposition and apportionment to SoS elements. Also, it can be a driver for evaluating the fitness of the design against the required attributes of system operation (a successful design will be easier to argue about).

Overall, the argument contained in a case brings forward many different sources of information, such as analysis and testing. The use of all the different elements that will be part of the final case should be clearly defined. Definition of a rigorous dependability case framework requires well articulated relationships between the concepts used during the evolution of a dependability case.

1.6.6 Interaction of Case and Design

Based on accumulated experience from the safety domain, many standards (such as [13]) suggest that a (safety) case should be constructed in parallel with the system and not at the end of the system lifecycle. Creating an argument about the achievement of dependability in retrospect, after the end of the system development process can be problematic. The design of the system may not be optimised for developers to create a strong argument about its attributes (e.g. safety) easily. Instead, developers will eventually 'force' the argument to support the overall claim often resulting in a weak argument (e.g. relying on operational constraints). Therefore a case will not be able to communicate a satisfactory argument about assurance on the system's required qualities. This means that either the system will be delivered with a case that doesn't provide a satisfactory degree of assurance – something that may result in rejection of the system – or parts of the design will need to be reconsidered so that stronger arguments

can be created. Late in the lifecycle, there are limited opportunities for developers to easily (and cost effectively) revise the system to address problems identified when establishing the safety case. Creation of an argument in parallel with the system serves to help evaluate record and justify decisions regarding the evolution of the system.

1.6.7 Ownership of the Dependability Case

Traditionally in safety, the contractor of a system is responsible for providing a safety case that accompanies the system, which may be reviewed by an independent authority and then submitted for approval to the appropriate regulatory bodies. Whereas this is a clear allocation of responsibility with regard to the safety case, there is a problem of ownership of the case when considering a safety case for a SoS, which increases in complexity when considering a dependability (i.e. multiple attribute) case for SoS. To begin with, typically there is no single contractor for the SoS, but instead there are multiple contractors – responsible for elements (or even parts of the elements) of the SoS. Arguing about aspects of the SoS that involve collaboration of elements may require input from the different individual contractors that develop the SoS elements. Moreover, some of stakeholders may be responsible for certain aspects of the SoS such as performance or safety. This results in responsibility that can be traced to many different elements of the SoS and their respective arguments in the case. Hence, according to the layout of the case architecture, the responsibility of a stakeholder may be dispersed. For these reasons it essential that a framework is defined under which the contribution of each of the contractors to the dependability case can be clearly and traceably identified. Moreover the framework should consider whether an overall authority should be overall responsible for the construction of the dependability case.

1.7 Thesis Proposition

This thesis demonstrates that it is feasible to establish a structured approach to evolving and presenting a dependability case for Systems of Systems through a unified approach to eliciting flexible dependability requirements, facilitating resolution of trade-offs between competing objectives, and combining and managing these activities using structured argumentation.

The main characteristics highlighted in this statement are the following:

- **Structured:** The approach is rigorously captured and documented using an underlying metamodel.

- **Evolving:** The case evolves in parallel with the system, providing feedback to the system development process and influencing design decisions.

- **Systems of Systems:** The type of systems for which the approach has been optimised.

- **Unified:** The proposed methodologies solving the identified challenges are unified within a single framework, with their associations and synergies defined.

- **Flexible:** There should flexibility regarding the specification and achievement of dependability requirements.

- **Facilitating resolution of trade-offs:** The proposed approach identifies and documents conflicts between requirements, facilitating selection of the best solution with respect to the stakeholders' collective interests.

- **Argumentation:** The resultant case is a collection of arguments structured in such a way as to provide compelling overall assurance of system dependability.

1.8 Objectives of the Research

This section summarises the objectives of this research, motivated by the identified challenges, and the means of their achievement. This research focuses on the following objectives:

- Definition of a rigorous framework by means of establishing the Dependability Case Metamodel (DCM).

- Identification of dependable operation (requirements elicitation) by means of applying Dependability Deviation Analysis (DDA) method.
- Resolution of conflicts by means of following a qualitative, argument based Trade-Off Method (TOM).
- Evolving the dependability case in step with system development by means of examining the design rational by following the Factors, Analysis and Decision Alternatives (FANDA) method; by means of specifying a paradigm dependability case architecture encompassing the products of the aforementioned methods.

The objectives of this research – listed above – target the following of the previously identified challenges in the management of dependability cases for systems of systems.

- Multiple dependability attributes
- Allocation and Apportionment of Requirements (not considering negative emergent behaviour)
- Conflicting requirements
- Traceability
- Interaction of between the system and the case development processes

The following subsections give an overview of the research objectives.

1.8.1 Definition of a Rigorous Framework

A dependability case entails the integration of information concerning dependability attributes and derived requirements, arguments, evidence, system models, and underlying design rationale. This requires a clear understanding of the underlying concepts within a dependability case as well as their interrelationships. The dependability case metamodel, aims to create a rigorous fully traceable framework by defining the elements of a dependability case and their relationships.

1.8.2 Identification of Dependable Operation

At the initial stages of the system lifecycle, system stakeholders do not have concrete requirements as to the operation of the system. The overall requirements are elicited in the context of the system's envisioned operation. In later stages of the system lifecycle, design decisions are made and the collaboration between the SoS elements in order to provide the envisioned operation is defined. According to the design, the initially elicited requirements are decomposed and apportioned appropriately. System requirements can be stated from the perspective of different dependability attributes. Moreover requirements can be interrelated – a requirement stated from the perspective of a dependability attribute may depend of the achievement of other requirements stated from the perspective of other dependability attributes. Dependability Deviation Analysis constitutes a method that elicits requirements for the elements of the SoS, by analysing appropriate system models throughout its development.

1.8.3 Resolution of Conflicts

It is inevitable for SoS developers to encounter conflicts between the system's objectives. Unless resolved, the development of the system will reach an impasse. Resolution of conflicts involves compromises during decisions made throughout the system's lifecycle, resulting to trade-off between the stakeholder's goals. The Trade-Off Method (TOM) provides a systematic qualitative approach during which stakeholders share viewpoints and justify the ease with which they can trade their goals. The final outcome of TOM is an argument of preference of a decision alternative (e.g. a design alternative) that is considered to best satisfy the stakeholders' interests.

1.8.4 Development of Case

The dependability case is not developed in isolation from the system. Although a case refers to the final artefact, development of the case takes place hand in hand with the design. System stakeholders specify what needs to be claimed for the final system and accordingly elicit appropriate goals that correspond to each stage of the system development. Evolution of the system and the argument involve making decisions about the architecture and the design of the system, which need to be justified and documented. Interaction between the argument and the design process exists during the

evolution of the system. The argument should evaluate the design's fitness to satisfy the stated goals. A design that is good satisfying the stated goals will result in a strong argument. If the involved stakeholders deem that the argument is not satisfactory, changes to the design will have to be made. FANDA examines how the features of the proposed design alternatives affect the achievement of the goals. Moreover, the thesis proposes a dependability case architecture optimised for the Ministry Of Defence Architectural Framework (described in chapter 2), which is used, in particular, to model SoS.

1.9 Thesis Structure

Chapter 2 provides a literature review of related published work. In brief, the chapter addresses nomenclature of Systems of Systems, examines research and practice in establishing safety cases and argumentation in general, presents an analysis of the notion of dependability, and finally investigates existing techniques in decision making – in particular trade-offs and design rationale within the context of systems architecture.

Chapter 3 presents the basis of the proposed dependability case framework. Specifically, it presents the fundamental concepts that were used to creating the dependability case framework. Moreover, it explains the approach taken to create a metamodel which formally and rigorously defines the framework.

Chapter 4 describes a technique for methodically eliciting and specifying dependability requirements. The technique extends existing analysis methods which focus on examining possible deviations from intended system operation. Dependability Deviation Analysis (DDA) is introduced as a means of identifying the overall goals and concerns of the stakeholders. Furthermore, DDA is used to examine the behaviour of the system elements from the viewpoint of each stakeholder's attributes of interest.

Chapter 5 presents an approach for facilitating conflict reconciliation by trading off dependability goals. The Trade-Off Methodology (TOM) is based on the ALARP principle in safety (explained in chapter 2). This chapter introduces the concept of flexible requirements – necessary for enabling trade-offs to be made. TOM allows

stakeholders to evaluate possible alternatives regarding the satisfaction of their goals – creating arguments for and against committing to each alternative.

Chapter 6 presents the evolution of a dependability case by integrating the proposed methodologies of the previous chapters. It presents the architecture of a dependability case for a System of Systems. The chapter describes how this structure can be populated according to the information available at each stage of the configuration of a Systems of Systems. Factors, ANalysis and Decision Alternatives (FANDA) is introduced as a method to facilitate collaboration and exchange of information between the design and argument development processes.

Chapter 7 presents the evaluation of the proposed contributions against the thesis proposal. The chapter discusses the findings for the methodologies individually as well as for the proposed framework as a whole.

Chapter 8 presents the overall conclusions drawn from the research as well as avenues of possible future work.

Appendix A presents an overview of the activities in the safety lifecycle as suggested by the civil aerospace guidance document ARP 4761.

Appendix B provides an overview of the Defence Architecture Frameworks (namely DODAF and MODAF). In addition, it presents a documented exemplar System of Systems, used as a case study throughout this thesis.

Appendix C presents the Dependability Case Metamodel (DCM), capturing the elements of a dependability case and their associations. Examples of code used to manage dependability case models are also included.

Intentionally Blank

Chapter 2

Literature Review

2.1 Introduction

This chapter describes background material related to the work presented in this thesis. The main areas reviewed are the following:

- **Systems of Systems**: this section includes definitions regarding the concept of SoS, and overview of suitable of modelling frameworks.
- **Dependability**: investigates approaches in defining dependability.
- **Trade-offs**: reviews methods supporting system evolution in the context of competing design alternatives.
- **Dependability cases**: Although the concept of safety cases is well-established, there is little work available regarding dependability cases. The section reviews concepts regarding argumentation of dependability attributes.

The review areas are presented in the following sub-sections of the chapter.

2.2 Systems of Systems

The term Systems of Systems is often used intuitively when large systems cooperate (for example, the term is often used to describe of enterprise networks). There are several existing definitions of a SoS. A review of the modelling methods (UML, ADL) and architecture frameworks (C4ISR framework, Openwings) that have been used to describe SoS are presented.

2.2.1 Definitions

The concept of Systems of Systems (SoS) was introduced as a term for complex systems, entailing components that were developed independently of the rest of the system. Although initially there were no formal definitions, and the identification of SoS was based on a common set of characteristics, the concept has considerably evolved and currently is considered to constitute a class on its own.

Within the military domain, the term Systems of Systems evolved from the concept of network centric warfare [3]; "*Systems of Systems comprise a variety of land and air assets integrated via network centric technologies and appropriate procedures*" [13]. The definition identifies the independence of the Systems that comprise the SoS. Integration of the systems requires certain procedures to ensure effective cooperation and exclusion of hazardous and undesirable operation. Moreover, the definition indicates a close association of the term with networks and communication infrastructure.

Kotov defined a SoS to be "*...large scale concurrent and distributed systems, the components of which are complex systems themselves (e.g. enterprise networks). Communicating Structures are hierarchical structures that represent SoS in a uniform, systematic way as composition of a small number of basic systems*" [14]. This definition mainly presents the communications aspect of a System of Systems. Communicating with each other the a number of individual elements can form a SoS. The definition provides an intuitive description of a SoS, however it only focuses on the network aspect of the SoS not indicating characteristics that could uniquely distinguish a SoS from a large network or a distributed system.

Maier [15] perceives the SoS concept as a natural consequence of the evolution of collaborative systems with increased complexity and operational independence. However he does not provide a specific definition of a SoS. Maier states that: "*While the term System of Systems has no clear and accepted definition, the phenomena are widespread and generally recognised*". Maier mentions that the most important characteristic that identifies a SoS, is the independence of the elements that consist the SoS: "*Systems of Systems should be distinguished from large but monolithic systems by the independence of their components, their evolutionary nature, emergent behaviours*

and a geographic extent that limits the interaction of their components to information exchange". Even though the author identifies some characteristics of a SoS, such as the independence of the elements, he too defines a SoS in terms of component intercommunication: "*Systems of Systems are defined by communication standards. Different problems require standards at different levels*". Although the identified SoS characteristics are broader than the 'pure' network view of Hewlett Packard, the definition is still based on the same principles (i.e. communication means). Maier identifies three categories of SoS: directed, collaborative and virtual. The criteria for defining these categories are the different levels of operational and managerial independence of the SoS elements, which can result in different SoS behaviour.

A broader definition suggested by Periorellis states that: "*...a SoS is a dependable system composed of independent autonomous systems. The purpose of the SoS is to provide a set of enhanced or improved 'emergent' services, based on some or all of the services provided by participating component systems*" [16]. Periorellis identified that SoS elements can support roles that were specified without having overall SoS objectives in mind. This results in different types of behaviour that emerge from the combination of the individual capabilities of the components of a SoS. Moreover, he identifies the fact that such systems are required to have collective mechanisms to ensure reliable operation, as theoretically there can be indefinite and undefined failure modes. This definition highlights a characteristic of SoS components, which is their ability to coordinate a collective behaviour. When a SoS is defined by focusing on its communication structure (as in [14]) the latter considerations aren't always apparent.

Independence of the SoS elements poses a major challenge. SoS elements cannot be implemented for a specific role or behaviour within a SoS. However some of its inherent functions can be used when operating as part of a SoS to achieve some overall objective. According to the description of the available functions, collaborating elements choose the elements choose to collaborate with SoS elements that can provide the necessary functions to fulfil their operational goals. "*...The analogy was drawn to planning a city, calling for simulations to work together as a community, in Systems of Systems. To build and operate an efficient city, a governing framework (e.g. street plans, building codes) is laid out and certain basic services (e.g. utilities, schools) are provided. Beyond that the residents are generally left to their own discretion as to what*

type of home or business they build, who to interact with etc" [17]. Under this perspective, a SoS is a collection of services provided by a set of elements that collaborate with each other. An important aspect of this definition is that communication lines are not the definitive element of a SoS. Instead, the SoS is defined according to the 'services' its collaborating components can provide. In this case the communication infrastructure is considered an equal element of the SoS providing a number o services.

Even though the previous definitions do not just focus on the communications of the SoS, but also on the behavioural properties of the elements of the SoS, a question remains as to when a system can described as a System of Systems. As defined by the US Department of Defence (DoD) research development and acquisition office, a System of Systems is: "an assemblage of components which individually may be regarded as systems and which possess two additional properties" [18]. The additional properties of the SoS are the following:

1. Operational Independence of its components: If the SoS is disassembled into its component systems, the component systems must be able to operate independently. That is, the component systems fulfil customer or operator purposes on their own.

2. Managerial Independence of the Components: The component systems not only can operate independently, they do operate independently. Component systems are separately acquired and integrated, and maintain a continuing operating existence independent of the SoS.

The definition provided by the DoD gives practical means for distinguishing between a large or complex system and a System of Systems. The fact that SoS consist of independent elements introduces characteristics such as element heterogeneity, autonomy and the capability of an element to interpret the overall SoS goals according to its atomic capabilities. Decentralised control is also a SoS characteristic that allows the elements to take decisions on their own. Such decisions can have an impact on the overall behaviour of the SoS.

Overall, the presented definitions put forward a unique aspect of SoS. Kotov [14] and Maier [15] focus on the communications infrastructure and identify some of the characteristics of remote interacting independent systems. Periorellis [16] and Hollenbach [17] focuses on the behaviour of the SoS and how it is achieved by assembling a system from independent systems. Although the DoD definition [18] identifies the characteristics of SoS in their domain, it focuses on an ontological view, specifying why an assemblage of components will be called System of Systems and not a complex system.

Even though all definitions have identified a particular viewpoint of SoS, it can be argued that are not abstract enough and appropriate as a generic definition of the subject. Earlier work of the research strand[1] of which the author was a member, produced a more generic and flexible definition of SoS. Defence companies involved with SoS development have shared our opinion, throughout comparison with their case studies. Therefore the definition that will be used as a basis for further research has as follows:

> A System of Systems is an organised complex unity assembled from distributed autonomous systems, capable of independent provision of services, collaborating to achieve an overall system purpose.

2.2.2 Modelling

The main purpose of modelling is to represent and communicate the structure, the architecture and the behaviour of the system. The unique characteristics of the SoS behaviour should be modelled and documented from early stages of the SoS development. Kaanich [19] presents the need to accommodate dependability within the model of a SoS from the early stages. This section presents frameworks capable of capturing the characteristics of a SoS.

[1] Defence and Aerospace Research Partnership (DARP) strand 2: Dependable Systems of Systems (DSoS). Members of DARP include BAE Systems, QinetiQ, Rolls-Royce and the University of York.

2.2.2.1 UML

The Unified Modelling Language (UML) is currently a well-established and popular language in Object Oriented Development. UML integrates different views of a system from its early conception to deployment (e.g. Use case view, Implementation View, Process view and Deployment view). UML can provide a seamless way of representing the system during its development lifecycle, maintaining consistency between the different models.

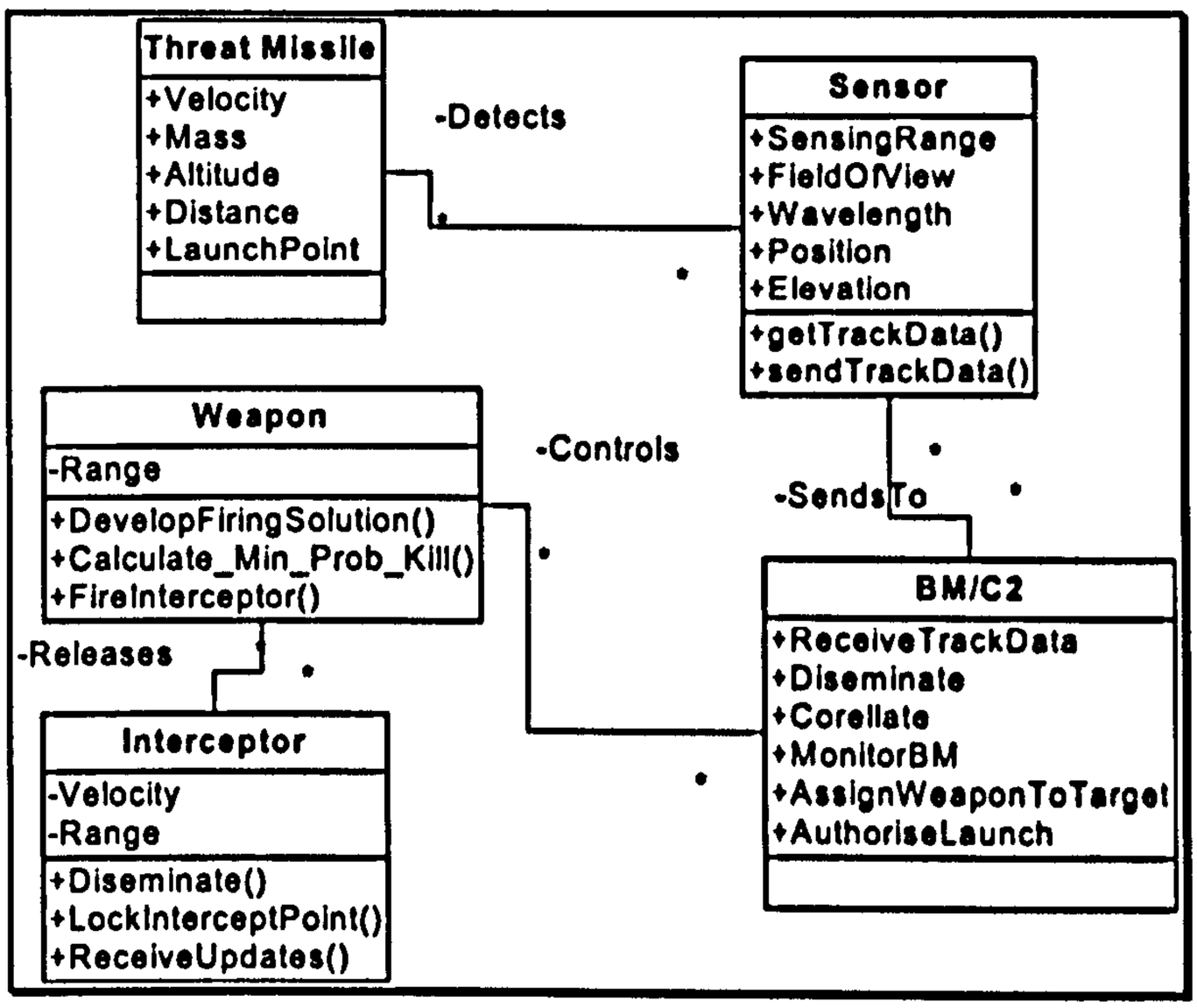

Fig.2.1 – UML Diagram for a BMD SoS [20].

With respect to SoS modelling, UML has been used to represent the SoS elements, and to show the data and the data types required in element collaboration. Caffal [20] uses a UML class diagram to conceptualise the elements of a Ballistic Missile Defence SoS.

The top-level design is a conceptual class diagram with abstract classes, classifying the type of systems used (e.g. sensors, weapon). The abstract classes have a description of data that are needed by the SoS, and are associated with other classes, indicating the role and collaborations of each element. For example the control class receives data from the sensor class and controls the weapons class.

UML has been suggested to be effective in abstracting the network infrastructure modelling the SoS as a single system: "*Rather than disparate reasoning about the individual systems of a proposed System of Systems, we propose that we develop a sound model for reasoning about the System of Systems a single functional unity*" [20].

UML provides a widely adopted modelling framework, with expressive power and a variety models capable of capturing capture many aspects of a system's operation. Although UML is not optimised in representing Systems of Systems, certain UML models can potentially be related (in terms of the information that capture) with frameworks optimised for SoS such as the ones presented in the next section.

2.2.2.2 C4ISR & Defence Architecture Frameworks

The identification of the need for global (military) awareness has led to the identification of C4ISR (Command Control Communications Computers, Intelligence, Surveillance, Reconnaissance) systems. In order to make the transition of SoS from concept to design, the C4ISR architecture framework [21] was proposed.

The C4ISR architecture framework provides 3 different architectural views: Operational, Technical and Systems [21]. The operational view describes the tasks and activities of concern and the information exchanges required. The systems view describes the systems of concern and the connections among those systems in context with the operational view. The technical view describes a profile of a minimal set of time-phased standards and rules governing the implementation, arrangement, interaction and interdependence of system elements.

A key challenge identified in C4ISR architectures is sharing of data between the elements of a C4ISR system. This means that each element will share their perception of the environment with other SoS elements eventually composing an enhanced picture of the battlefield. Jameson [22] uses the term data fusion to describe a means of sharing information between SoS elements, and presents an example of how this can be achieved between the heterogeneous elements of a SoS.

The C4ISR framework was superseded by the Department of Defence Architectural Framework (DODAF) [23]. DODAF provides a framework organised in views similar to the C4ISR, in which defence systems can be modelled. In particular, systems similar to NCW can be organised and modelled from every aspect of their operation. The Ministry of Defence Architectural Framework (MODAF) is a very similar framework used by the UK Ministry of Defence. DODAF and MODAF are covered extensively in appendix B in which an example system (AGO) is modelled using the framework.

2.3 Dependability

Dependability is an abstract term describing a system's overall behaviour. Dependability as a concept is broadly accepted as being the "*...the system's characteristic that justifies placing ones reliance on it.*" [24]. Littlewood et al. state that: "*We use dependability informally to designate those system properties that allows us to rely on a system functioning as required*" [25]. Although in abstract terms the definitions are similar there is no overall consensus as to the exact definition of dependability.

2.3.1 Definitions

McDermid sates that: "*...dependability can only be thought of as a function of a system and its environment, not as a property itself...the form of specification used sometimes depends on the particular dependability characteristic of interest*" [26]. Villemeur, from a systems engineering perspective, gave the following definition for dependability [27]: "*In its broadest meaning, dependability will be defined as the science of failures: it, therefore encompasses the knowledge of these failures, their assessment, their prediction, their measurement and their control*". According to Villemeur dependability can include the following characteristics:

- Reliability
- Availability
- Maintainability
- Safety
- Durability

- Service retainability performance.
- Servability performance.

Villemeur defines dependability as a composite term, comprising of seven characteristics, which are achieved by being able to predict, control, and assess failures. The criteria to assess dependability are associated to the fulfilment of the required functions and the conditions in which the system operates. This is in accordance to the previous suggestion that dependability is a function of a system and its environment. The primary focus of the definition is on the failures of a system. Laprie et al. extended the initial concept and defined dependability for computer systems identifying dependability attributes, as well as means of achieving it: "*Dependability is the system property that integrates such attributes as reliability, safety, confidentiality, integrity, survivability and maintainability, is achieved by means of fault tolerance, fault prevention, fault removal and fault recognition and it may be compromised by faults, errors and failures*" [24].

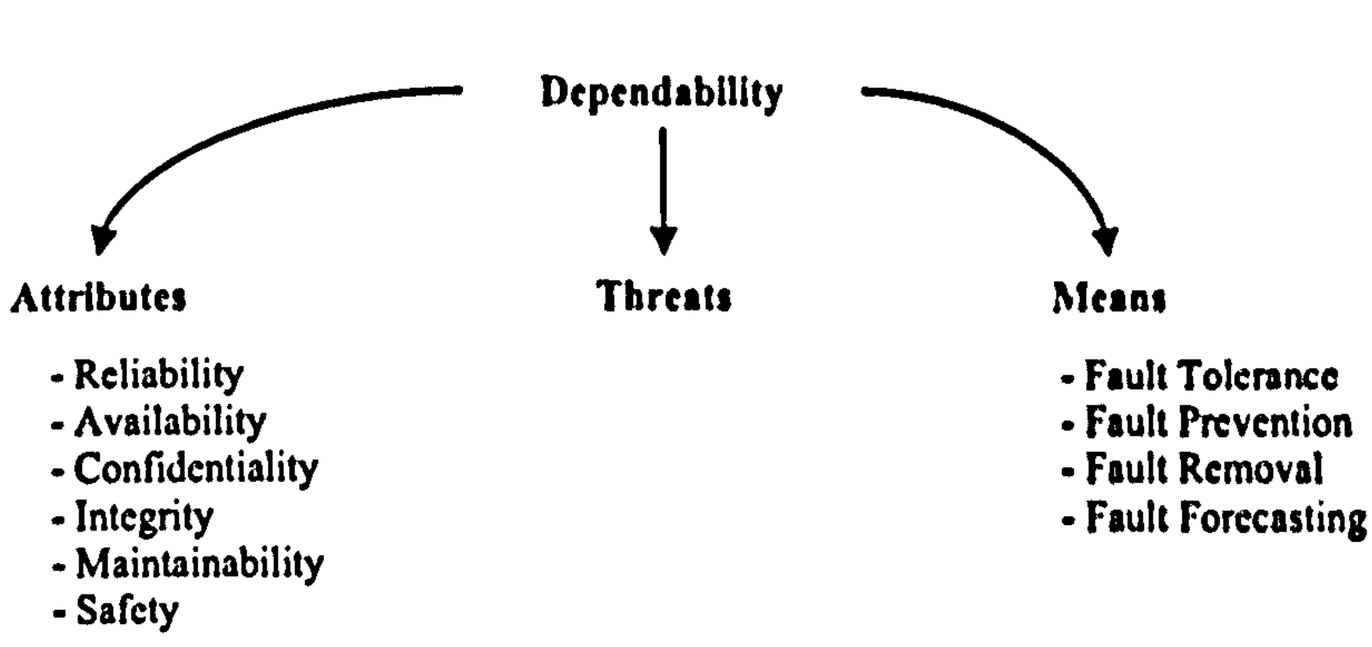

Fig.2.2 – Laprie's Dependability Tree

This is the most complete of the definitions as it provides a set of attributes that comprise dependability. Laprie recognises the causes that can compromise dependability, and identifies the means of achieving it. The attributes are system properties that describe the system's overall behaviour in respect to its requirements.

Lack of consensus on a specific definition is mirrored by the few standards defining dependability. IEC 50-191 defines dependability similarly to Laprie: "*The collective term used to describe the availability performance and its influencing factors:*

reliability performance, maintainability performance and maintainability support performance." [28].

Saridakis et al [29] give a fault-tolerant centric definition of dependability based on Laprie's observations. According to Saridakis dependability properties fall into two groups:

1. Abstract Properties specified in terms of system states, which are defined independently of any fault tolerant technique. They serve to characterise the dependability behaviour of an overall architecture, when this behaviour is too abstract to associate a fault tolerance technique with it.

2. Concrete Properties specified in terms of system actions, whose definition is closely related to some fault tolerance technique. They serve to characterise the dependability behaviours associated to architectural elements, with respect to a given fault tolerance technique.

Saridakis comments that "*...the most abstract dependability property, simply qualified as Dependability, ensures that a system makes progress despite the occurrence of failure*" [29].

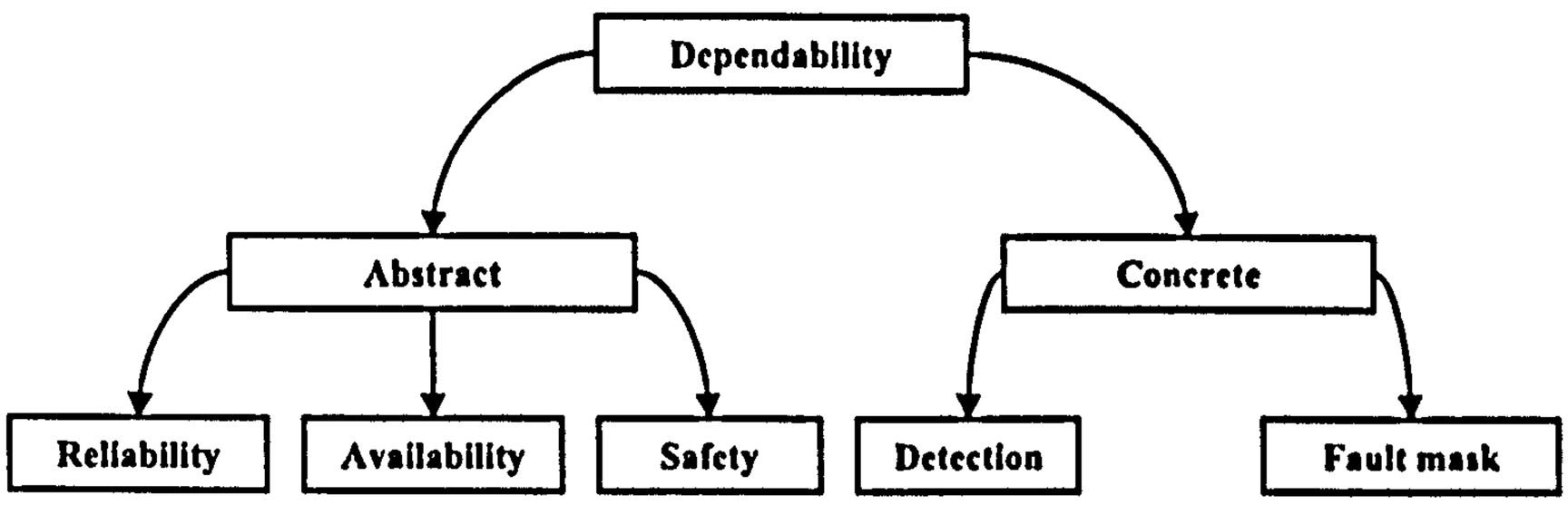

Fig.2.3 – Categorisation of Dependability Properties

Prasad defines Dependability as, "*... a variable sized vector of attributes describing overlapping desiderata, chosen subjectively, in accordance to the stakeholders' particular requirements*" [10]. Furthermore, Prasad highlighted that dependability

attributes can be interrelated, but they are not orthogonal to each other, and can be in conflict or in harmony. Prasad's definition focuses on the behaviour of the system with its environment, as suggested by McDermid. Prasad's definition is also the definition adopted throughout this thesis.

2.3.2 Dependability & 'Fault Science'

As mentioned by Laprie [24], dependability has threats that can compromise it, as well as means of achieving it. Taking in account the broader definition of dependability, a threat should be anything that will result into the system operating erroneously and not as expected. Both Laprie and Villemeur, as well as a broad range of researchers, adopt the opinion that dependability is compromised by faults that exist within the system, resulting in failures and hence in erroneous behaviour. However, it has been obvious that dormant faults within the system are not the only reason a system's dependability can be reduced. Also, the environment can cause the system to enter an erroneous behaviour due to environmental hazards, operator errors or malicious attacks.

A fault is a defect within the system; faults can be random (e.g. a defective component) and systematic (e.g. wrong design of a system). Not necessarily all faults result in failures. Faults may be dormant in a system until activated by one or a sequence of events. When activated faults produce failures, which results in the system failing to perform its required function. When a failure occurs, if the system's behaviour deviates from required operation, then the system has entered an erroneous state.

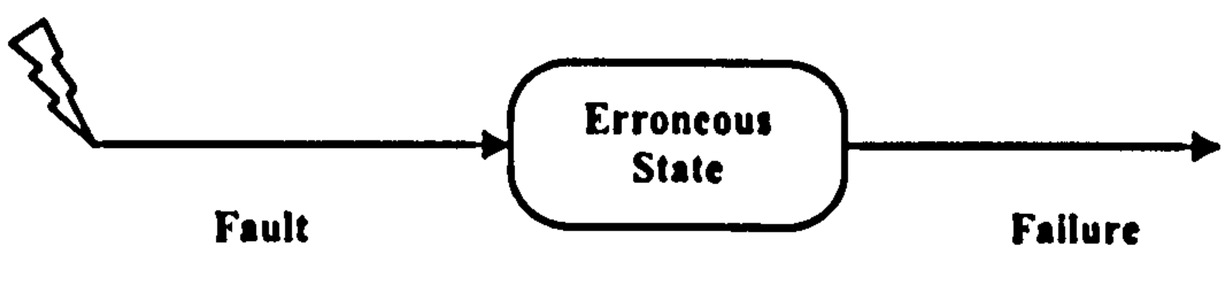

Fig.2.4 – Fault Transition to Failure

Fig.2.4 illustrates an example modelling the propagation of a fault [30]. Failures are deviations from expected behaviour and we can identify that a system has failed in respect to its environment. Failures are the result of the fault activation to the system's environment. Faults can be classified by the persistence and their source [31]. The

source of a fault depends on the use of the system and the environment within which it has been deployed. For example, there can be operator faults, interface faults, storage faults etc. According to their persistence, faults have been categorised as design, operational and transient faults. Design faults are removable and can be corrected by redesign of the faulty components. Operational faults are non removable and occur when a part of the system breaks (e.g. database corruption). Finally transient faults are random faults that are not deterministic and often cannot be reproduced. Laprie et al. [24] combined all fault classes and categorised the faults according to the measures that need to be taken to eliminate the faults into design, physical and interaction faults.

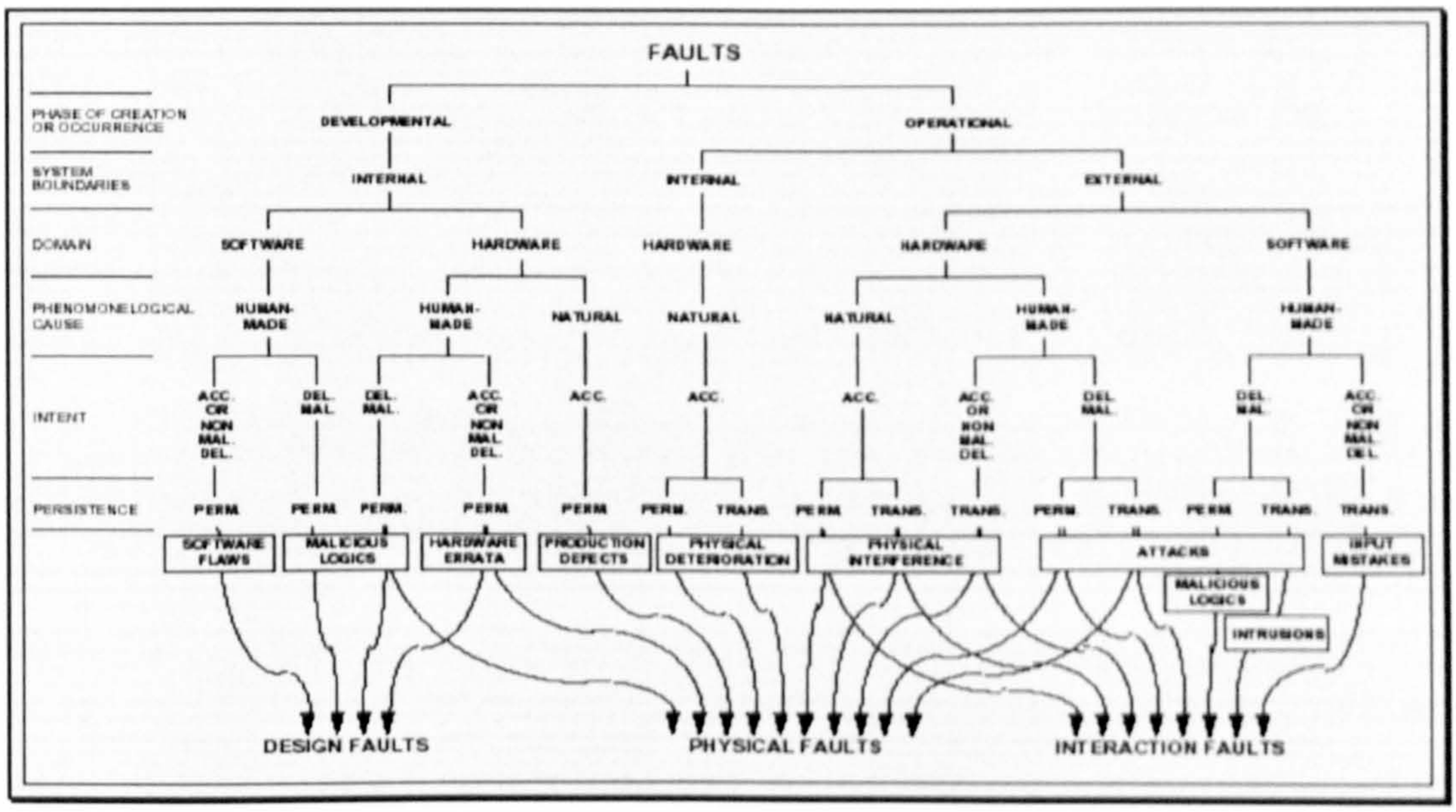

Fig.2.5 – Laprie's Categories of Faults

The number of faults in a system has been associated with the reliability of the system. Reliability has been defined as "*the probability that a piece of equipment or component will perform its intended function satisfactorily for a prescribed time and under stipulated environmental conditions.*" [32]. Although it bears similarities to the definition of dependability, reliability is a mathematical representation of the probability of the components failing [33]. The impact of a reliability failure on the system's dependability can affect other dependability attributes. For example a reliability failure of a processor may have an impact on safety if the processor is used to perform safety critical functions, or it could have an impact on the performance on the system. Hence it is not the fault itself of primary interest to the system's stakeholders but its effect on the dependability attributes, which describe the system's behaviour.

2.3.3 Unification of Safety and Security

Safety and security are two dependability attributes which are considered to share common concepts. There have been a number of studies examining the potential of unifying these two attributes which are presented in this section. A major difference between safety and security is the intent. The methods involved in safety are concerned with non-malicious faults and how these can be avoided or mitigated. On the other hand, security is involved with planned malicious attacks to the system.

Jonsson et al. [34] identified the functional relations between security and dependability threats. He suggests that the overall objective is: "*to arrive at a general and clear-cut framework that would describe how trustable (dependable, secure) a system is, regardless of the reason for its not being totally trustable*". For example, it should be possible to treat a system failure caused by an intentional intrusion or a hardware fault using the same methodology". Jonsson's analysis is based on the fact that "*...in the dependability discipline, reasons for failures are called faults and errors, whereas security people traditionally talk about attacks that cause breaches and vulnerabilities*". Jonsson also provides an integrated framework for dependability and security, illustrated in Fig.2.6.

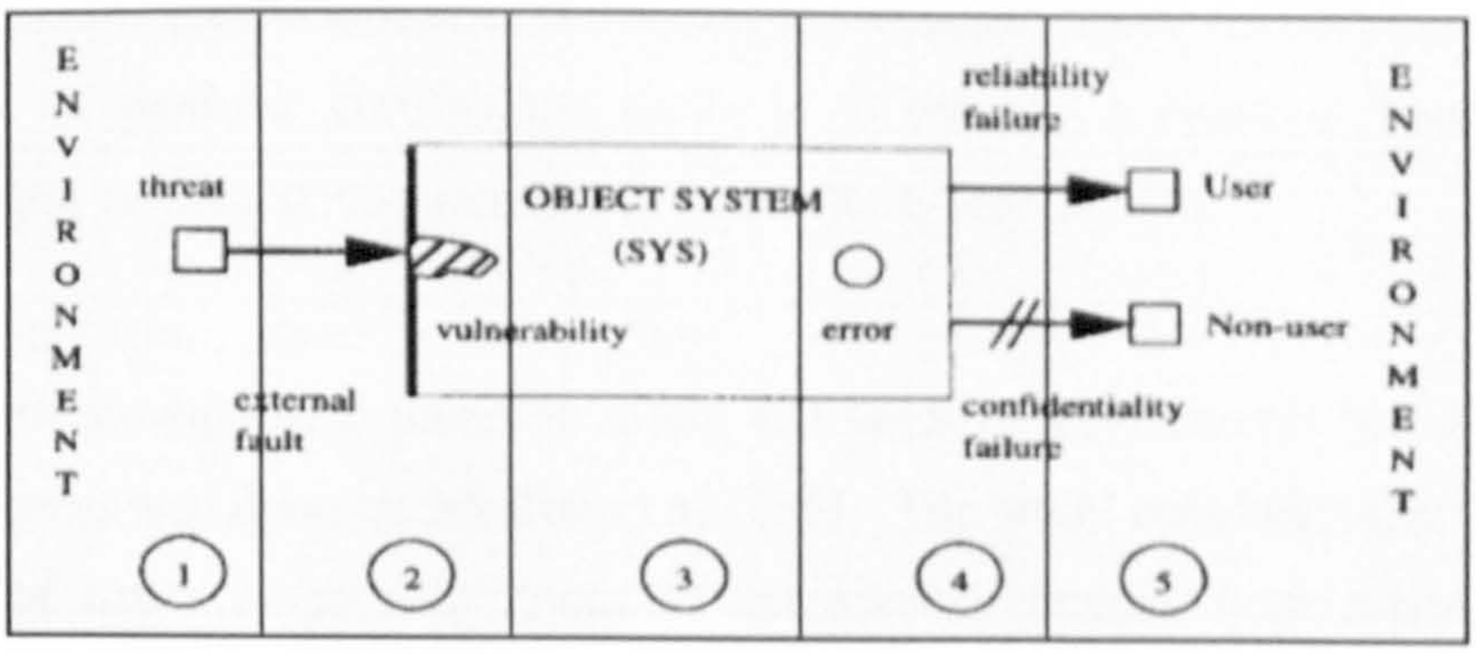

Fig.2.6 – Unification of Safety and Security [34]

The framework identifies five main areas in the system and its relationship with the environment. A threat (1) ("an environmental subsystem that can possibly introduce a fault in the system") introduces a fault exploiting a vulnerability (2) ("a place where it is possible to introduce a fault"). This propagates into the system and causes an error (erroneous state) (3), which can have as a consequence the occurrence of a failure (4)

that then affects the system's behaviour (dependability) with respect to its environment (5). In general Jonsson's framework provides a generalisation and does not accurately present all possible states of a system in presence of a fault. Moreover, the generalisation cannot be extended in covering other attributes such as safety, for which there exist well defined frameworks and descriptions of how faults can cause accidents. However, Jonsson's framework is useful in introducing the concept of interaction between dependability attributes.

Another attempt to integrate safety and security is SafSec [35]. The objectives of the SafSec project were to evaluate the feasibility of combining current and future methods of acceptance of the next generation of military avionics against their safety and security requirements. Predominantly, SafSec seeks to provide a means for joint certification of military aircraft for safety and security. A unified approach proposed by SafSec involves simultaneous reasoning about achievement of assurance regarding the system's safety and security properties.

SafSec proposes the integration of safety and security in the following way:

- Unified approach to risk assessment – combining the error oriented approach of Safety with the action-oriented approach of Security.
- Adoption of a risk directed approach throughout the assurance life-cycle – using common language for specifying the desired properties of a system.
- Use of modular certification methods to provide a common framework for setting evidential requirements to facilitate re-certification.

Similar work attempting to integrate safety and security requirements based on unified risk assessment was done by Moffett et al. [36]. The study concludes that: "*while the definition of safety or security could be extended to include both concepts, in the majority of the situations it is inappropriate to attempt to unify safety and security risk analysis techniques...*", he also point out that a safety case must provide separate argument and goals when there is a security and a safety conflict [36].

2.3.4 Measuring Dependability

Being an abstract concept, it is difficult to examine the dependability of a system as there is no metric associated with it. This has implications in decision making because of the difficulty in comparing the dependability of two or more systems. Individual dependability attributes have been associated with metrics; for example it possible to identify that a system A is more reliable than a system B, based on quantitative representation of reliability. However there were few attempts to provide an overall metric for dependability. A notable attempt is from Prasad [10] who integrated the individual attribute metrics in order to produce a single metric that will express dependability numerically. The suitability of a number of methods and models (e.g. additive method, hurt model, multi attribute utility theory) to produce a single metric for dependability was investigated. Results showed it is not infeasible to derive a useful and meaningful single metric of dependability that will reflect the behaviour of a system with respect to its dependability attributes. Prasad explicitly states that: "*It is infeasible to directly measure dependability as a composite property using a single real number, even on a scale as weak as ordinal. This is because any given pair of systems may not be comparable due to multiple attributes being implicit in the comparison, or, may be ordered by different stakeholders*" [10].

2.4 Trade-offs in System Design

Developing a dependable system involves addressing many (non-orthogonal) attributes. Consequently, having to resolve possible conflicts between the different dependability attributes and make trades is an inevitable situation, especially for large-scale systems. Laprie states that tradeoffs are one of the reasons that developmental errors are introduced. "*During development, faults result generally from tradeoffs, either a) aimed at preserving acceptable performance and facilitating system utilisation, or b) induced by economic considerations*" [24]. Establishing a dependability case for a system necessitates reasoning about those possible trade-offs. In some circumstances trade-offs can be made relatively straightforwardly. For example, consider the availability and safety of an airliner. Flying an aircraft with all its systems operational would require repair times that would make the aircraft unavailable. On the other hand 'loss' of functionality has an impact on the safety levels of the aircraft. This compromise has been historically resolved by consulting the Minimum Equipment List

(MEL) of the aircraft. A MEL specifies the systems that must be operational so that the aircraft can take off. The list is populated after assessing the impact on risk (probability and severity) when a system is unavailable. By comparing the resultant risk with the maximum acceptable it is possible to specify whether the aircraft is airworthy and the time interval within which the system has to be repaired. However, there can be cases when simultaneously attempting to trade-off a large number of different goals in a context where the priorities may not be so clearly defined (e.g. the defence domain as opposed to the civil domain where safety is ideally the highest priority). In such cases, in order to establish justified trade-offs a methodical approach is required, systematically addressing a number of considerations.

By identifying the tradeoffs between the dependability attributes and their relation to the design of the system, an acceptable balance can be found. Three main strands were investigated under the trade-off section.

- Methods used in systems engineering to make trades between system requirements. ATAM is one of the most prominent methodologies used for trade-offs and is presented in this section. Several other methods are briefly described in chapter 5, along with discussion regarding identified concepts used in trade-offs.

- Design Rationale is used during the development of a system in order to facilitate elicitation of design decisions and achieve agreement between the different balancing their requirements.

- As Low As Reasonably Practicable (ALARP) is an approach imposed by the UK Health and Safety Executive (HSE), requiring developers to justify trade-offs involving safety.

2.4.1 The Architecture Tradeoff Analysis Method (ATAM).

ATAM is a method developed by the Software Engineering Institute (SEI) in Carnegie Mellon University. The objectives are to understand the tradeoffs of candidate system architectures and select the one that will satisfy the best, the requirements of the system.

ATAM is a structured technique for understanding the tradeoffs inherent in the architectures of software intensive systems. The method was developed to: "*provide a principled way to evaluate a software architecture's fitness with respect to multiple competing quality attributes*" [37]. According to the definition adopted in ATAM, quality attributes of a system constitute its behaviour with respect to the environment.

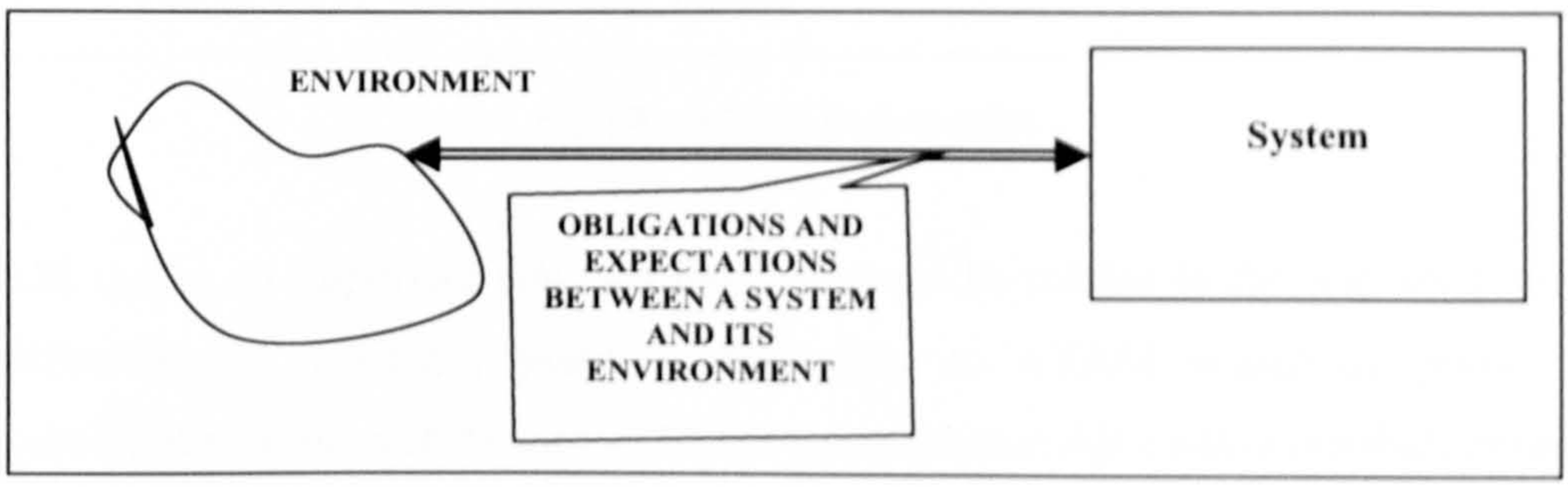

Fig.2.7 – System Quality Attribute Obligations

Quality attributes represent the expected behaviour of a system, observable at its boundary as presented in Fig.2.7 [38]. This observation introduces the notion of a contract between the system and its environment, based on the success of the System to provide the environment the expected behaviour. Although the concept is introduced from the viewpoint of quality, the overall objective (i.e. the system should behave as expected) of defining quality attributes is the same as in dependability.

According to ATAM, quality attribute behaviour is related to the design of the system – several designs may satisfy one attribute but not another, when a trade off exists. "*The ATAM is meant to be a risk identification method, a means of detecting areas of potential risk within the architecture of a complex software intensive system. ...Risks are architecturally important decisions that have not been made or decisions that have been made but whose consequences are not fully understood*" [39]. The tradeoffs are then resolved by prioritising the systems actions to the external stimuli according to the severity of the scenario, and hence by prioritising the quality attributes that correspond to the respective scenario actions. Fig.2.8 depicts the inputs and outputs of ATAM.

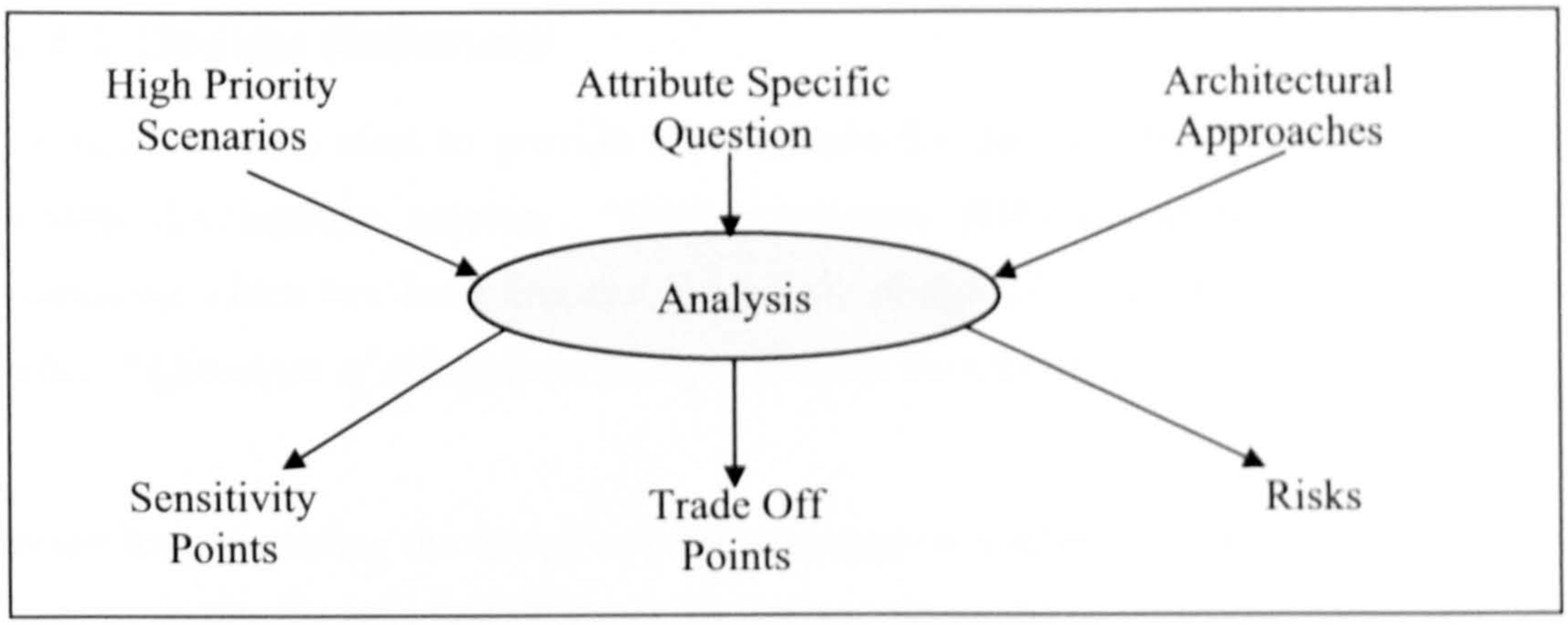

Fig.2.8 – ATAM Deliverables

ATAM makes an important contribution that could be related to the SoS concept; the identification of sensitivity points. According to ATAM sensitivity points are: "*...parameters in the architecture to which some measurable quality attribute response is highly correlated.*" [39].

Therefore we can see that the behaviour of a system is related to the design decisions that are made. Possible use of the concept in a SoS dependability case would be to relate the arguments about the dependability attributes to evidence directly coming from the SoS design. Identification of sensitivity points can facilitate in understanding how the design decisions made during system development, can help achieving the system goals described in the dependability case.

ATAM is a very useful method that identifies tradeoffs between system attributes that are related to the design of the system and provides a methodology in order to select the most suitable design according to a set of scenarios the system has to satisfy. This could form a basis for the concept of dependability case. However the method does not mention interrelation of a system's components and how one's behaviour affects the other. Moreover the ATAM methodology report explicitly mentions that ATAM is not a structured method that can be used for formally reasoning about a system, but it is a method to evaluate the effect of different scenarios in combination with the design on the quality attributes.

2.4.2 Design Rationale

Design Rationale aims to provide the rationale for design selections throughout the system development process. "*Design rationale (DR) expresses elements of the reasoning which has been invested behind the design of an artefact. A DR answers why...? Questions of different sorts, depending on the class of DR represented.*" [40].

Apart from providing the design options necessary to evolve a system, there is also the need to justify the selection of a specific option. Some design rationale approaches use argumentation techniques to justify design selections based on a set of criteria, concerning a particular problem in the system [41]. Argumentation based design rationale is claimed to help in the problems such as [40]:

- Clarify vague requirements, and tracking the rationale for their inevitable evolution.
- Represent multiple stakeholders' viewpoints, including that of end-users in participatory design.
- Negotiating trade-offs between multidisciplinary analyses, such as software and user criteria.
- Maintaining consistency in decision-making, e.g. through propagating changes through networks.
- Communicate rationale to other designers.
- Building cumulative design knowledge, through systematic re-use of rationale.

The ability of DR to allow negotiation of trade-offs can be used to show how a dependability attribute is traded off against another based on a design decision. This bears similarities to the ATAM method. The objective of the ATAM is to evaluate the different design selections and identify the (design) sensitivity points that affect the achievement of the quality attribute. Design rationale representations can be formal, semiformal and informal (Fig.2.8). The more informal the representation is the easier it is for humans to conceptualise the rationale behind any design decisions (Fig.2.9).

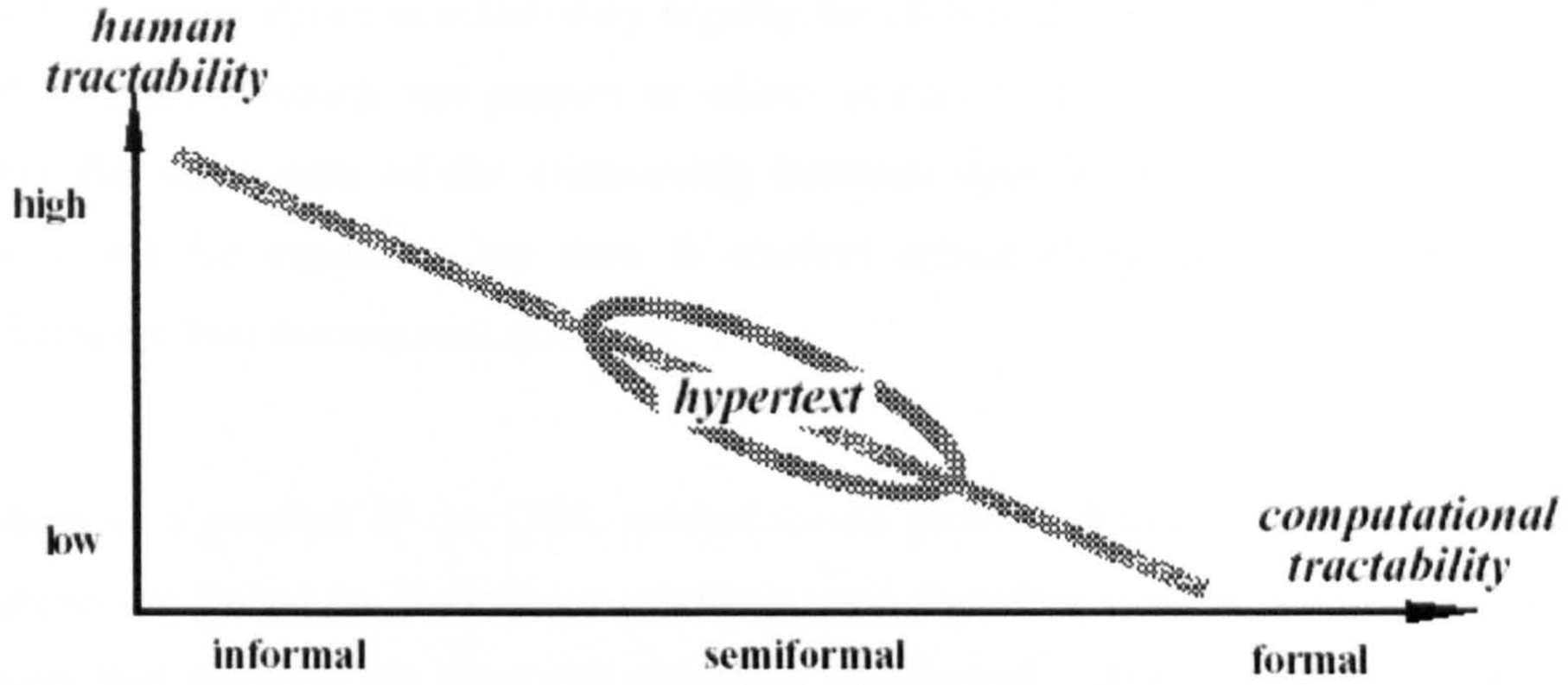

Fig.2.9 – Design Rationale Representations [40]

In this chapter semiformal representations of design rationale are examined, since they are more suitable to accommodate argument based rationale [40], [41]. Design rationale captures the dependency each requirement of the attributes has, on design options. The following pages provide a brief description of DR representations that were studied.

2.4.2.1 QOC

The Question Option Criterion (QOC) is a graphic representation for design rationale used for design space analysis. Fig.2.10 illustrates QOC. Questions used to describe the system's required behaviour, which are answered by design options which are considered to be able to achieve the question indicates.

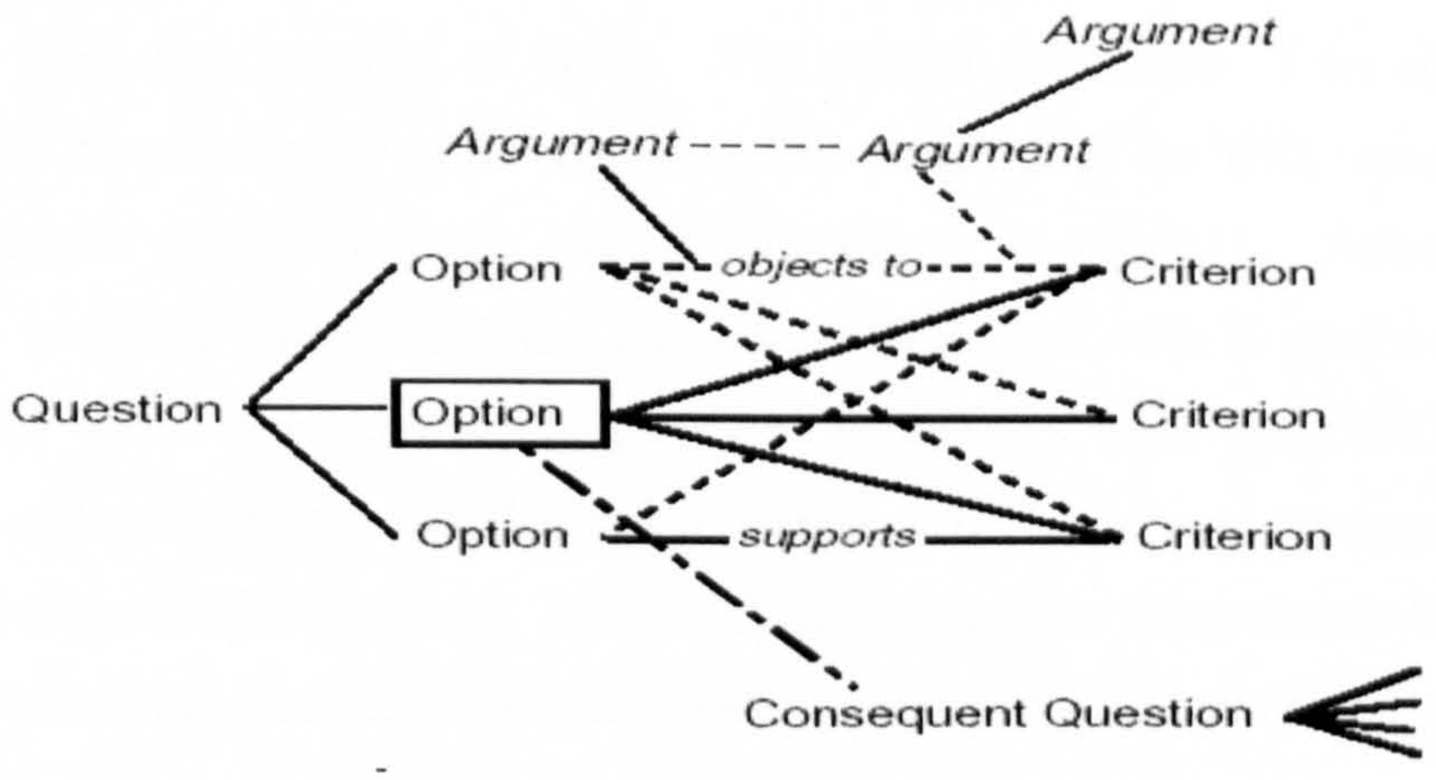

Fig.2.10 – The QOC Method [40]

The best design option is selected by arguing for each of the available options based on certain criteria, which can support or object selection of a particular design option. After the assessment of the relationship between options and criteria (supports or objects to) the arguments are used to conduct debate about the design option that satisfies the best the required question.

Debate as a product of the QOC method could provide discussions about tradeoffs. Options are linked to consequent questions, and therefore inherently the best design option that satisfies the most the questions is selected. However tradeoffs are not explicitly mentioned and there are not any defined procedures used to identify the tradeoff point as in ATAM and its impact on all the required attributes.

2.4.2.2 SIBYL

SIBYL is another tool used for design rationale. Lee describes SIBYL as "*a system that supports group decision making by representing and managing the qualitative aspects of decision making; such as the alternatives, the goals to be satisfied and the arguments evaluating the alternatives with respect to these goals*" [42].

The two main motivations for SIBYL are knowledge sharing and qualitative decision support. SIBYL uses the decision representation language (DRL). Using DRL, a decision problem represents the problem of choosing the alternative that best satisfies the system goals. Each alternative is related to a goal via an 'achieves' relation. A relation in DRL is a subclass of claim. The overall evaluation of an alternative is presented by the plausibility of the relation. The structure of the DRL vocabulary and the achievement of the rationale capture are shown in Fig.2.11. Selection of an alternative is based on whether the alternative associated with a goal can credibly achieve that goal. Justification is provided supporting each association between alternative and goal. Being a goal-based approach, SIBYL supports decomposition and refinement of goals into sub goals. Although SIBYL captures the rationale behind some design decisions, the trade off points of a system cannot be easily evaluated.

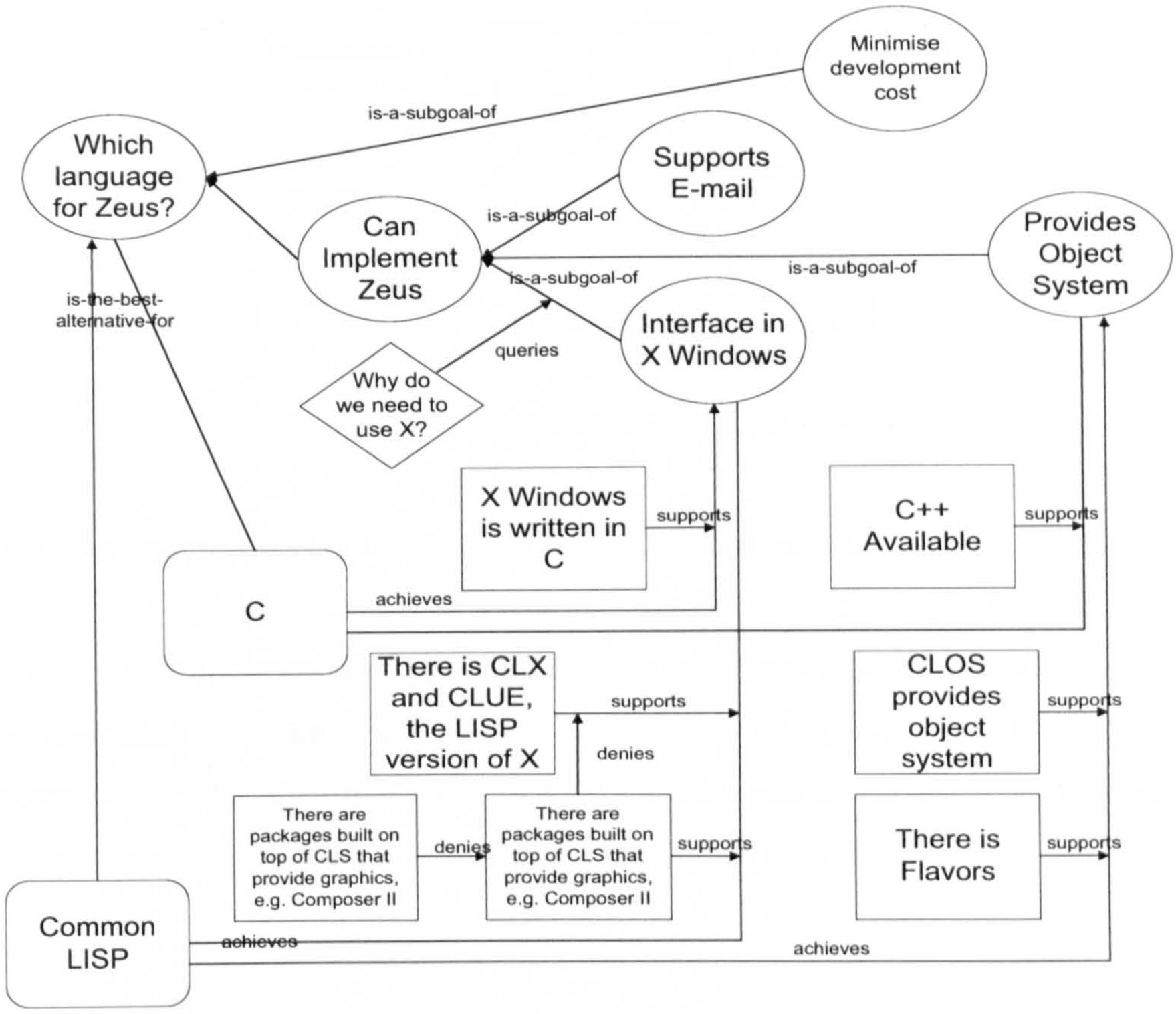

Fig.2.11 – SIBYL Elements [42]

The decision problem does not provide relations to other attributes so as to evaluate architectures based on a collective view of the attributes as ATAM does. However a useful conclusion of DRL is that claims can have both a negative and a positive effect on a goal or sub goal.

2.4.2.3 gIBIS

gIBIS is the last design rationale approach presented. "gIBIS is a hypertext system designed to capture early design considerations" [43]. gIBIS is based on the IBIS method, which helps providing arguments for the different stakeholders' viewpoints during the design process. The gIBIS structure is shown in Fig.2.12. IBIS is concerned with 'issues' that need to be resolved during the design process. An issue is questioned or suggested from a position. A position is a statement or assertion, which resolves the issue. Each position is related to one or more arguments that can support or object it.

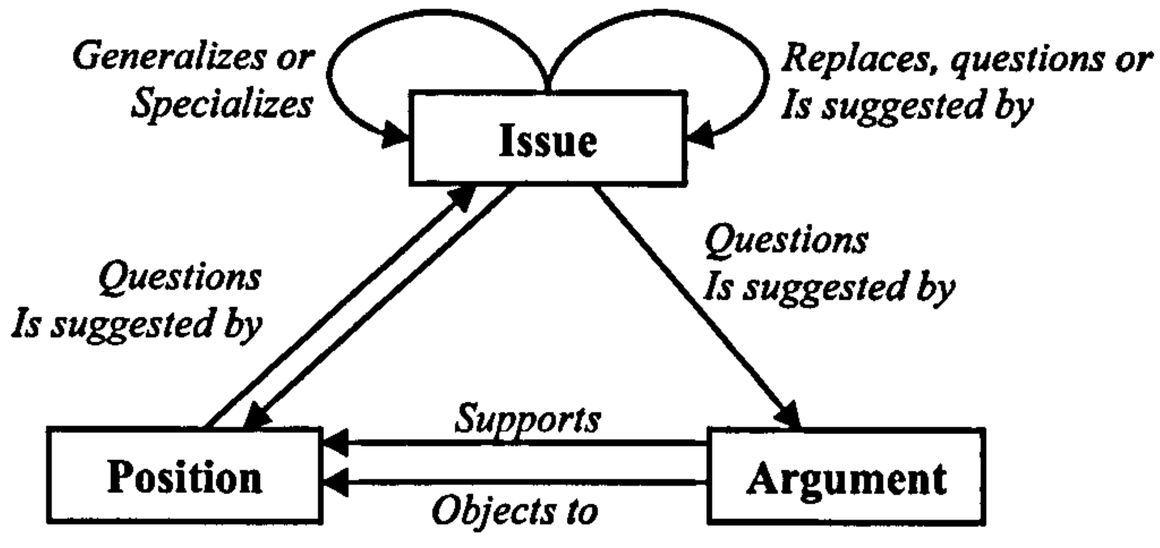

Fig.2.12 – IBIS Main Elements and Relations [43]

Issues can be parallelised with goals and questions in other DR methods, whereas positions and arguments are similar to the 'achieves' and 'supports' associations in SIBYL. As with the rest design rationale methods reviewed, IBIS is used to facilitate communication between stakeholders in the early design stages. It identifies the concept of the design objecting to a goal of the system but there is no structured approach for a collective view of a system based on its requirements.

2.4.3 As Low As Reasonably Practicable (ALARP) Principle

ALARP is a principle applied in safety critical systems, imposed by the UK Health and Safety at Work (HSW) Act of 1974. According to HSW, system developers have a legal obligation to demonstrate to the safety governing bodies that the risks have been reduced to a level that is As Low As Reasonably Practicable (ALARP) [44]. According to ALARP risk falls into three major categories; broadly acceptable (negligible), intolerable and within the ALARP region. The ALARP region is defined by a target and a limit value. When developers of a system accept a risk that falls in the ALARP region, they need to show that the cost of further risk reduction would be grossly disproportional to the actual risk reduction achieved. Decisions made are based on an argument about disproportional benefit (although according to law there should be a bias to safety), and on justification about the target and limit that define the (ALARP) region of tolerability of risks. Walker underlines that ALARP *"requires a comparison of risk and the sacrifice involved in taking measures to avert the risk*" [45]. When making a decision, the system stakeholders need to justify their incentive for making the decision, which in this case will take the form of an argument about the disproportion between safety improvement and cost sacrifice.

Fig.2.13 shows an adapted example of the categories for classification of risk used in safety standards such as [46]. Risk can be classified in three major categories: Negligible, Intolerable, and within the ALARP region. The safety 'target' is to achieve risks that are considered negligible. Additionally, a limit on risk is defined. Risks exceeding the defined limit are considered intolerable (i.e. unacceptable). The region between these two values (target and limit) defines the risks that can be considered to be intolerable if they can be argued to be As Low As Reasonably Practicable (ALARP). A risk is considered ALARP if costs of further risk reduction options can be shown to be disproportionate to the risk reduction that would be achieved.

Frequency \ Consequences	Catastrophic	Major	Marginal	Minor
Improbable	1	1	1	1
Unlikely	2	2	1	1
Possible	3	3	2	1
Frequent	3	3	3	2

KEY
3: Intolerable Risk
2: ALARP
1: Negligible Risk

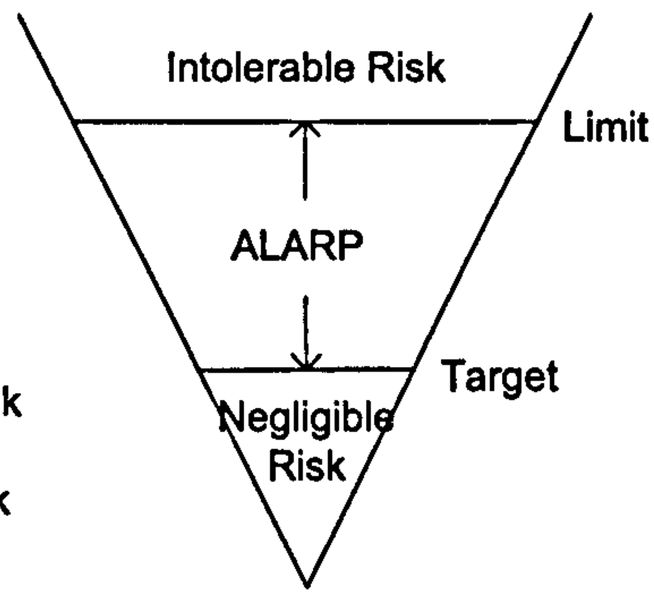

Fig.2.13 – ALARP Categorisation and Risk Acceptance

The resultant system risk can be classified in the following categories:

1. Intolerable Risk, Risk cannot be justified except in extraordinary circumstances.

2. Undesirable Risk, tolerable only if reduction is impractical or costs are disproportionate o improvements gained.

3. Tolerable risk if cost of risk reduction would exceed improvements gained.

4. Negligible risk.

Categories 2 and 3 belong to the ALARP region, where the risk has to be reduced as low as reasonably practicable.

ALARP is a principle indicating a 'pure' tradeoff between safety and cost; the rationale as well as the methods that are used in safety assessment could be extended to accommodate more attributes than just safety. ALARP is used to assess the overall system safety.

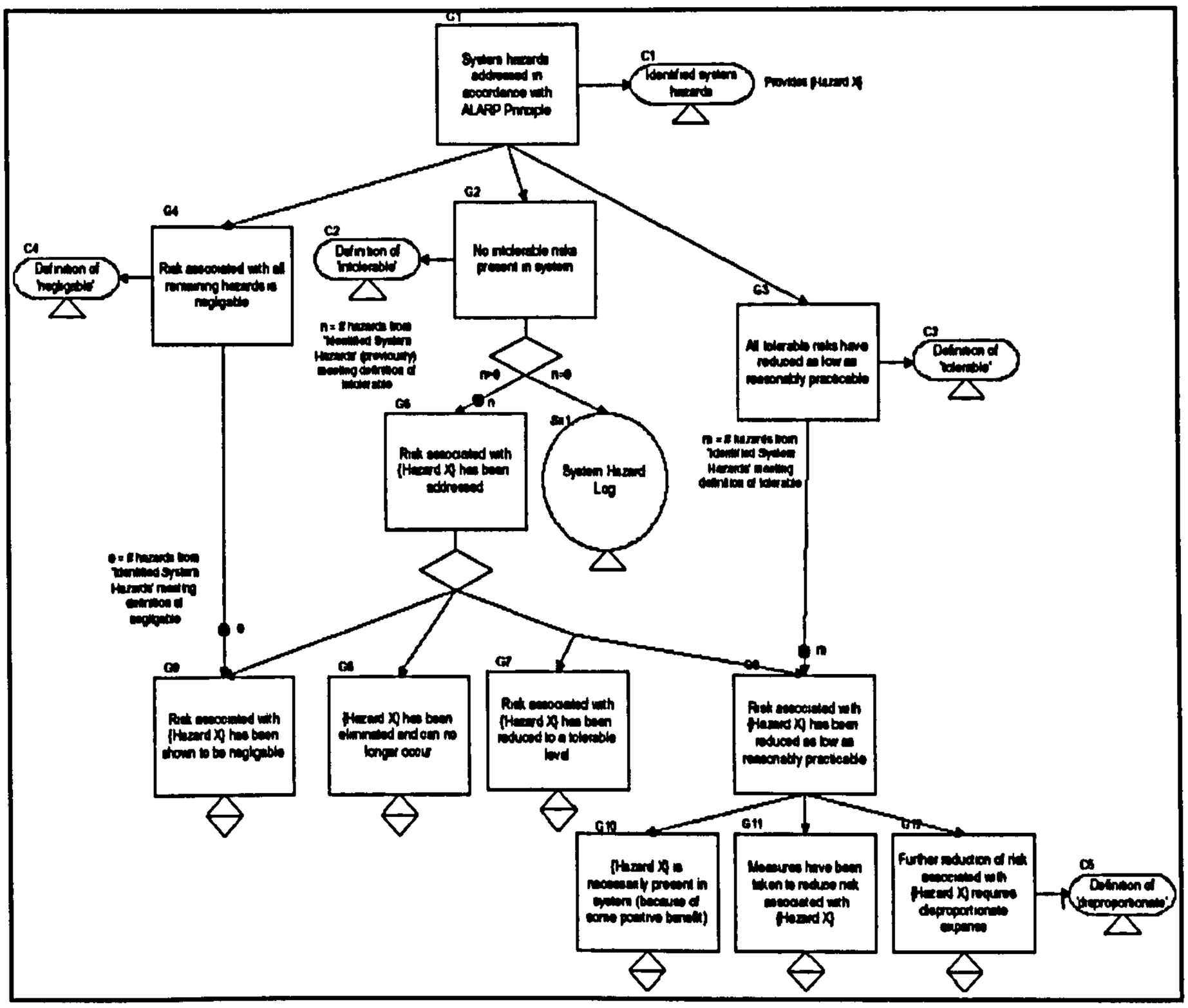

Fig.2.14 – GSN Pattern for the ALARP Principle [47]

Kelly [47] has formalised the principle by providing a pattern for ALARP arguments, based on the Goal Structuring Notation (explained in the following sections of the review). McDermid states that especially for software, ALARP cannot be quantified and it would be better to look for a qualitative approach: "*Perhaps more realistically, the above says that we cannot apply ALARP in a quantified manner and we should instead look for qualitative arguments to decide when risk has been reduced ALARP.*" [48]. This statement, along with Prasad's conclusions provide strong indication that arguing the relation and the tradeoffs between the dependability attributes should be based on qualitative approaches.

2.5 Dependability Cases

Standards such as the UK Defence Standards 00-55[2] and 00-56 have an explicit requirement that systems should be accompanied by a safety case, communicating a comprehensible argument that a system is acceptably safe in a given operational context. However there are no standards requiring a dependability case. The only association of dependability case with a standard is by Froome and Jones [49], who describe the dependability case as a possible supportive document for a system developed with the IEC-61508.

Maxion suggests that "*dependability cases comprise an organising framework and methodology for thinking about exceptions and the conditions under which they occur*" [50]. In this definition, the author identifies the association between assurance regarding the dependability and system failures. The definition identifies the need to understand exceptions. This refers to Laprie's statement that in order to achieve dependability, faults in a system must be understood and controlled. However, he does not provide any guidance on how to structure arguments regarding the attributes of dependability.

Further work by Maxion identifies the dependability case as a basis for reasoning about the "dependability" behavioural characteristics of a system stating that: "...*a dependability case is a documented body of evidence that provides a convincing and valid argument that a system is adequately dependable for a given application*" [51].

Established practice regarding reasoning about a system involves well defined and researched methods in the area of safety cases, which can be found in a variety of industries (e.g. nuclear, defence). However safety cases represent only one aspect of dependability. There are examples of standards and practices that require or propose a 'case' regarding the rest dependability attributes.

[2] 00-55 has been superseded by 00-56 and at the time of writing up this thesis has been made obsolescent. However it is a good source for some of the principles used in this thesis and their rationale.

2.5.1 Safety Cases

Adelard define Safety Case as "*a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment*" [52]. This definition is almost identical to the definition given by Maxion for dependability cases. Kelly having analysed numerous defence and public standards concludes that: "*A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context*" [47].

Safety cases have been used both for public and military projects. Guidance on railway [53] safety cases underlines that the aim of a safety cases regarding public transport systems should:

1. Give confidence that the operator has the ability, commitment resources to properly assess and effectively control risks to the health and safety of staff, contractors, passengers and public.

2. Provide a comprehensive core document, with links to other more specific documents, rules and procedures against which management and the department can check the accepted risk control measures and the health and safety management systems that have been properly put into place and continue to operate in the way originally intended.

The Adelard Safety Case Manual presents the safety case structure as a set of claims which, using an argument, are supported by evidence (Fig.2.15). An argument represents a set of inference refining the original claim. The argument can be deterministic (true/false claims) or probabilistic (e.g. MTTR, MTTF). DEF STAN 00-55 provides a required list of contents that a safety case should have to comply with the standard. Initially safety cases were text based. However especially for large systems a text-based arguments can often be difficult to understand [47]. This has led into enhancing the text based argument with tabular based arguments (e.g. 00-55), and graphical notations (e.g. GSN [47]).

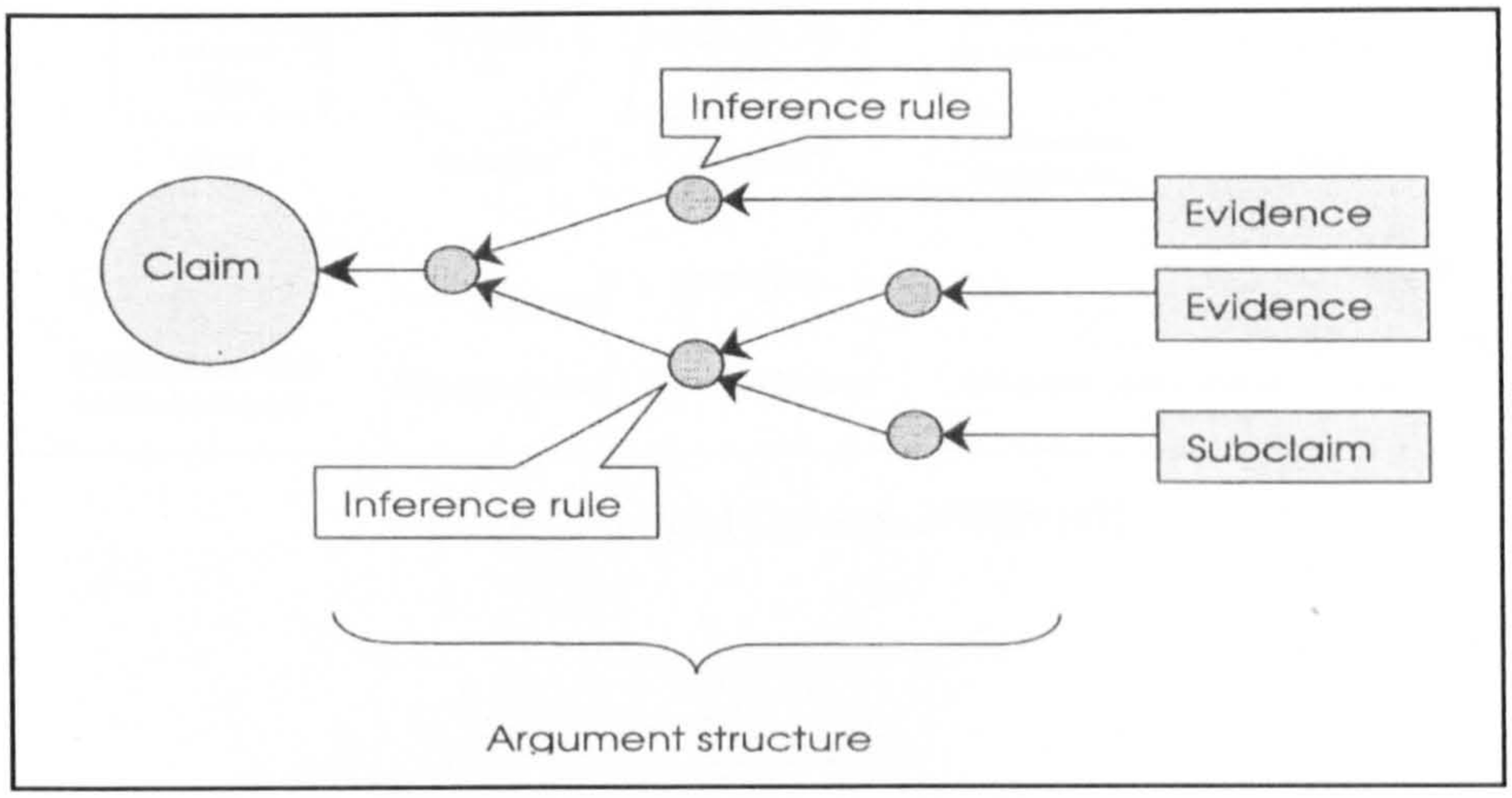

Fig.2.15 – Safety Case Structure [52]

The Goal Structuring Notation (GSN) explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument) [47]. The main symbols of the notation are shown in Fig.2.16.

The principal purpose of a goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach (assumptions, justifications) and the context in which goals are stated (e.g. the system scope or the assumed operational role. The Goal Structuring Notation (GSN) [54], considerably improved the expressiveness of safety cases, as it provided a graphical way of representing the elements of the case. Fig.2.17 presents an example of a goal structure arguing the fault free implementation of an industrial control system.

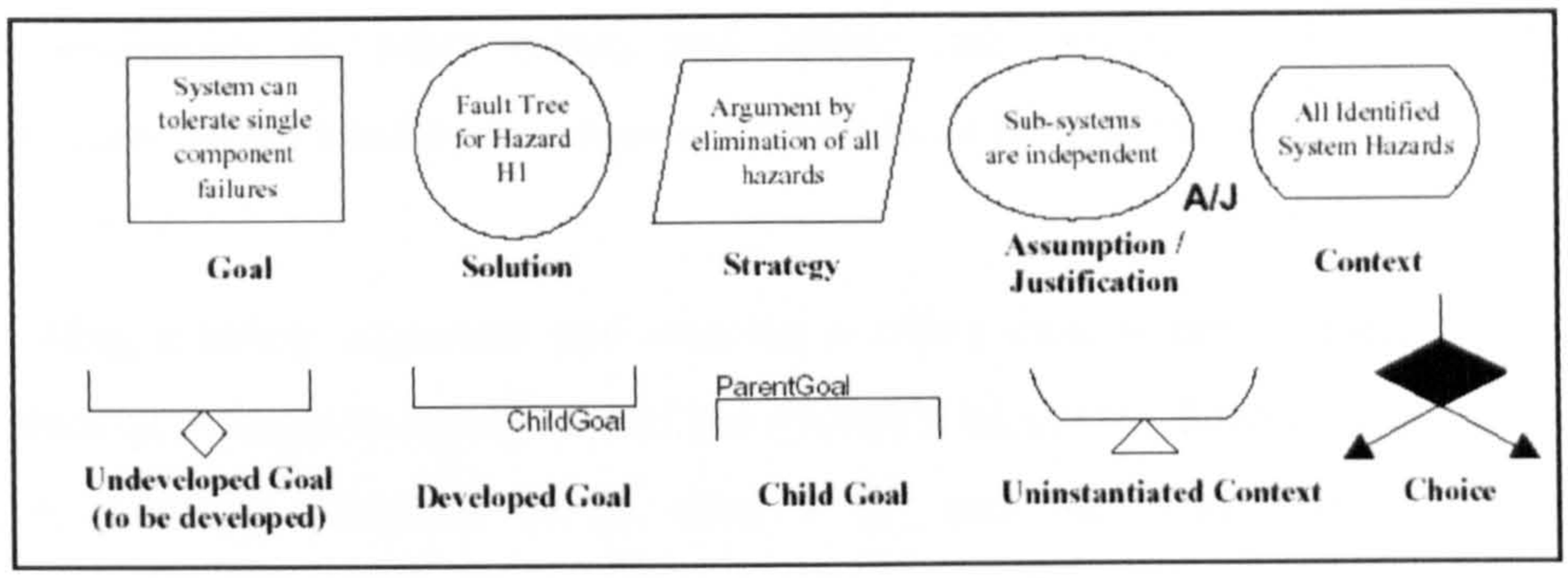

Fig.2.16 – Principal Elements of GSN [47]

Fig.2.17 – 'Control System' Example GSN Argument

In order to manage complex safety cases – in which there are complex relationships between arguments – the principles of compositional, modular, safety cases have already been established [55]. In this approach, the safety case for an overall system can be divided into a number of modules – containing the separate arguments and evidence for different aspects of system safety. For example, for a complex avionics platform the overall safety case can be reasoned about as the composition of separate arguments for each of the separate avionics subsystems. However, as described above for the dependability case, these arguments cannot be reasoned about in isolation. For example, it may only be possible to argue about the safety of one avionics subsystem in the context of assumed safe behaviour of another. To help manage the relationships that exist between safety case modules the concept of modular safety case interfaces (defining clearly the objectives, evidence, and assumed context of the case together with

any dependencies on other cases) and safety case contracts (recording how the dependencies between safety cases are resolved) have been defined.

Establishing a safety argument and creating a safety case is not a single process that takes place at a single defined point of the system's lifecycle. Safety standards, such as the U.K. Defence Standard 00-56 issue 4 [8] and the Ship Safety Management Handbook JSP430 [12], require that safety case development be treated as an evolutionary activity that is integrated with the rest of the design and safety lifecycle. For example, Defence Standard 00-56 states that: "*The safety case should be initiated at the earliest possible stage in the safety Programme so that hazards are identified and dealt with while the opportunities for their exclusion exist*".

In addition, JSP430 specifies that at least three versions of the safety case should be constructed:

- Preliminary safety case – After definition and review of the system requirements specification.

- Interim safety case – After initial system design and preliminary validation activities.

- Operational safety case – Prior to in-service use, including complete evidence of requirements satisfaction

At each stage of the evolution of the safety case, the safety argument is expressed in terms of what is known about the system being developed. At the early stages of project development the safety argument is limited to presenting high-level objectives, as design and safety knowledge increases during the project these objectives (and the corresponding argument) can be expressed in increasingly tangible and specific terms (as depicted in Fig.2.18).

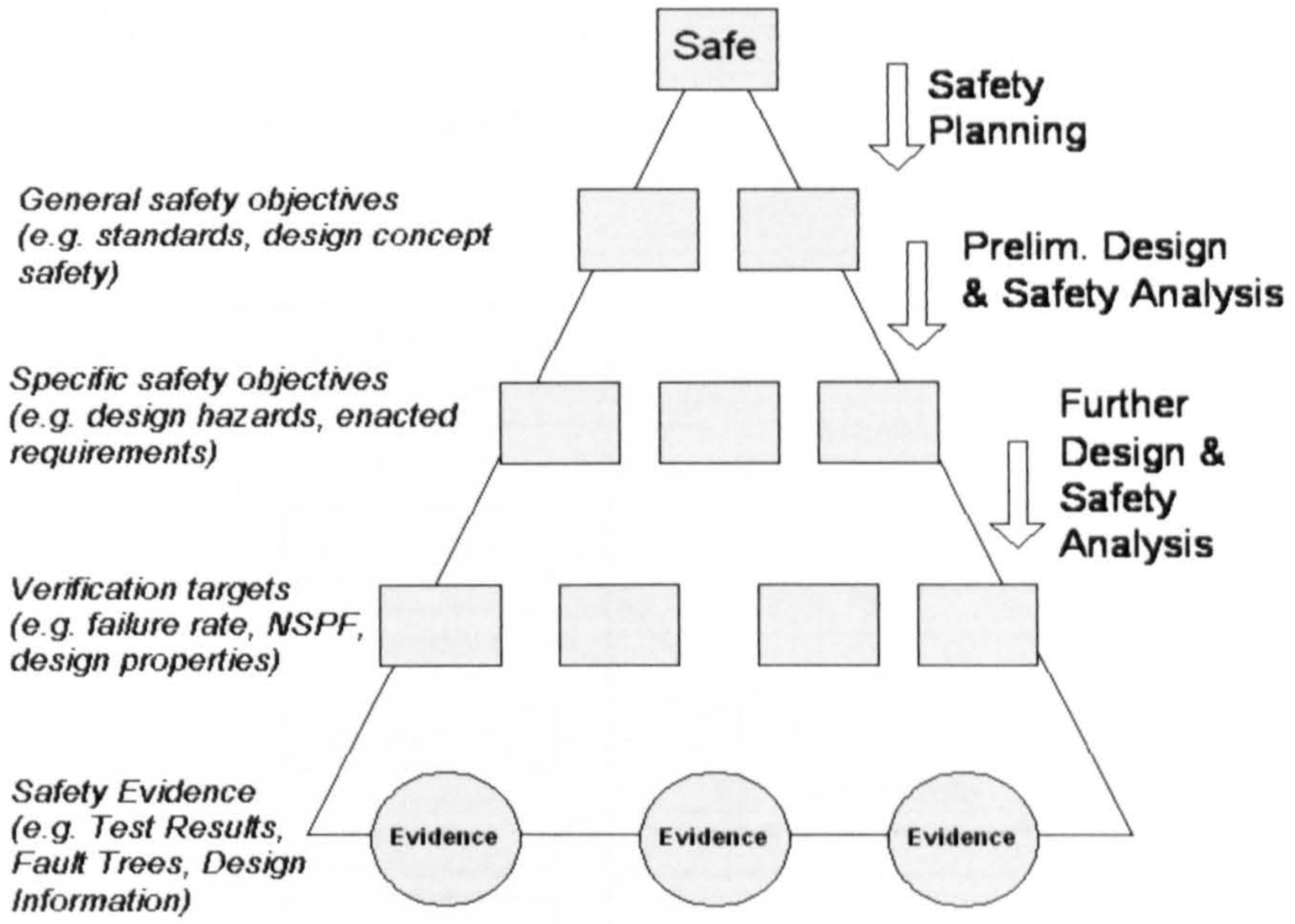

Fig.2.18 – Evolution of an Argument

Although safety cases focus only on one dependability attribute, often there are relations identified to other attributes. However the rest of the attributes have a secondary role acknowledged only if they affect the system's safety goals.

2.5.2 Reliability and Maintainability Cases

Defence Standard 00-40 requires the construction of a reliability and maintainability case. As defined by the standard, a R&M Case is "a reasoned auditable argument created to support the contention that a defined system satisfies the R&M requirements: *"...is also produced progressively during a project life cycle and will typically be summarized in a R&M case document in the end of the phase"* [56].

The R&M case is required to provide assurance that the R&M requirements of the systems have been met. The standard requires the case to be reviewed and updated in case:

- The system is modified
- The context of its operation is modified.
- The R&M requirements are modified,
- If there is deviation between the actual and intended performance

Fig.2.19 presents the elements of a R&M case.

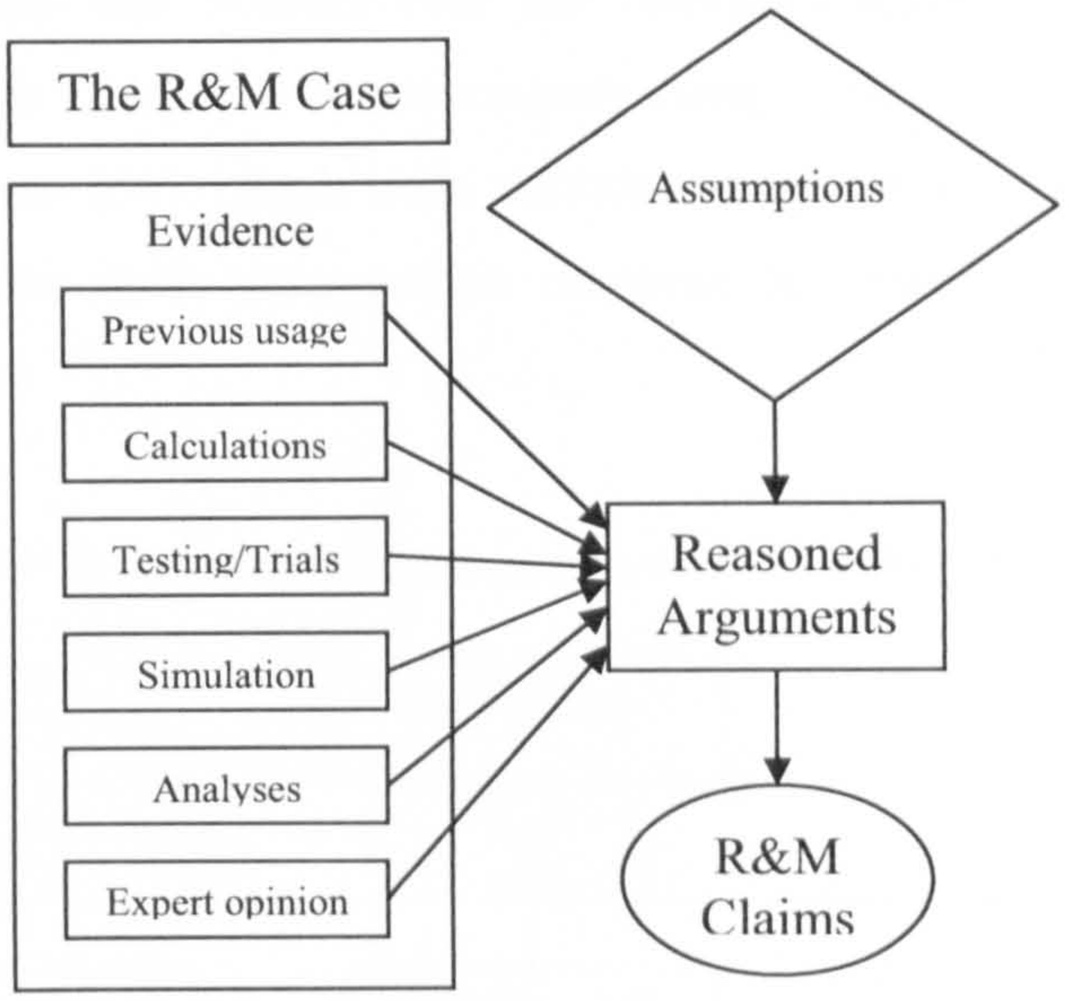

Fig.2.19 – Elements of the R&M Case [56]

An important aspect of the R&M case is that evidence of the R&M case can also be used as input to the safety case, as suggested by the Defence Standard (DStan) 00-42 [57]. As already described dependability attributes are not independent to each other. A reliability or performance failure may lead to a safety or security failure. For example consider an ATC SoS delaying processing of aircraft data. Hence claiming achievement of acceptable safety or security may depend on arguments regarding the reliability or performance of the system. Collaboration between cases representing different system attributes is essential in order to establish assurance about the overall dependability of a system.

2.5.3 Security Cases

It is widely accepted that security is a composite attribute consisting of aspects such as reliability, availability and confidentiality. Security failures are usually related to malicious attacks from persons that deliberately attempt to exploit system vulnerabilities.

The concept of security case has not been explicitly defined similarly to safety or the R&M case. However, there are attempts to provide an argument of security. An example is security Methodically Organised Argument Trees (MOATs) which are "…used to document and communicate the assurance argument that establishes a security property for a system under consideration" [58]. MOATs are based on hierarchical fault trees [59]. Each node contains an assurance claim and the interior nodes document how these assumptions compose to establish the required claim (Fig.2.20).

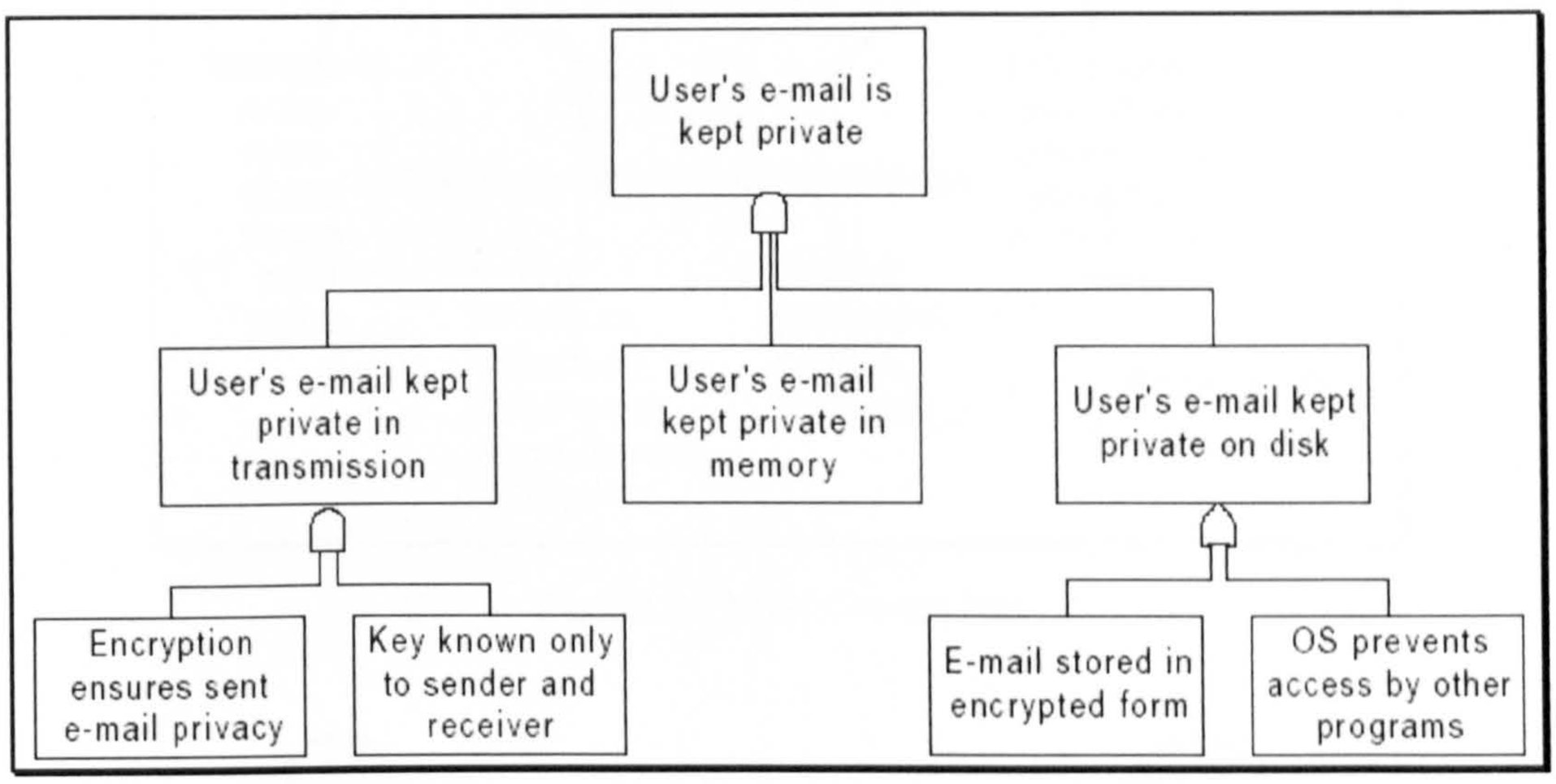

Fig.2.20 – A Security MOAT [58]

The goals are identified based on a security risk analysis of the system, evaluating the consequences of possible security failure. The risk analysis provides the goals the system must satisfy, in the same sense as a hazard analysis. MOATs are accompanied by a methodology in order to produce an effective and valid argument concerning the security of a system.

Other approaches include the Network Visual Rating Methodology (NVRM) developed by the US naval research laboratory. NVRM is derived from GSN providing a set of "decomposable" goals that using a justified strategy are supported by evidence. *"The VNRM is a toolset and language for developing and evaluating a map of an argument that mission critical information is adequately protected by a system in its larger operational environment"* [60].

The methodology combines the three attributes of security (confidentiality, integrity and availability), as well as four security disciplines (Physical, Technological, Operational, Personnel) in order to compose an effective argument about the security (Fig.2.21).

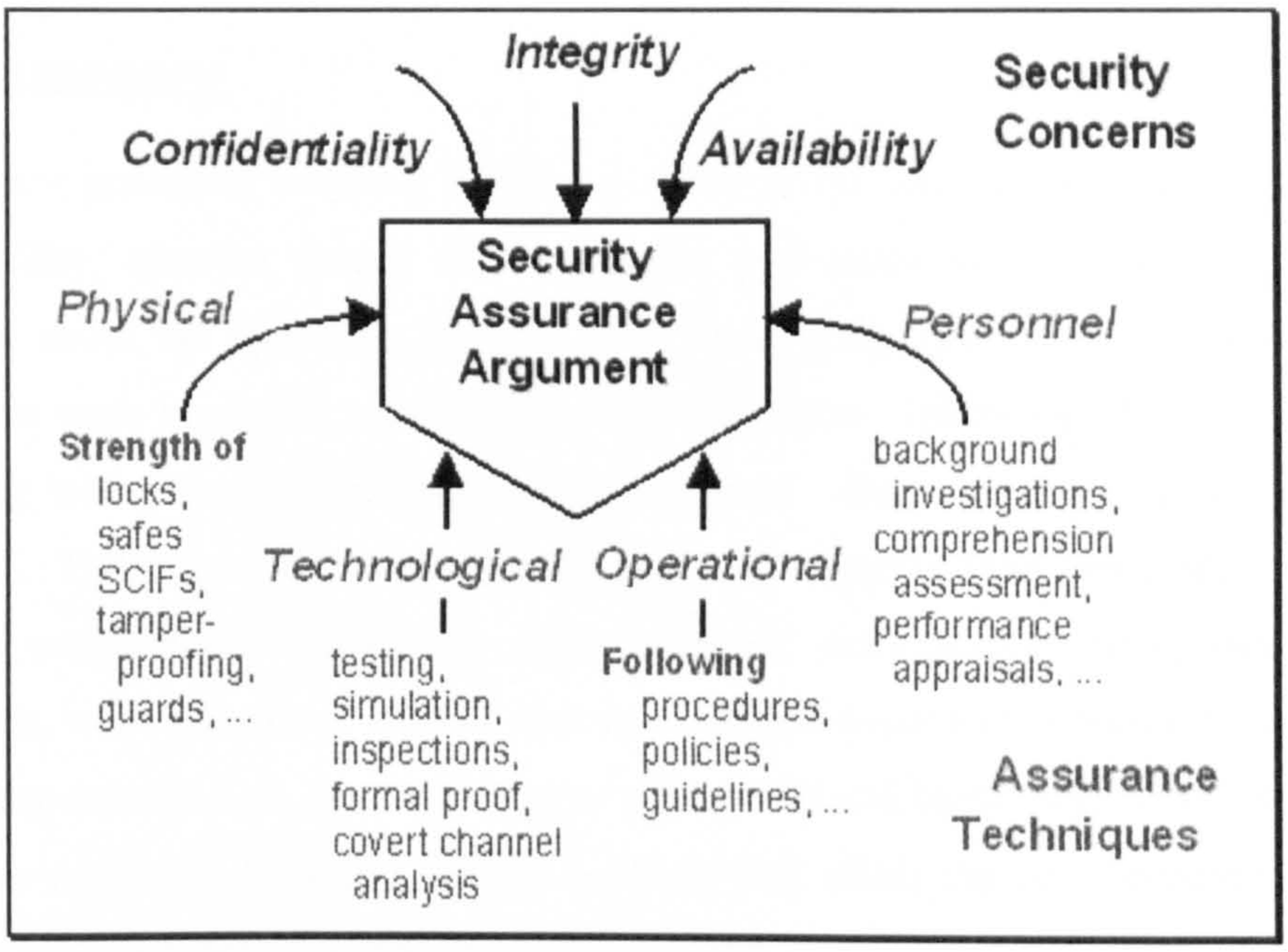

Fig.2.21 – VNRM Concept [60]

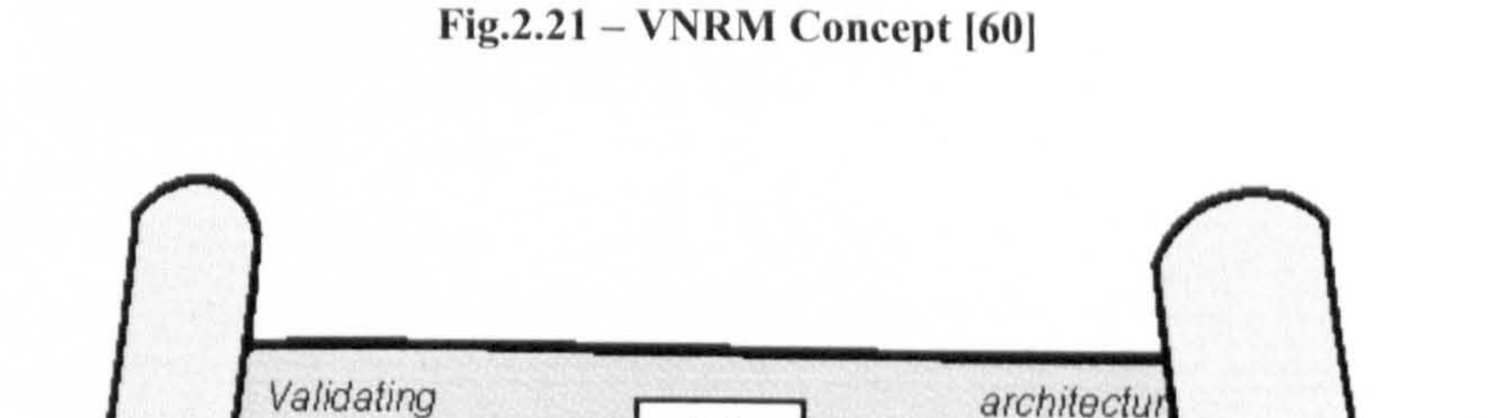

Fig.2.22 – A VNRM Map Combining Fault Trees and GSN [56]

VNRM constructs argument maps (Fig.2.22), which contain claims interrelated via dependencies, assumptions and design decisions. Moreover, sub-claims of a map can

have their own argument in a separate map in a style similar to modular GSN. An important feature of VNRM is its ability to show the associations between different claims based on design decisions.

2.6 Summary

The chapter presented a survey of related work in the areas of Systems of Systems, dependability, systems design and trade-offs, and cases as a means of achieving assurance about the operation of a system. Concerning Systems of Systems, several definitions were presented, each having its own unique viewpoint. Moreover certain modelling methods and frameworks were reviewed. Dependability was another topic examined. Definitions are not always aligned, each approaching dependability from a different perspective. However, some common assumptions, upon most of the definitions, were identified. Finally system assurance cases were reviewed. In order to make a dependable SoS, the development process should be structured, and we should be able to determine how any design decisions will affect the SoS behaviour. Safety cases are the most mature and widely used concept.

Intentionally Blank

Chapter 3

Establishing a Dependability Case Framework

In the previous chapter a number of concepts were reviewed suggesting the creation of an argument about dependability attributes such reliability, security and safety. However, little has been done about integrating all dependability attributes in a single case. This chapter lays the foundations for integrating dependability attributes by establishing a dependability case framework. Firstly, the characteristics of a dependability case are discussed. Then, key challenges are identified introducing the work done in this thesis to address them. Finally, the chapter presents the technical approach undertaken in creating a dependability case metamodel, rigorously defining the concepts and their associations. The dependability case metamodel is used throughout the thesis to clearly present the concepts of each of the proposed methods.

3.1 Dependability Arguments

Although the dependability case is a relatively new and untested concept, the idea of developing 'cases' for system attributes other than safety is not unprecedented. Maintainability cases are a requirement of the U.K. Defence Standard 00-40 [56]. Defence Standard 00-40 defines the reliability and maintainability (R&M) case as *"a reasoned auditable argument created to support the contention that a defined system satisfies the R&M requirements"*. The Common Criteria for security [61] suggest a document explaining why a system has met its required security level. Similarly, the US Naval Research Laboratory whilst not explicitly using the term security case; have described the development and evaluation of *" a map of an argument that mission critical information is adequately protected by a system in its larger environment"* [60]. In all cases the argument communicates how the available evidence can support an overall claim about the acceptable behaviour of the system regarding the respective attribute. Table.3.1 shows typical claims, arguments and evidence required when reasoning about different dependability attributes.

Table.3.1 – Claims, Arguments and Evidence for Dependability Attributes

Attribute	Overall Claim	Typical Argument	Typical Evidence
Safety	System is adequately safe	Hazard mitigation argument	Hazard Analysis, Causal analysis
Reliability	System meets reliability requirements	Adequate redundancy, resilience of components	Testing / Simulation, Markov analysis
Maintainability	System meets maintainability requirements	Modular cohesive design, ease of installation, ease of replacing components	Expert opinion, simulation.
Security	Mission critical information is adequately protected	Assets protection argument	Access control, policies, MOATs[3]

Despite the fact that there are attempts to create cases for other attributes, all of them focus on a single dependability attribute. Also, none of them have been the subject of as extensive research and development (e.g. tool support) as safety cases.

3.2 Dependability Attributes and Non-functional Requirements

It is common for the dependability attributes to be characterised as non-functional requirements (e.g. Robertson's classification of non-functional requirements [62]). This classification separates the concerns of the stakeholders into functional and non-functional, with the former describing what the system has to do, and the latter describing properties of the system such as safety, performance and usability.

However this distinction between functional and non-functional requirements is not always clear. Non-functional and functional requirements can be related to each other. For example, in order to achieve a safety requirement, further functionality may be needed that contributes to increasing the safety levels of the system, such as the functionality provided by TCAS which mitigates failure to maintain the prescribed aircraft separation. Furthermore although to some degree dependability properties are

[3] Methodically Organised Argument Trees (MOATs) are described in chapter 2

inherent, their implementation to acceptable levels will require certain design characteristics. For instance, it is common practice to ensure the availability of a function by adding redundant components that provide the function. Hence, establishing acceptable dependability levels can have impact on both the implementation and the design/architecture of a system. Therefore it is often the case that such attributes may have to be planned from the beginning of the system development. This was recognised by Prasad [10] and Bass et al. who also add that *"you can't get functionality right and then go back and put in qualities. They have to be designed from the start"* [63].

Dependability can be thought of as a composite system property describing the system's behaviour with respect to different viewpoints, with ultimate objective the reliance on the system's operation. A dependability case communicates assurance about acceptable operation of the system. According to Prasad's (more generic) definition, which has been adopted in this thesis, dependability entails any requirement that the stakeholders perceive to be important with regard to their interests.

A key challenge is to maintain the multiple attributes of dependability at acceptable levels, addressing the achievement of each attribute in context of the others. Engineering practice has shown that it is impossible to achieve all dependability requirements without compromise. Satisfaction of the requirements depends on design decisions during system evolution and contextual information about the operation of the system. This task requires definition and justification of clear levels of acceptability for each dependability requirement, as well as traceability of the requirements' rationale throughout the entire lifecycle.

3.3 The Role of Argumentation in System and Requirements Evolution

A safety case exists to communicate an argument. It is used to demonstrate how someone can reasonably conclude that a system is acceptably safe from the evidence available. A safety case is a device for communicating ideas and information, usually to a third party (e.g. a regulator). In order to do this convincingly, it must be as clear as possible. The safety argument is that which communicates the relationship between the

evidence and objectives. Both argument and evidence are crucial elements of the safety case that must go hand-in-hand. An argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without an argument is unexplained; it can be unclear that safety objectives have been satisfied [47]. Although the argument of a case can be presented in a textual form, experience has shown that this is inefficient, often resulting in weak arguments that are hard to comprehend. For this reason the Goal Structuring Notation (GSN) was introduced to structure and present clear and comprehensible cases [47], which was described in chapter 2.

The Goal Structuring Notation is used in a dependability case to structure the argument about acceptable fulfilment of the system's dependability goals. In most goal-based requirements engineering (RE) methods, goals represent something that a stakeholder hopes to achieve in the future [64]. Often in RE, Goals refer to something general and abstract that cannot be directly verified, as opposed to requirements that can be validated and verified by the end of a system's development lifecycle. Thus goals need to be refined until they can result in specific requirements. However, in a GSN goal structure, goals are phrased as propositions. This means that they are statements that can either be *true* or *false*.

Evolving in parallel to the system development, goals are decomposed until they can be directly supported by evidence collected during the development and testing phases of the system. GSN goals are specific claims that a system has achieved a particular requirement. Being able to explicitly represent and associate all the elements of an argument, GSN helps to articulate post-conditions for the initially identified requirements of the system in question. In a dependability case, GSN is used to create arguments supporting claims of sufficient achievement of the dependability attributes. Fig.3.1 presents a dependability adaptation of the evolution of a safety argument illustrated in Fig.2.18. Fig.3.1 shows how a high level dependability claim can be decomposed during development of the system. At each stage of the evolution of the safety case, the dependability argument is expressed in terms of what is known about the system being developed. At the early stages of project development the dependability argument is related to the high-level objectives as conceived by the stakeholders in the concept of operations (CONOPS). As design knowledge increases

during the project, these objectives (and the corresponding argument) can be expressed in increasingly tangible and specific terms.

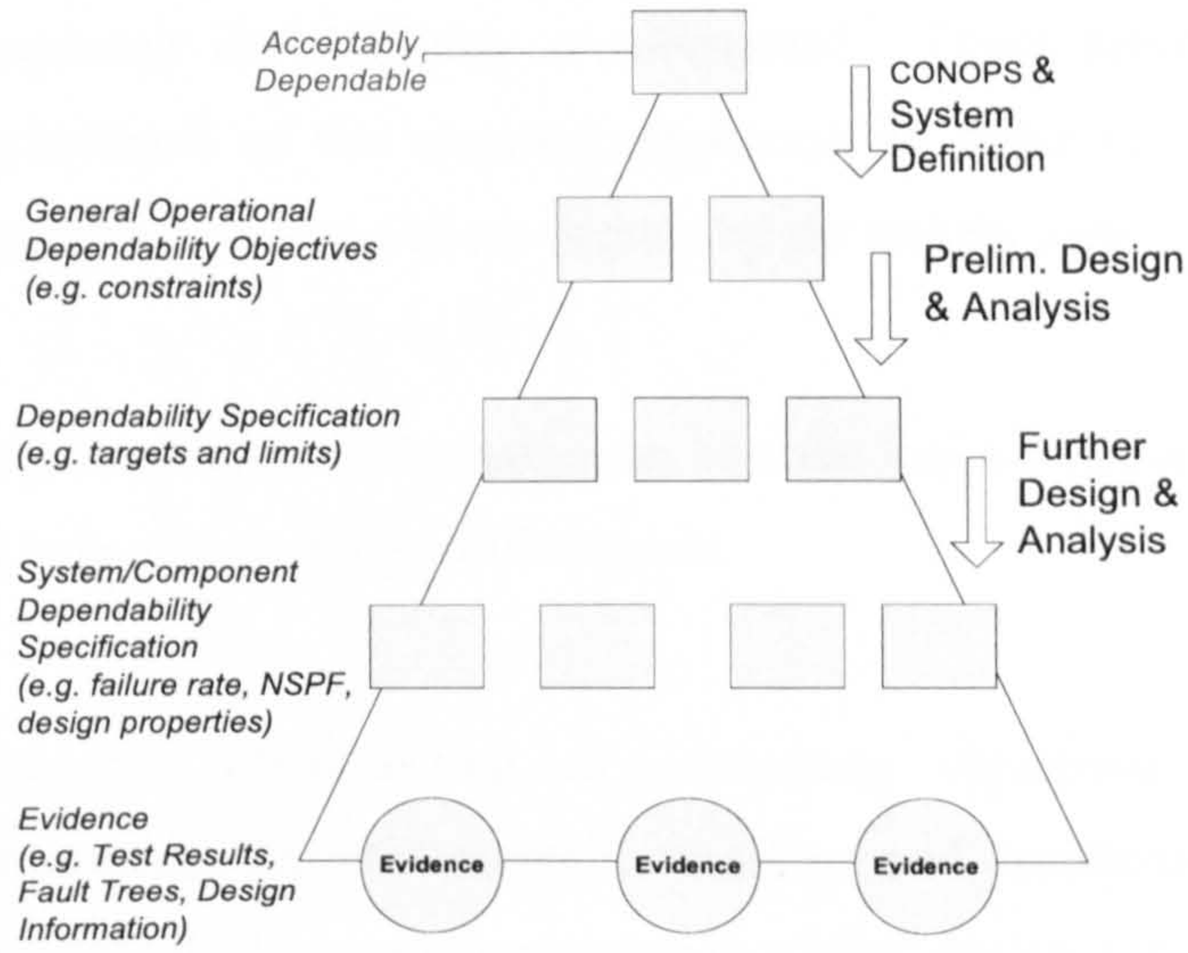

Fig.3.1 – Evolution of a Dependability Argument

Analysis identifies the required behaviour of each component of the system in order to satisfy the overall dependability goals that were identified by the stakeholders. The goals are decomposed in context of the system as the latter evolves, which includes specification of requirements and design decisions that will eventually affect the final structure of the dependability argument.

3.4 Creating and Capturing Argument Context

One of the merits of GSN is its ability to explicitly capture the context in which the claims of an argument are stated. In GSN context can be captured in two different forms. The first is by using the context element, which can contain a reference or a statement of contextual information. For example, a goal in a safety case claiming acceptable safety can be stated in the context the definition of acceptable safety. A reference to contextual information can point to resources available from other processes such as system models or information available in other documents. The second type of context is represented by GSN (argument) modules. The argument is stated in the context of the top level claim of the GSN module which is supported by argument and evidence. In this way the main dependability argument can be decomposed in the context of other decisions or processes which need to be argued.

The contextual information which affects the evolution of the dependability argument and should be captured in the dependability case is the following:

- *Component dependability requirements*: These represent the (derived) requirements of the system components in order to maintain an overall acceptable behaviour as envisioned by the stakeholders.

- *Design rationale*: Information on how the various characteristics of a design can help achieve the specified goals.

- *Trade-offs*: Identification of competing objectives documentation of compromises within acceptable margins and justification of selection of most suitable design option among candidate design alternatives.

Contextual information, whether represented by the GSN 'context' element or as GSN modules, are created by three methods:

- *Dependability Deviation Analysis (DDA)*: DDA (described in chapter 4) provides a systematic analysis of the system identifying the effects of deviations from the normal operation of the system. In the context of dependability cases, the purpose of DDA is to elicit acceptable behaviour of the system with respect to dependability and identify associations between failures under the viewpoints of each dependability attribute.

- *Trade-Off Method (TOM)*: TOM (described in chapter 5) describes a methodical way of using the established space of admissible requirements to trade-offs goals and resolve conflicts. TOM ultimately creates arguments of preference between candidate decisions, in the context of which the dependability argument evolves.

- *Factor ANalysis and Decision Alternatives (FANDA)*: FANDA (described in chapter 6) complements GSN in developing arguments. The method can be thought of as a catalyst between the argument and the design. It assists analysts

to examine the system goals, record and manage rationale, and elicit candidate options for decisions taken during the system lifecycle, such as design decisions.

The three methodologies in addition to the existing GSN method compose the overall dependability case framework presented in this thesis. The methodologies are not independent to each other but are associated and collaborate during the development of the case. In order to provide a clear and rigorous description of the framework the Dependability Case Metamodel (DCM) has been created, capturing the concepts, their attributes, their legitimate associations as well as constraints that need to be applied when creating a dependability case instance.

3.5 Rigorous Definition of the Framework

At present, efforts to formalise cases, and in particular argumentation, are largely tool-driven and mostly limited to the safety community (e.g. The ASCE tool [65]). Existence of a rigorous metamodel that captures the concepts existing in the assurance cases domain in an open and standard format, can deliver a number of benefits. Most importantly, it offers a common vocabulary and consensus on the concepts involved in the task of assurance case development and their semantics. This can contribute to avoiding 'deviant' implementations across tools from different vendors. Moreover, evaluation of a tool implementation can be difficult as there is no common point of reference. Also, a uniform serialization format can provide a tool-independent platform to facilitate information exchange between different vendors' tools. Finally, establishing a dependability case metamodel can provide the basis for further introduction of concepts such as metrics management.

Fig.3.2 shows the technologies employed in defining the DCM as well as their use in defining and managing dependability case models (based on the DCM). Initially the metamodel was defined using the Kernel MetaMetaModel (KM3) modelling language. Using tools built in Eclipse (a software and modelling development tool) [66], the KM3 metamodel was transformed to an Ecore metamodel. Ecore is an implementation of the Object Management Group (OMG) standard Meta Object Facility (MOF) 2.0 metamodelling architecture. Using Eclipse's EMF visual editor it is possible to

instantiate the defined dependability case metamodel, and create and edit new models (each created model is an instance of a dependability case).

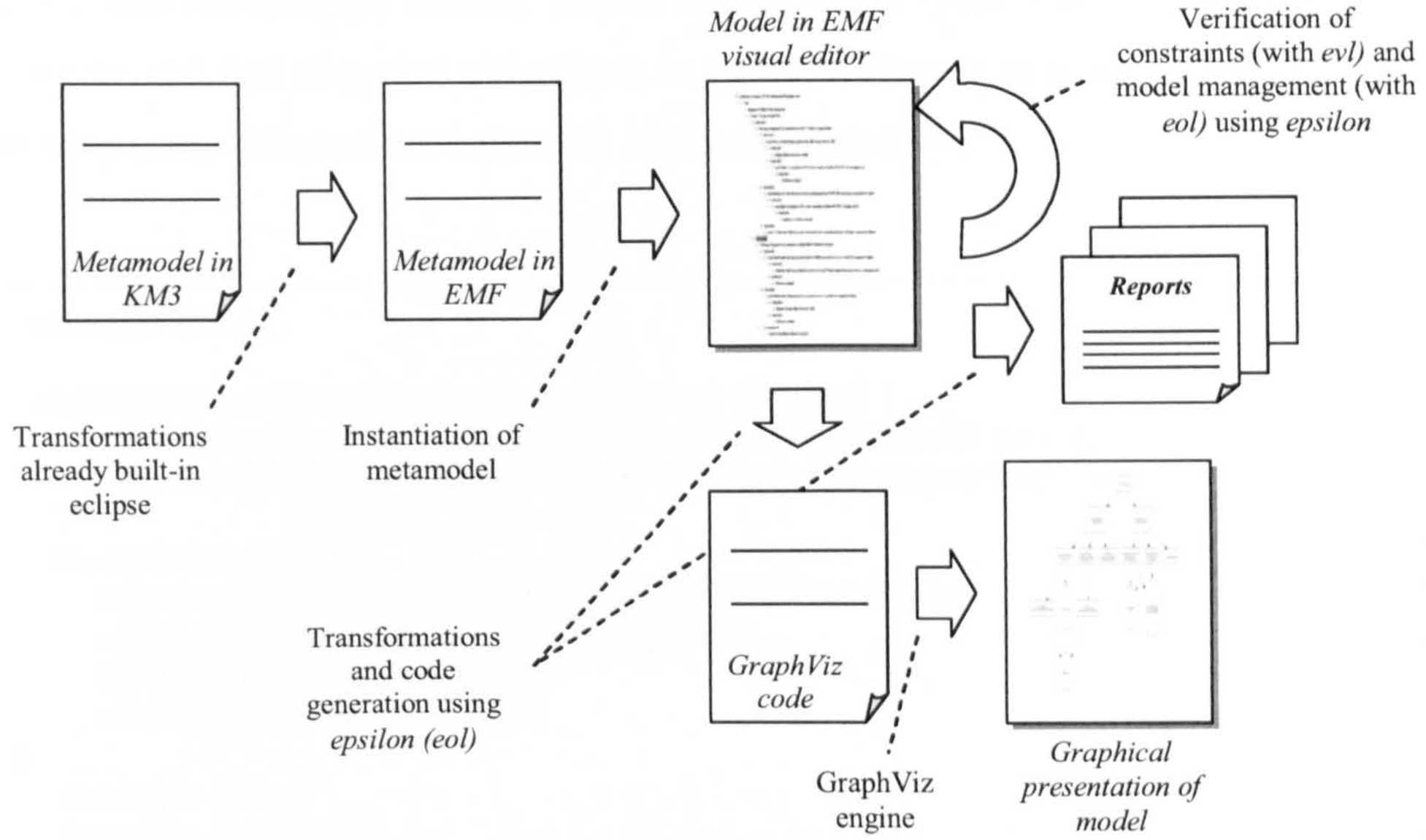

Fig.3.2 – Technologies Used in Dependability Cases

Epsilon [67] is a platform consisting of a number of model management languages which were used to manage and verify the created model. In particular, the languages used were Epsilon Object Language (EOL) and Epsilon Validation Language (EVL). The implemented functions included among others, the automated generation of GraphViz [68] code, a package used to create graphical representations of the models, automated production of reports and logs of the dependability case and automated support for the proposed methods.

3.5.1 Definition Using Kernel MetaMetaModel (KM3)

KM3 is a modelling language used to define metamodels [69], employed by the author to define the dependability case metamodel. KM3 was selected mainly for its ease of use. KM3 consists of a relatively (to other languages) small number of classes, and its simple syntax allows straightforward definition and editing of the metamodel. Moreover the formally defined semantics make verification of the metamodel possible, eliminating potential errors. Fig.3.3 shows an extract of the DCM in KM3 code defining elements of GSN. The extract presents the class *SpinalElement* (goals,

strategies, solutions are all types of spinal elements), and the class *SolvedBy* which represents the decomposition of a GSN spinal element. A class can have either references to other objects, or attributes. The *oppositeOf* keyword defines bidirectional associations between two classes. Hence by selecting a spinal element we can navigate the model and find its parent and objects (e.g. parent strategy or goal). A spinal element can have many references to solved by objects, denoted by *[*]*.

```
package GSN {

 abstract class SpinalElement extends ModelElement {
     reference solvedBy [*] container : SolvedBy oppositeOf parent;
     reference inContextOf [*] container : InContextOf oppositeOf parent;
   }
   class SolvedBy {
     reference parent : SpinalElement oppositeOf solvedBy;
     reference child container : SpinalElement;
     attribute cardinality : Integer;
     attribute optional : Boolean;
}
   datatype String;
   datatype Boolean;
   datatype Integer;
}
```

Fig.3.3 – Extract of GSN in KM3

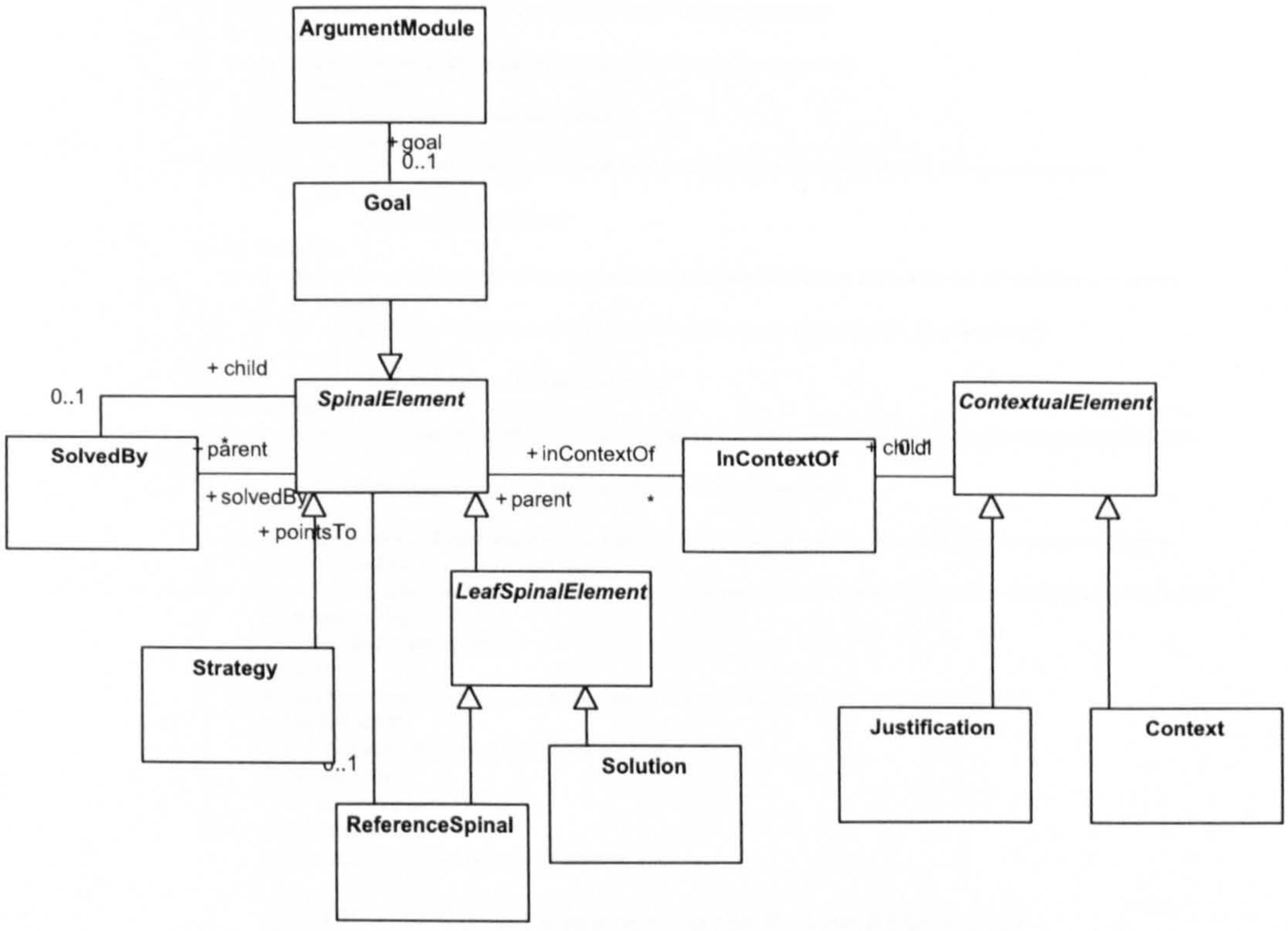

Fig.3.4 – UML Modelling of Basic GSN

Using EOL, the KM3 metamodel can be transformed to UML for better visualisation of the concepts. Fig.3.4 shows the metamodel of GSN (as described by Kelly in [47]), without the extensions for modularising GSN, as proposed by Kelly in [55][4]. Throughout the thesis, excerpts of the metamodel represented in UML will be used for better understanding and illustrating the relations between the concepts. The metamodel defined in KM3 can be found in appendix C.

3.5.2 Eclipse Modelling Framework (EMF)

The Eclipse Modelling Framework (EMF) is a modelling framework built on top of Eclipse, an open development framework with extensive user base. The KM3 definition of the metamodel is transformed to the Eclipse Modelling Framework (EMF) allowing the creation of instances of the metamodel, which can then be edited with the EMF editor. Fig.3.5 shows an example of the model of a GSN structure in the EMF editor (by instantiating the DCM objects representing GSN).

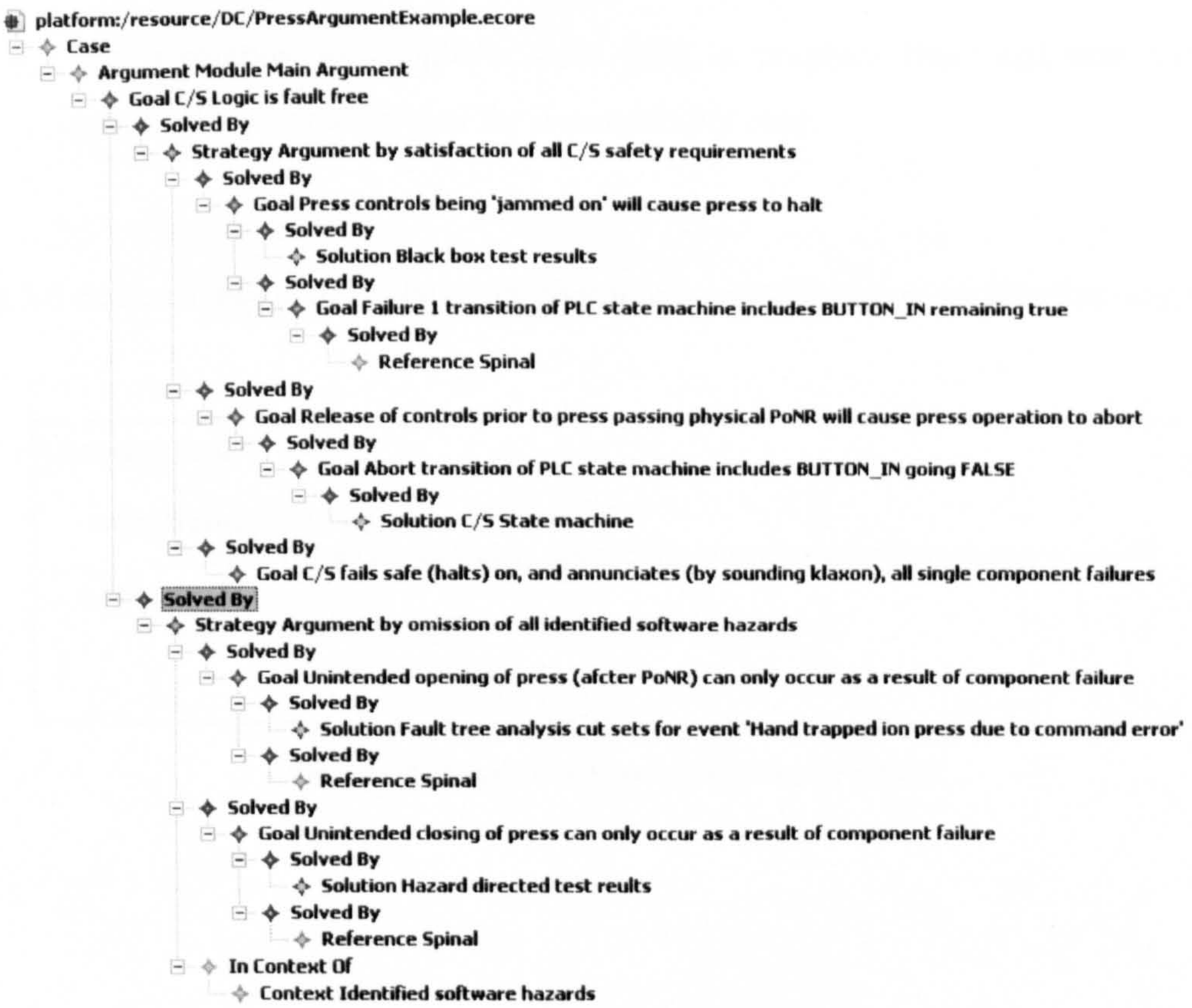

Fig.3.5 – GSN Press Argument in the Eclipse EMF Editor

[4] The modular GSN extension has been defined in KM3 (appendix C)

The editor does not show the graphical representation of the GSN elements, but it shows the objects that have been created by instantiating the appropriate (GSN) classes of the metamodel. Similarly, the rest of the concepts introduced in the proposed methodologies can be modelled by instantiating the appropriate classes.

3.5.3 Model Management Using EPSILON

EPSILON is a platform of model management languages for tasks such as model merging, model transformation and model validation [67]. It was used to manage the instantiated dependability case models. Among the purposes for using EPSILON were:

- Automating the creation of parts of the dependability case (where possible)
- Automated reporting of warnings or errors spotted in the model
- Checking the constraints of the metamodel
- Transformation to GraphViz code [68], a graphics tool, that was used to graphically present parts of the dependability case.

Fig.3.6 demonstrates a GSN constraint example using the epsilon verification language.

```
context Goal {

    constraint HasUniqueDescription :
          Goal.allInstances.forAll(g|g.description = self.description implies g = self)
     fail:
          'Goal ' + self.description + ' has not unique ID'}
}
```

Fig.3.6 – Example of Constraints Using EPSILON

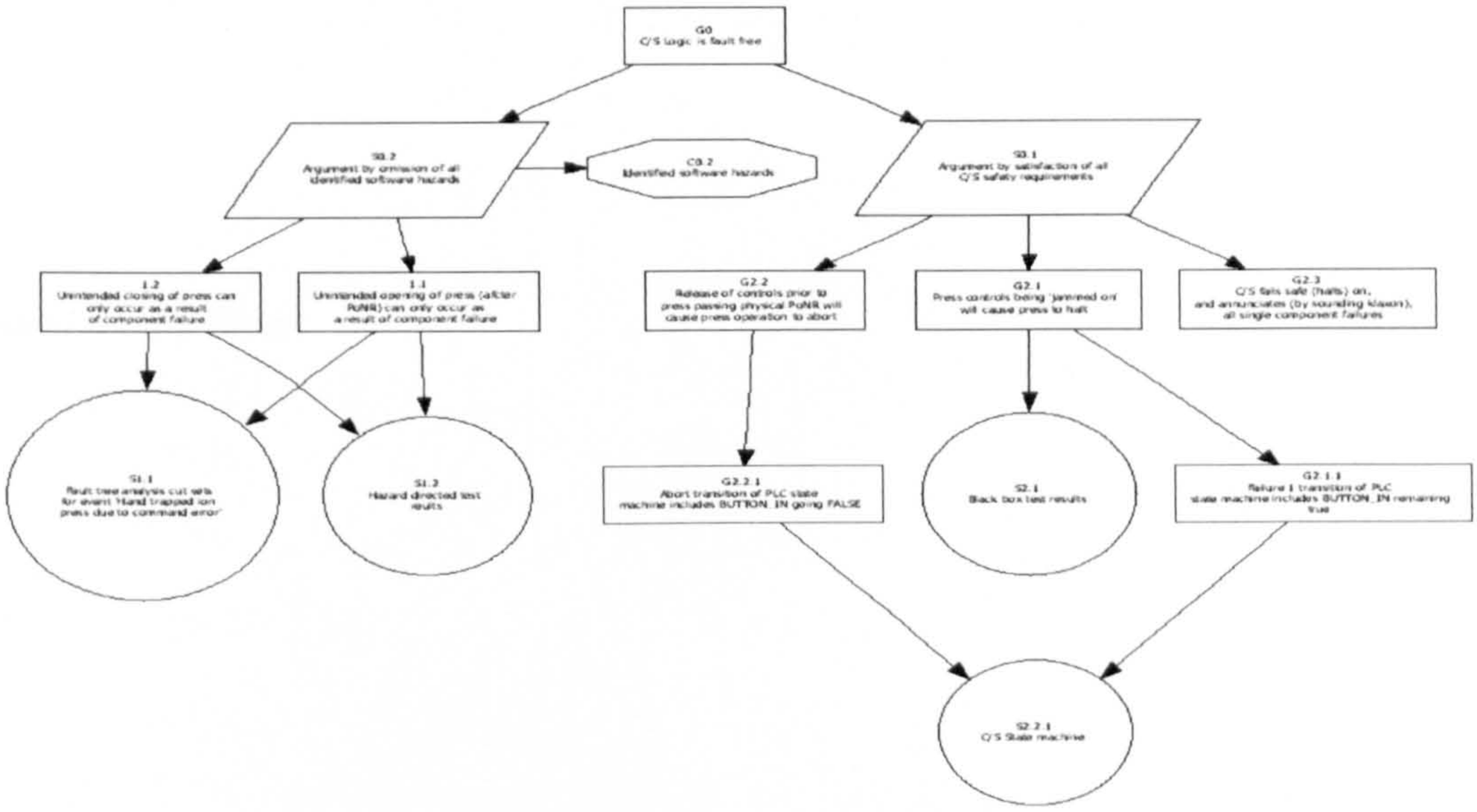

Fig.3.7 – Press Argument in GSN (produced in GraphViz)

The constraint is applied on objects of type 'Goal' and it checks whether a goal has a unique ID. The attribute *description* represents the ID of the goals. If the constraint fails the code will report an error. Fig.3.7 shows the (graphical) GSN structure of the press argument example presented in Fig.3.5after transformations using the Epsilon Object Language[5].

3.6 Summary

The chapter has defined the concept of dependability cases, identifying prominent characteristics, such as typical arguments and evidence for each dependability attribute. Also the chapter describes the fundamental concepts of the proposed work. In addition a dependability case metamodel has been defined, using the KM3 language, and then transformed to the Eclipse Modelling Framework (EMF). The metamodel can be instantiated enabling the creation of dependability case models. The Epsilon Object Language was used to create scripts that will manage the created dependability case models

[5] The graphical representation of some GSN elements (i.e. context and context association) differs from original GSN. GraphViz required definition of some of the shapes not included in its library, which was considered to be outside the scope of this thesis. The example demonstrates the application of GraphViz for future reference. However throughout the thesis GSN structures are represented in a proper manner.

Intentionally Blank

Chapter 4

Requirements Elicitation Using Dependability Deviation Analysis

4.1 Introduction

When examining the required system behaviour overall, system stakeholders can identify some overall goals for the system. However, during evolution of the system designers need to examine how their initial requirements and concerns can be related to the behaviour of the more detailed layers of the system design. In safety, requirements for a system are typically elicited through following a structured safety assessment process during each stage of the lifecycle [70].

Among other techniques and methodologies used, the safety assessment process includes deviation analysis techniques and methods such as Hazard and Operability Studies (HAZOPS) [71]. Deviation analyses are used to identify possible deviations from intended behaviour and their effect on the overall safety levels of the system. Hence, the required behaviour for the elements of the system which are under analysis is identified. Even though other attributes are recognised during the analysis such as performance, the main focus is on safety. However eliciting dependability requirements necessitates a more explicit analysis from the viewpoint of each dependability attribute. Stakeholders interested in an attribute need to identify the effects of the behaviour of the system (or an element of the system) with respect to that attribute. This entails possible effects to other attributes that are of interest to other system stakeholders. Dependability Deviation Analysis is a method for eliciting dependability requirements, by identifying how dependability attributes affect each other. Establishing the required operation of a system element in terms of the dependability attributes of interest, allows creation of a dependability profile for that system element that encapsulates all derived dependability requirements.

4.2 Deviation Analyses

Deviation analyses are commonly applied in the safety domain in order to achieve a better insight about possible safety implications of system deviant behaviour. Deviation analysis methods aim to identify the causes and effects of deviations from intended operation. Delivery of a service different to the one intended can be described as a failure. Ultimately a failure can be characterised in terms of *risk* – the severity of the outcome combined with the probability of the failure occurring.

Fig.4.1 shows a bow tie model [72] used in safety. The focus of the model is commonly a hazard, represented by the 'knot' of the tie. The bow-tie model consists of two elements: a causal model that describes the causes of the hazard and a consequence (or outcome) model that describes the effects of the hazard (alongside other contributing factors). Deviation analyses are used to explore variations around the intended behaviour of a system design and identify deviations with unsafe consequences (i.e. 'hazardous' deviations).

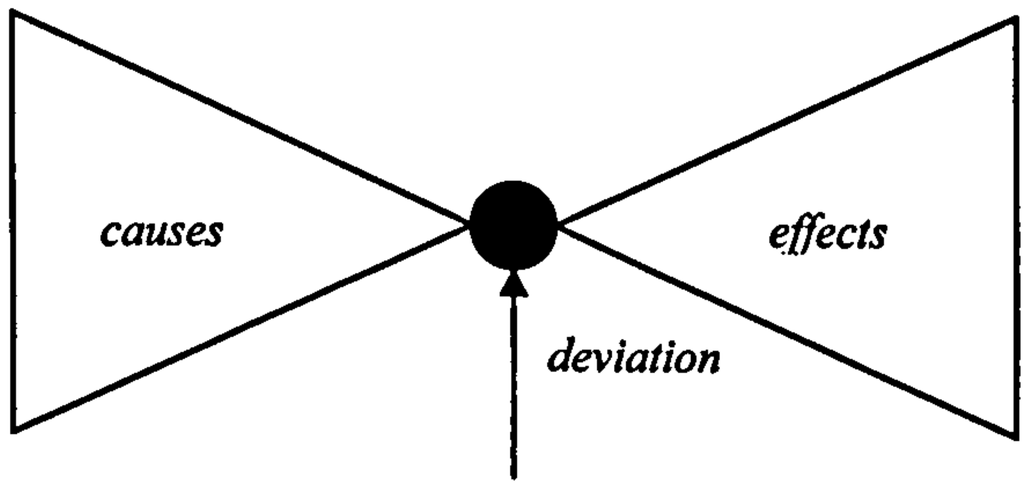

Fig.4.1 – Bow-Tie Analysis

4.3 Analyses during the System Lifecycle

Application of analyses is affected by the analysts' knowledge about the design in question. At first, during the initial stages of the system design the analysts do not possess a great deal of information about the design. Furthermore at early stages the design is still volatile as it undergoes a large number of changes in accordance to the results of the analyses. During the latest stages of design evolution the analysts have more data about the system as well as more detailed data. This is reflected on the type of analysis that is employed during the safety lifecycle stages. Airborne Recommended Practice 4761 (Appendix A) provides guidance on the safety lifecycle and its activities

in the context of safety analysis for complex aircraft systems. The following sections provide an overview of the main methods that influenced development of DDA. A common characteristic of the reviewed methods is the fact that they use deviations from intended operation, to probe the system design and examine whether the deviation can credibly affect the safety of the system. Deviation based techniques have been applied extensively to safety and are considered very successful during analysis of the system. Although these techniques have been developed with the intention to be used in safety analysis, there have been examples of deviation techniques applied in security [73]. The following subsections present an overview of the most notable safety analyses, the principles of which and their application in dependability are discussed in §4.7

4.3.1 Failure Modes and Effects Analysis

Failure Modes and Effects Analysis (FMEA) [74] is a method that considers deviations that are *known* to occur to the components examined, and assesses their effect. According to the design of the system, deviations affect the operation of the overall system.

4.3.2 Hazard and Operability Studies (HAZOPS)

The HAZard and Operability Studies (HAZOPS), is a methodology introduced in the chemical industry domain by the Imperial Chemical Industries (ICI) [75]. HAZOPS is a systematic analysis of the flows between the different parts of a chemical plant by considering deviations from the intended behaviour of the flows' attributes. HAZOPS uses a set of guidewords with which the design is probed. The guidewords identify a possible deviation. The aim of the technique is to identify the effects and the possible causes of the deviation. HAZOPS is nowadays a very popular technique used extensively in the safety domain. Technique was primarily for the process industry it has been extended for other uses, most notably by the Defence Standard DStan 00-58 for application on programmable electronics. Furthermore Pumfrey extended HAZOPS for safety analysis of software, specifying the Software Hazard Analysis and Resolution in Design (SHARD) [76]. Despite the fact that HAZOPS is a safety technique its popularity has prompted its application in security analyses [73] in which the deviations are substantiated in the context of security.

4.3.3 SHARD

A prominent characteristic of SHARD is that it uses a simplified set of guidewords compared with HAZOPS. The guidewords used in SHARD are optimised for software failures, eliminating potential ambiguity in interpreting the existing HAZOPS guidewords for software. The subject on which the SHARD guidewords are applied, consist of the attributes of the services provided by the components of a system. These include provision, timing and value of the provided service. When applying SHARD the guidewords (omission, commission, early, late, coarse value, subtle value) are applied on the services of the software. A service is defined as the *"communication of a piece of information, with a specific value, at a particular time"* [76]. Table.4.1 presents the main steps of SHARD.

Table.4.1 – SHARD Process

1	Understand the Design
2	Select Information Flow
3	Describe Flow and its Intended Behaviour
4	Ensure Intended Operation is Safe
5	Use Guideword to Suggest Deviation
6	Investigate Causes
7	Investigate Effects
8	Examine Detection Protection and Mitigation

4.3.4 What-if Analysis

A method akin to HAZOP is 'what-if' analysis [74]. However the involved stakeholders can be more flexible than analyses such as HAZOPS, as the deviations considered can be the result of brainstorming, or previous experience.

4.3.5 Sneak Analysis

Sneak analysis found widespread use in Boeing [77] as a means to systematically analyse electrical systems for unintended situations. Commonly there are five conditions for which the circuit is analysed:

- *Path*, examining for current flow along an unintended route.
- *Open*, examining lack of flow along an intended route.

- *Timing*, that prompts the circuit for flow at an incorrect time or lack of flow at the correct time.
- *Indication*, examining the false or ambiguous indication about a system.
- *Labels*, which prompts for a false or ambiguous label at the controls of the system.

Sneak analysis is limited in considering conditions regarding the electrical/electronic circuits of systems. This bears similarities to HAZOPS and SHARD in that the participants need to consider specific conditions and their respective causes and effects.

4.4 Dependability Deviation Analysis (DDA)

The Dependability Deviation Analysis (DDA), proposed by the author, is a methodical, exploratory, deviation based system analysis approach. Its purpose is to elicit multi-attribute (dependability) goal-based requirements, by being applied throughout the stages of the system lifecycle.

4.4.1 Overview

A failure that occurs during the operation of the system may impact on the achievement of a dependability attribute, which in its turn may have an impact on other attributes or result in unacceptable system operation. DDA extends deviation analysis techniques, investigating the system from the standpoint of each attribute. For example, a safety hazard can lead to accident due to a number of different 'types' of contributing factors such as the reliability of a component, performance of an algorithm or human error. Fig.4.2 shows a generalisation of the bow-tie analysis diagram used in safety, showing the interaction between dependability attributes. In safety the 'knot' of the bow-tie represents safety hazards. Wilson et al. [78] in attempt to model and relate all concepts in safety analysis described hazards as conditions. Generalising for dependability, the knot of the bow-tie represents potential failure to meet the required dependable operation, and in this thesis is defined as failure condition. The diagram also illustrates a significant characteristic of dependability; a failure to achieve a dependability attribute can be caused by and may result in failures regarding other attributes.

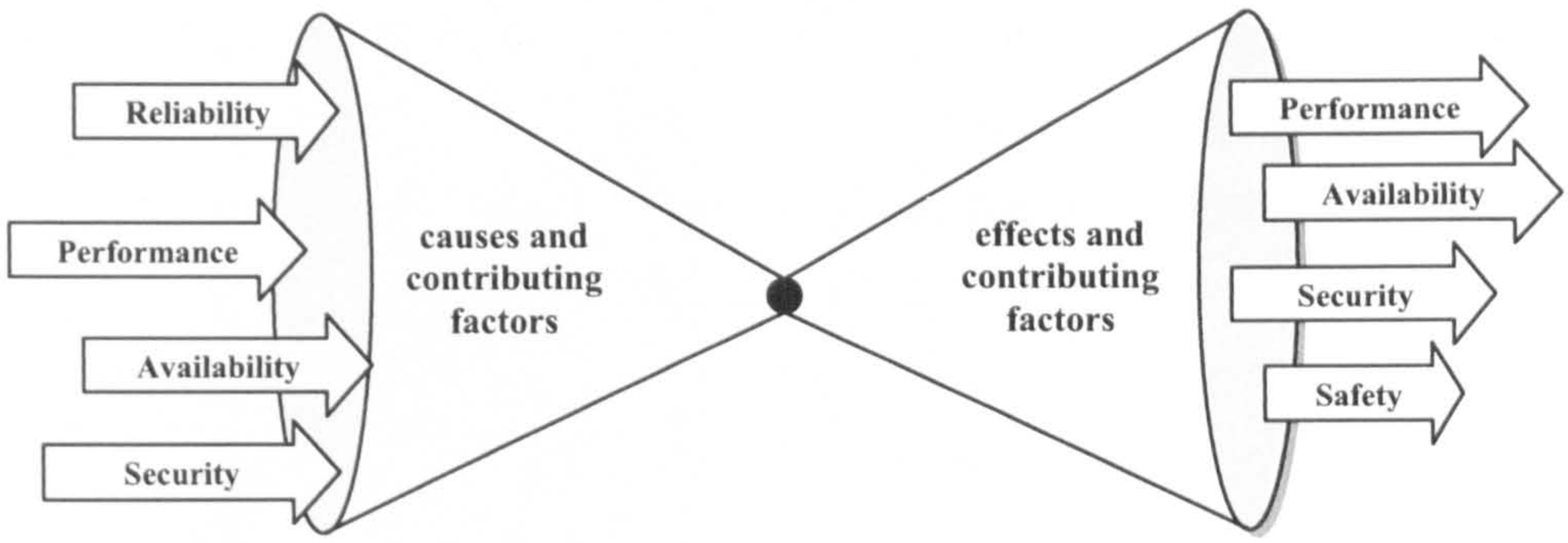

Fig.4.2 – A Multi-attribute Perspective of the Bow-tie Concept.

Reasoning about the overall behaviour of a system, necessitates considering the failure conditions with respect to the attributes of interest to the stakeholders, and their impact on the overall operation of the system. This requires understanding of how each of the attributes of interest can have an effect on another, as well as their contribution to the system's operational behaviour that is required by the system stakeholder. The projected effects of a deviation should not only be restricted to one dimension just by focusing on only one attribute. Instead, wider consequences should be assessed understanding how a deviation from the viewpoint of an attribute can affect the operation of the system. Fig.4.3 presents a schematic of the DDA, using the bow-tie diagram.

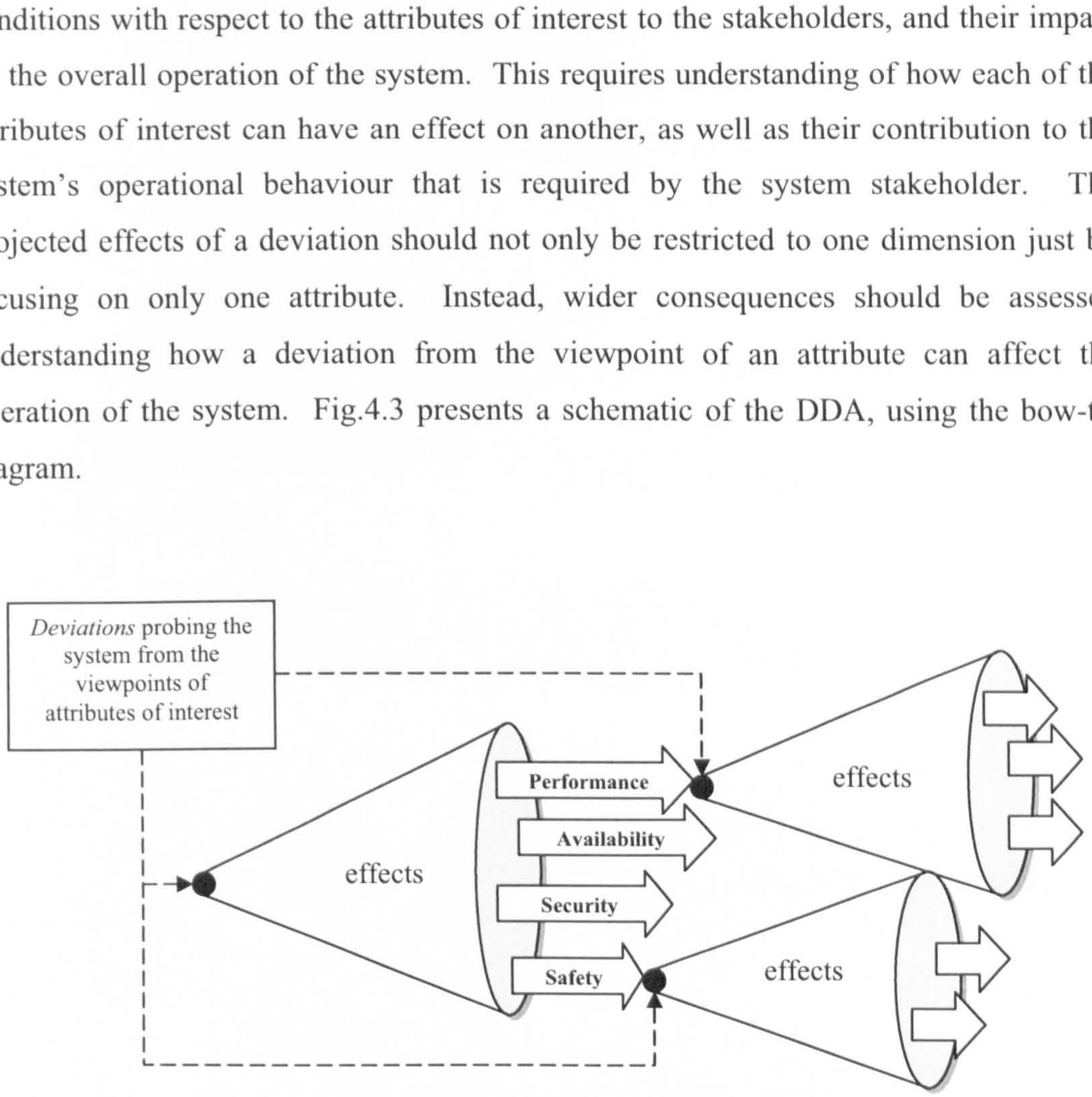

Fig.4.3 – Schematic Overview of DDA

DDA is an analysis method used to identify potential failure conditions with respect to dependability attributes and examine how these interrelate. It is an exploratory method focusing on the effect analysis part of the bow-tie diagram. The main 'questions' that the method aims to answer are: "What dependability failure conditions will affect the

required operation of the system?", and "What is the required operation of the system given the identified failures?". In the fashion of already established safety techniques, a representative set of deviations (discussed later during the presentation of the DDA process) is used to prompt the elements of the system, in order to reveal possible failures from the viewpoints of a dependability attribute (of interest to the stakeholders). The effect of the deviation may constitute the cause for a failure regarding a different dependability attribute. DDA provides a methodical way of identifying the relation between failures, which have been revealed after examining the design from the viewpoint of each attribute of interest. Ultimately, the DDA will result in a *failures' map* showing how the failures of the system are interrelated and how they can affect the overall behaviour of the system. DDA (using the failures' map) results in identifying how the deviations from intended behaviour of the system elements affect the overall behaviour of the system. This helps defining a profile for the system elements, specifying the required behaviour of that particular system element. The requirements are specified with respect to the identified dependability attributes from the viewpoint of which, the contribution of the system element to the overall operation was identified. The failures' map and the profiles of the system elements are the two DDA products. DDA has been defined in two levels. The underlying framework that presents the concepts and their **structure,** and the **process** that explains the necessary steps in order to perform he analysis.

4.4.2 Structure and Elements

The structure and relations of the concepts that make up the dependability deviation analysis have been defined in the dependability case metamodel. Fig.4.4 presents the metamodel of the DDA in UML. The following concepts are part of the DDA metamodel:

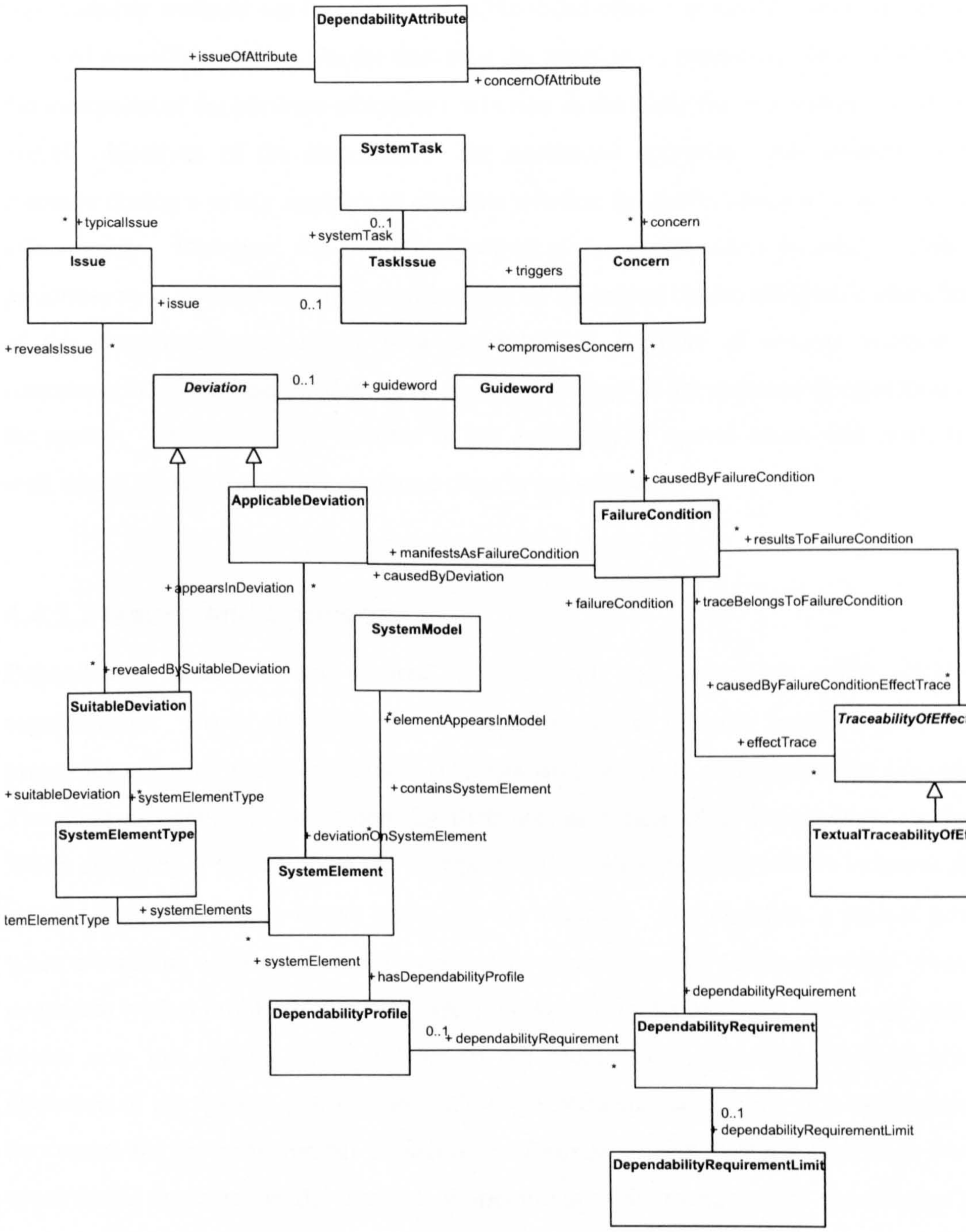

Fig.4.4 – DDA structure in the dependability case metamodel

4.4.2.1 Dependability Attributes

The class *dependability attribute* represents the dependability attributes of **interest to the stakeholders.** There are two main motives identified in this thesis, as to why a dependability attribute can be of interest to the stakeholders; in terms of analysis and in terms of overall objectives. In the first case the focus is on examining the system from the viewpoint of the attribute of interest, whereas in the latter the attribute is one of the overall objectives of the stakeholders for successful operation For example it is common during a safety analysis to examine whether the performance of a system can affect safety. However, the overall objective of the stakeholders is safety. Hence performance is an attribute of interest because of its impact on the acceptable operation of the system, whereas safety is a dependability attribute of interest because it constitutes what the stakeholders have specified as part of the dependable operation of the system. This distinction resulted in the definition of typical *issues* and *concerns*, with which the (dependability attribute) class is associated.

4.4.2.2 Issues and Concerns

Dependability attributes are abstract and difficult to substantiate when eliciting requirements. Issues and concerns are used to define in more tangible terms the properties affecting the stakeholders' interests with regard to a dependability attribute. The main distinction is the motive for their use, as explained in the previous section. When analysing a system from the viewpoint of an attribute, stakeholders examine the system regarding certain *issues* typical to the attribute. For example, a *typical issue* when examining a system from the perspective of performance can be *overload.* Issues constitute typical problems that may affect the system under analysis. However typical issues may not always be of interest to the stakeholders regarding the high level operation of the system. For example, although performance of parts of a system may be crucial for the achievement of safety, performance itself may not be a high level requirement described in the concept of operations of the system. The significance of *concerns* lies to the fact that a concern relates to the overall envisioned operation of the specific system under analysis. For example, consider the performance regarding the number of incoming requests of an online system. In this case overloading of the system will directly affect the envisioned operation of the system. Hence the performance of the system is an attribute in the stakeholders' envisioned system

operation. A *concern* is any potential occurrence with an unwanted impact on system operation. A dependability attribute *issue* may constitute a *concern* if it can be said that its effects impact the envisioned concept of system operation.

4.4.2.3 System Elements, System Element Types and System Models

The class *system element* represents any system entity that exists in the models of the system. A *system element type* is a class that is used to specify the type of the system elements such as activities, classes, actors, flows. Hence a system element refers to a unique entity in the design; for example 'information exchange 1' and 'information exchange 2' are two unique system elements of the same system element type. The system element type class was introduced in order to provide the capability of creating DDA templates that would not have to refer to system elements of the particular system. System elements types belong to *system models*. For example a use case diagram would be a system model having as system elements actors and functions.

4.4.2.4 Deviation and its Children Classes

A *deviation* is an abstract class representing possible behaviour of an element of the system in a way that was not intended. Similarly to HAZOPS a deviation is associated with a *guideword*, which indicates the nature of the deviation. Deviations are used in DDA to prompt the system revealing the possible effects of the specified issues. Deviation is specified in the metamodel as an abstract class, which means that it can be instantiated in a model. Instead the children class s*uitable deviation* and *applicable deviation* can be instantiated. The difference between the two classes is the subject of the deviation, which is prompted by the guideword inherited by the parent abstract class. A suitable deviation is associated with system element types whereas an applicable deviation with system elements. A suitable deviation is one that can efficiently reveal the effect of an *issue* by prompting the appropriate system elements types with the appropriate guidewords. When a suitable deviation is applicable to the system examined then an applicable deviation is created for each system element of system element type as the one related to the suitable deviation.

4.4.2.5 Guidewords

These are the *guidewords* used in deviations. They are used to prompt the appropriate system elements in order to establish the deviations that will reveal the (possible) effects of the identified issues. In methods such as HAZOPS and SHARD there is a standard set of guidewords, perceived to be capable of revealing failures common to the domain in which they are applied. However it is often the case that the set of guidewords prescribed by a method may not be optimised for all system models and potential failure conditions. Although guidewords used in existing methodologies can be used different guidewords may be selected more intuitive to probe the system for the issues that they reveal (further discussed during the description of the 'definition of suitable deviations' DDA stage).

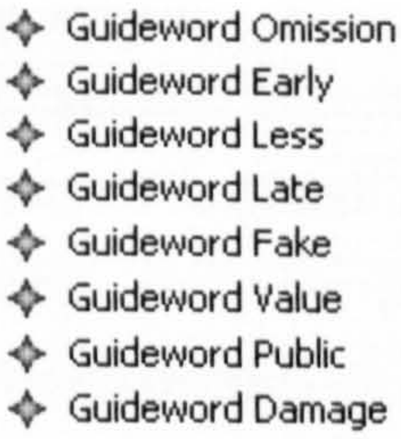

Fig.4.5 – Example Guidewords used in DDA

Fig.4.5 shows an exemplar set of guidewords, consisting of a mix of guidewords taken from existing methods such as SHARD, and guidewords defined during an example DDA process.

4.4.2.6 Failure Conditions

A *failure condition* describes a possible state of the system, which results to consequences unwanted to the stakeholders. A failure condition can be perceived as a credible manifestation of a deviation regarding a dependability attribute. Failure conditions can be associated with concerns and other failure conditions. Associations between failure conditions and concerns are declarative, implying that the stakeholders have identified cases, which can be justified to directly compromise their concerns. Associations between failure conditions imply that the stakeholders did not identify a direct compromise of a concern. The effects of a failure condition can result in other failure conditions, contributing indirectly to the overall behaviour of the system. The manner in which a failure condition will propagate in the system depends on the design

of the system. Associations between failure conditions are derived not declared, since they refer to eventualities of an existing system.

4.4.2.7 Traceability of Effect

Traceability of effect captures the effect of a failure condition. The concept captures the rationale, explaining the operation of the system in the presence of a failure and how this will propagate. Traceability of effects works as an association class between two failure conditions. Deriving the traceability of effect can be done manually through analysis of the design, or by use of more mechanistic approaches (an example of a mechanistic analysis is given by Mauri in [79]), according to how the system elements cooperate resulting in the overall system. This requires the existence of a more sophisticated traceability approach, capturing the relations between the system elements and how they are combined to produce the overall system. In order to be compatible with different means of traceability, the class (*traceability of effect*) is abstract, constituting the interface to other metamodels (e.g. the MODAF metamodel for a system described in MODAF). Initially DDA used a simple textual representation of traceability, represented by the *Textual Traceability of Effect* which extends the class *traceability of effect.* However later applications of DDA included a semi-automated approach using a basic traceability model.

4.4.2.8 Dependability Profiles and Dependability Requirements

The ultimate purpose of applying DDA is to elicit requirements for system elements from the viewpoints of the dependability attributes of interest. During deviation analysis, the participants identify the behaviour that would compromise their concerns, recording in the same time how the system element should perform to avoid unintended behaviour. The *dependability profile* represents a collation of requirements that define the behaviour of the system elements. The dependability profile was introduced to facilitate the evolution of the dependability argument (chapter 6), by grouping the acceptance criteria of a system element. A dependability profile is associated with the system element to which it belongs and to the *dependability requirements* it contains. A dependability requirement can be thought of as a specific declaration of the required target for the system element's behaviour with respect to an attribute. However as explained in chapter 1, conflicts between requirements will result in compromises of the

original targets. *Dependability requirement limits* is a fundamental concept used in the trade-off method (presented in chapter 5), representing the limit to which a requirement can be compromised whilst maintaining the overall system dependability to acceptable levels. DDA helps eliciting the rationale for the dependability requirement limit.

4.4.2.9 System Tasks

System tasks are the highest level tasks of the system that can be identified by the stakeholders. The class represents the direct interests of the system in terms of functionality. The collection of system tasks represents the (in MODAF terminology) SoS operation scenario. System tasks can be thought of as the highest level system activities that can be directly identified by the stakeholders. An alternative representation in terms of the metamodel would be systems tasks to be a separate system element type. Although earlier versions of the dependability case metamodel followed this approach, a distinction was made for two reasons. Firstly, system tasks are used in a different type of analysis (which is part of DDA) with the purpose to identify the concerns of the stakeholders. This type of analysis, similar in principle to Functional Failure Analysis (FFA), has different purpose and characteristics to the rest of DDA (which resembles the analyses during PSSA as suggested by ARP 4761), as it takes place very early in the system lifecycle. Separating the concepts provides better context for the DDA participants making the overall approach more intuitive. Secondly, tasks constitute the overall concept of operation of the system, and inevitably are abstract and stated in an ad-hoc way. Furthermore in MODAF, which was the modelling framework mostly used in the thesis, as well as in the tried case studies, tasks were (often) described in plain text making it difficult to represent high level tasks as system elements.

4.4.2.10 Task Issues

Task issues are used to identify the concerns of the stakeholders. Its function is to juxtapose the top level tasks with typical dependability issues, identifying whether they constitute concerns to the stakeholders' interests. A task issue is associated with a system task and an issue. Despite its similarities to a deviation, a distinction was made for the same two reasons as system tasks.

4.5 Overview of the DDA Process

This section presents the main stages that constitute the overall DDA process. During the process the participants instantiate and use the concepts that are defined by the metamodel.

4.5.1 Using the Metamodel to Create Templates

Some of the concepts in DDA do not need to be recreated with every application of the DDA but can be reused in analysis of many systems (e.g. typical issues). Instantiation of the concept using the metamodel allows 'storage' as objects in EMF (Eclipse Modelling Framework). This is a useful capability allowing reuse of certain parts of the DDA.

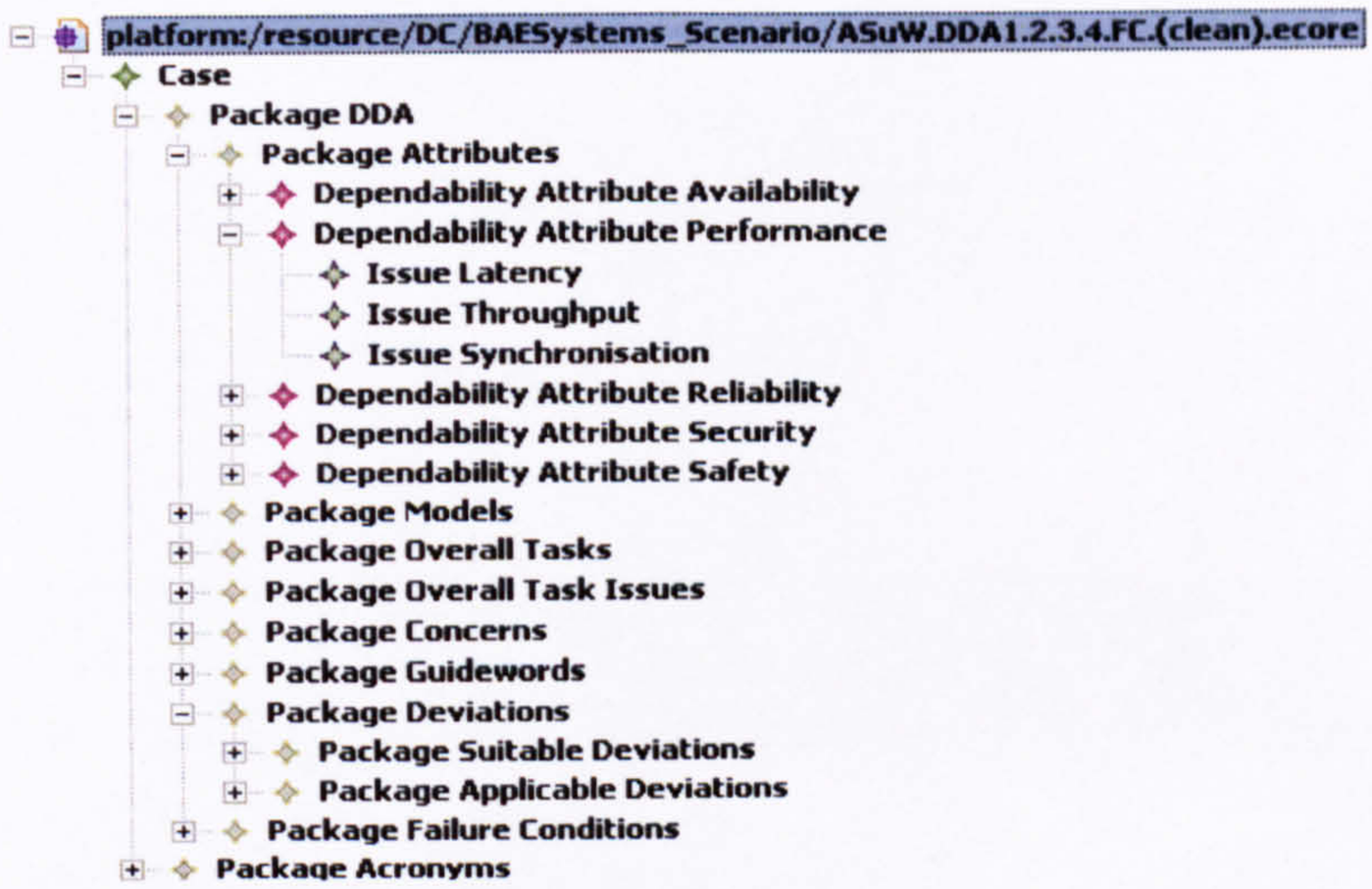

Fig.4.6 – A DDA Template in the EMF Editor

Fig.4.6 shows a DDA template used by the author in a number of analyses. It contains the initial packages that need to be populated during DDA, the attributes of interest, their issues as well as suitable deviations (with the respective system element types and guidewords) able to reveal the identified issues. The template was a starting point of the analysis as it contained all this information that did not have to be recreated. However concepts such as the applicable deviations, the system elements, and the failure conditions depend on the specifics of the system under analysis and therefore cannot be used as templates.

4.5.2 Overall DDA process

The DDA process consists of seven stages. Each stage is composite, consisting of a number of steps. Fig.4.7 presents the overall DDA process and its stages; each DDA stage (apart from identification of issues) is described in more detail with a dedicated flow diagram. The sequence of the stages depends on reuse of existing templates. For the purpose of describing the process, we assume no use of templates. This implies application of all possible stages. The process starts with identification of typical issues for which the system is analysed. Following identification of issues, the stakeholders need to examine their overall dependability objectives by defining what is of primary concern to their interests.

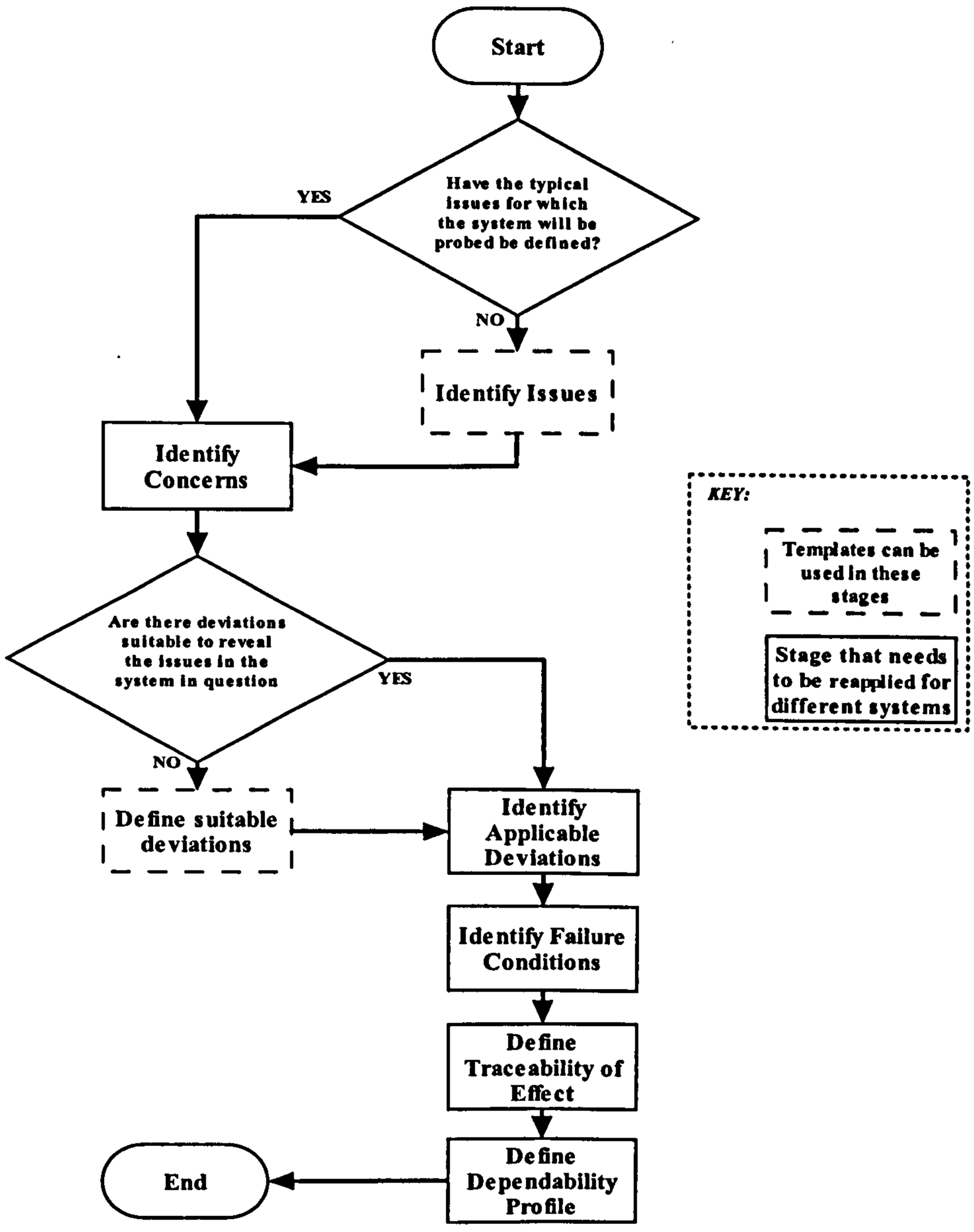

Fig.4.7 – Overall Stages of the DDA

After completion of identification of concerns the participants specify the deviations suitable to reveal the typical issues in the model types used in for the system design. Once the participants state which of the suitable deviations are applicable for the system in question, the resultant failure conditions are described examining their effect to the system. Finally, after identification of the possible failure conditions, the participants identify the required behaviour of the system. The required behaviour is specified in terms of the dependability attributes of interest, in order to satisfy the stakeholders' overall objectives.

4.6 Underlying Principles of DDA

DDA exercises the following two principles, which have influenced the definition of its structure and concepts:

- Commonality of concepts between dependability attributes.

  Dependability attributes have commonality between their concepts which are generically represented in the metamodel.

- Extensibility of (deviation) guidewords and representation of typical issues.

  In traditional deviation analyses (HAZOP) as well as in application of deviation analyses in specific domains (SHARD, security deviational analysis), the guidewords prompt for issues from the viewpoint of a particular attribute.

4.6.1 Similarity of Concepts between Dependability Attributes

The Dependability Case Metamodel (DCM) encompasses the dependability attributes, by generalising on some of their common concepts. Security and safety are two dependability attributes which have interested developers extensively, and as a result have well-established concepts. Table.4.2 juxtaposes similar concepts in safety and security and presents the concepts with which parallelisms can be drawn in the dependability case metamodel.

Table.4.2 – Comparison of Concepts in Safety, Security and the DCM

Safety	Security	DCM
Asset	Asset	System Element
Hazard	Threat	Failure Condition
Fault	Vulnerability	Issue
Accident	Attack	Concern

Similarity of concepts in safety and security is a position asserted in the SafSec project [35], which was an attempt to establish a framework for jointly reasoning about the safety and the security of military systems. SafSec adopts a risk based approach according to which both security and safety threats are evaluated in terms of their effect on the assets. This is achieved by defining the concept of loss; a generic term for any unwanted system state that can occur from either a (safety) accident or (security) attack, which is the ultimate *concern* of the stakeholders.

Assets in both safety and security represent entities of the system whether this is humans, other systems or monetary value. In order to meaningfully analyse how deviations may affect them (entities) they should be part of the system model. Hence during the deviation analysis, whether the subject of the analysis are entities such as humans data or other systems, they are all part of a container system model and are represented by the *system element* concept.

Hazards and threats are two similar concepts that in some instances have been used interchangeably. SafSec [35] uses the two terms to refer to occurrences that can have unwanted impact on either safety or security. Laprie [9] used the term threat to describe anything that can compromise the behaviour of a system in terms of dependability attribute. Hazard is a concept tightly associated with safety. Pumfrey, among other hazard characteristics, mentions that hazards are *"conditions which can be mitigated, but from which an accident can arise through a sequence of normal events or actions"* [76]. Similarly a *failure condition* can lead to compromising dependability attributes by contributing to the realisation of the stakeholders' concerns.

Issues represent potential occurrences, under the viewpoint of any of the dependability attributes, which can result in compromise of dependable operation of the system. Issues can result in failure conditions. Srivatanakul examines in her thesis the application of deviation analyses to security of a system. Similarly to safety, the

analysis is done by interpreting the HAZOP guidewords from the perspective of security. Of interest is the addition of a column for identifying vulnerabilities that when exploited could threaten the system. A threat to a computer system is defined as any potential occurrence, malicious or otherwise, that can have an undesirable effect on the assets and resources associated with a system [80]. Similarly to hazards being conditions that can result in an accident, threats constitute the potential for an attack. Firesmith [81] also identifies the similarities and further describes threats and hazards as types of danger for the system. A vulnerability is a weakness in a system that may be exploited by a threat, resulting in an attack [82]. Correspondingly a fault in the system can be the cause of a hazard that may result in an accident. Firesmith uses the term vulnerabilities to describe internal system conditions that may impact both safety and security [81]. Finally, similarly to SafSec, both Firesmith [81] and Srivatanakul [73] contend that an asset is any component of a system which is of value to its stakeholders.

4.6.2 Extensibility of (deviation) Guidewords Representing Typical Issues

HAZOPS uses a set of guidewords to probe the design for possible deviations. A deviation occurs when the system does not operate as the designers intended it to. However in safety analysis all deviations do not necessarily constitute hazards. The examined deviation may affect stakeholders with interests towards other attributes, however from the viewpoint of safety they are not examined.

An issue with HAZOPS is completeness of the guidewords. There is a question whether the guidewords can identify all possible deviations than may occur. Kletz [75] stresses that batch plants (as opposed to continuous flow plants) require a different set of guidewords in order to comprehensively probe the design. Being a methodology that emerged from the chemical industry, the set of guidewords in HAZOPS is optimised to identify failures relating to the flows and containers in a chemical plant. However the types of failures that the original set of HAZOPS guidewords covers is not representative of all systems, which can demonstrate other classes of failures. Related to more specific applications, Pumfrey [76] indicates the existence of a set of guidewords adapted to probe for failures relating to human factors. Defence standard 00-58 proposes an extended set of guidewords that covers failures common in programmable electronics.

Furthermore, SHARD [76] uses an alternate set of guidewords suitable for identifying failure conditions related to software. In all approaches, definition of guidewords relies on an underlying failure model. This is defined taking into account a number of sources including previous experience domain knowledge and existing failure models. Fig.4.8 presents the deviations in three analysis methods; HAZOPS, SHARD and sneak circuit analysis.

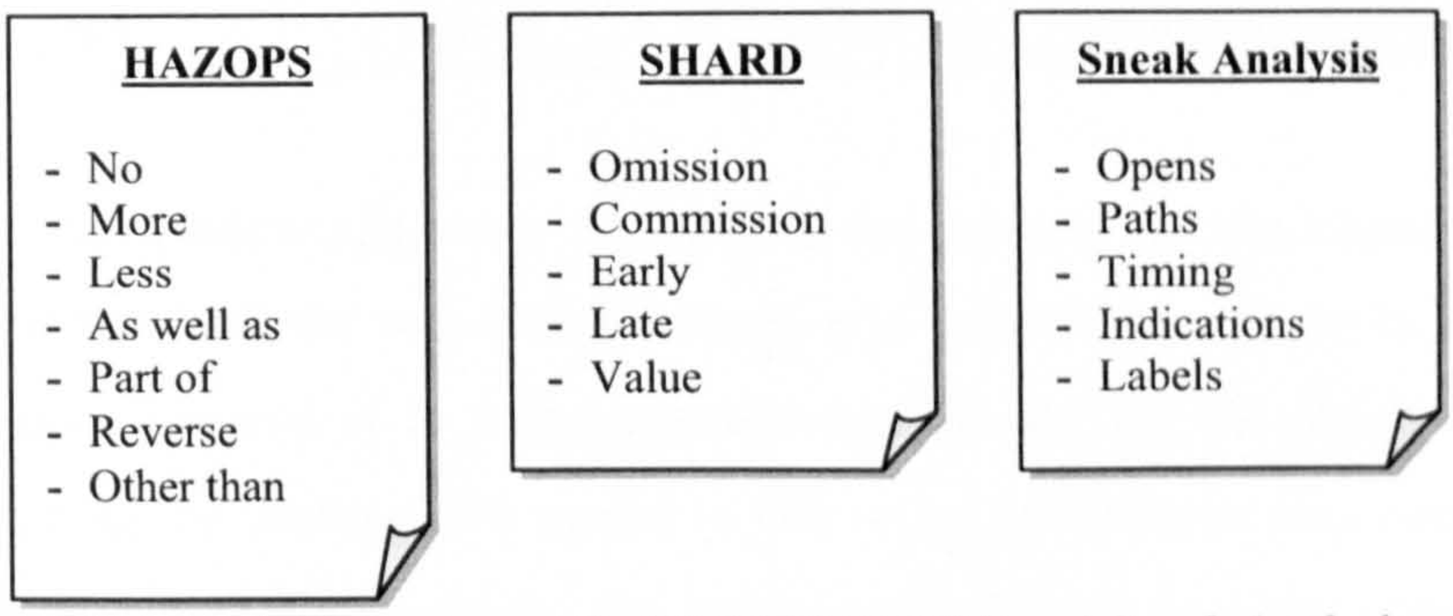

Fig.4.8 – Guidewords used in HAZOPS SHARD and Sneak Analysis

Comparing the guidewords with dependability we can observe that the guidewords represent a set of issues from the viewpoint of different dependability attributes. For example guideword NO could be seen as representing an availability issue. In other words, the analysts are prompted to investigate the condition in which the system element is not available. Despite the fact that the three methods were developed for different domains (HAZOP for process industry, SHARD for software and sneak analysis for electrical circuits) there are similarities in the underlying deviation guidewords. Omission of a service in SHARD (e.g. no data flow between two processes) could be seen as the equivalent of 'NO' in HAZOPS and an open circuit (i.e. no current flow along an intended route). In all three cases the examined deviation was related with the availability of functionality, whether this was data or messages or current. Table.4.3 presents how the deviations in HAZOP, SHARD and sneak analysis can be interpreted as being associated with the domain of particular dependability attributes. This table shows how, for example, service provision guidewords (such as 'omission' and 'no') relate to the domain of service *availability*.

Table.4.3 – Mapping Between Guidewords and Attributes

Deviation	Attribute	Deviation	Attribute	Deviation	Attribute
No	Availability	Omission	Availability	Path	Reliability
More	Reliability	Commission	Reliability	Timing	Performance
Less	Reliability	Early	Performance	Indication	Reliability
Part of	Performance	Late	Performance	Label	Human factors
Reverse	Reliability	Value	Reliability		
Other than	n/a	Open	Availability		

In HAZOP, the guidewords were defined in the context of the chemical industry domain. The resultant set was broad enough and generic enough to be applied in a number of domains and it is being applied confidently by the safety community. However limiting the probing of a model to this set of guidewords may cause problems during the analysis. Being generic, the guidewords require scoping, imagination and extensive experience on interpreting the deviations. Srivatanakul in [73] uses HAZOP in order to perform a security analysis of a design. Again, the need for appropriate interpretation is highlighted. SHARD is an example in which the guidewords were altered to be compatible with software failure models. Furthermore the Department of Trade and Industry (DTI) safety case for the parson's current limiter [83] indicates the use of concerns such as *failure of power supply* directly as guidewords probing elements of the design models. Definition of the guidewords is important when analysing a system. Care needs to be taken, as the guidewords have not necessarily been specified taking into account the classes of failures that an analyst may want to probe the model for. It is important for the participants of the analysis to have a clear understanding of the scope of deviations that the defined guidewords cover. Even though a set of guidewords (representing a set of possible deviations) can be reused, once defined caution must be taken when the analysts intend to investigate the system for different issues which may not be revealed effectively.

4.7 DDA Stages

This section describes the individual stages of DDA as presented in Fig.4.9. The description entails the steps followed in each stage as well as the decisions need to be taken. Application of the DDA stages is illustrated using the AGO scenario example.

4.7.1 Identification of Typical Issues

Each of the dependability attributes represents a unique viewpoint regarding the operation of the system. There is no universal definition of typical issues. Instead the stakeholders of a system decide what is important to them regarding the operation of the system. For example in the safety case report for a current limiter device, the analysis identified typical issues prevalent to electrical devices, such as *failure of power supply*, and *electricity contamination*. These are not issues that would appear in analysis of other systems. However the system was designed primarily for being used in electrical networks, a domain in which these are typical issues that can result in failures affecting the (in this case) safe operation of the system. Stakeholders may be influenced with the definition of typical issues according to the domain in which the system is deployed. This is apparent in the descriptions of some defence standards which identify typical issues that have been inherited from accumulated past experience.

Standards often provide guidelines about what needs to be addressed in a system, echoing experiences and typical issues gathered over a period of years by a relatively large sample of practitioners. There are several examples of standards and good practice guides which can constitute sources for typical issues regarding the dependability attributes. The Ministry of Defence (MoD) (safety) Standard 00-56 (which supersedes 00-55 [84]), in its definition of safety identifies issues such as physical injury and material damage. Defence Standard 00-25 [85] which addresses human factors, mentions among others fatigue and workload. Pumfrey identified failure classes applicable to operating systems, such as incorrect value and deadline miss, which resulted in definition of the respective guidewords in SHARD (As discussed in §4.1 and §4.2, these failure classes can be modelled in the DCM as issues related to dependability attributes). The Common Criteria for security [61] suggest issues such as unauthorised value and identity assurance. Finally, Quality Attribute scenarios [86] allow identification of issues regarding a number of attributes. Table.4.4 presents example issues for the dependability attributes, identified in the aforementioned standards and practices.

Table.4.4 – Compilation of Typical Issues

Attribute	Issues	Attribute	Issues	Attribute	Issues
Availability	A service is not available as specified when required	**Reliability**	Physical Failure	**Safety**	Damage to health
			Incorrect value		Physical damage
			Data integrity		Pollution
			Incorrect state		
Performance	Latency	**Security**	Unauthorised Access	**Human Factors**	Comfort
	Deadline miss		Repudiation of transaction		Mistakes
	Throughput		Identity assurance		Fatigue

Although indicative, according to the type of the systems the stakeholders can specify a different set of issues. DeGarmo in [87] identifies issues regarding the operation of Unmanned Aerial Vehicles (UAV). Furthermore a different set of issues was used during a case study involving High Altitude Platforms (HAPs), which are unmanned aerial vehicles with the purpose to provide broadband communications over a large area. These included issues common to communication aspects of the system such as denial of service, interference and power emissions.

4.7.2 Identification of Concerns

During this stage of the DDA the stakeholders identify the possible impact of the concerns on the system's concept of operation. Subjectivity of the stakeholders' requirements needs documentation of the rationale based on which the concerns are defined. At this stage it is essential that the acknowledged attributes of interest and their concerns are related to the concept of operations. Hence, reasoning regarding possible compromise of an attribute can be evaluated in the context of the system's operation.

The stakeholders of the system recognise a concern because it principally affects their interests regarding the envisioned operation of a system. In contrast, failure conditions are recognised as potential contributing factors to systems concerns. Fig.4.9 shows the sequence of steps that need to be taken and decisions that need to be made in order to establish the concerns during DDA.

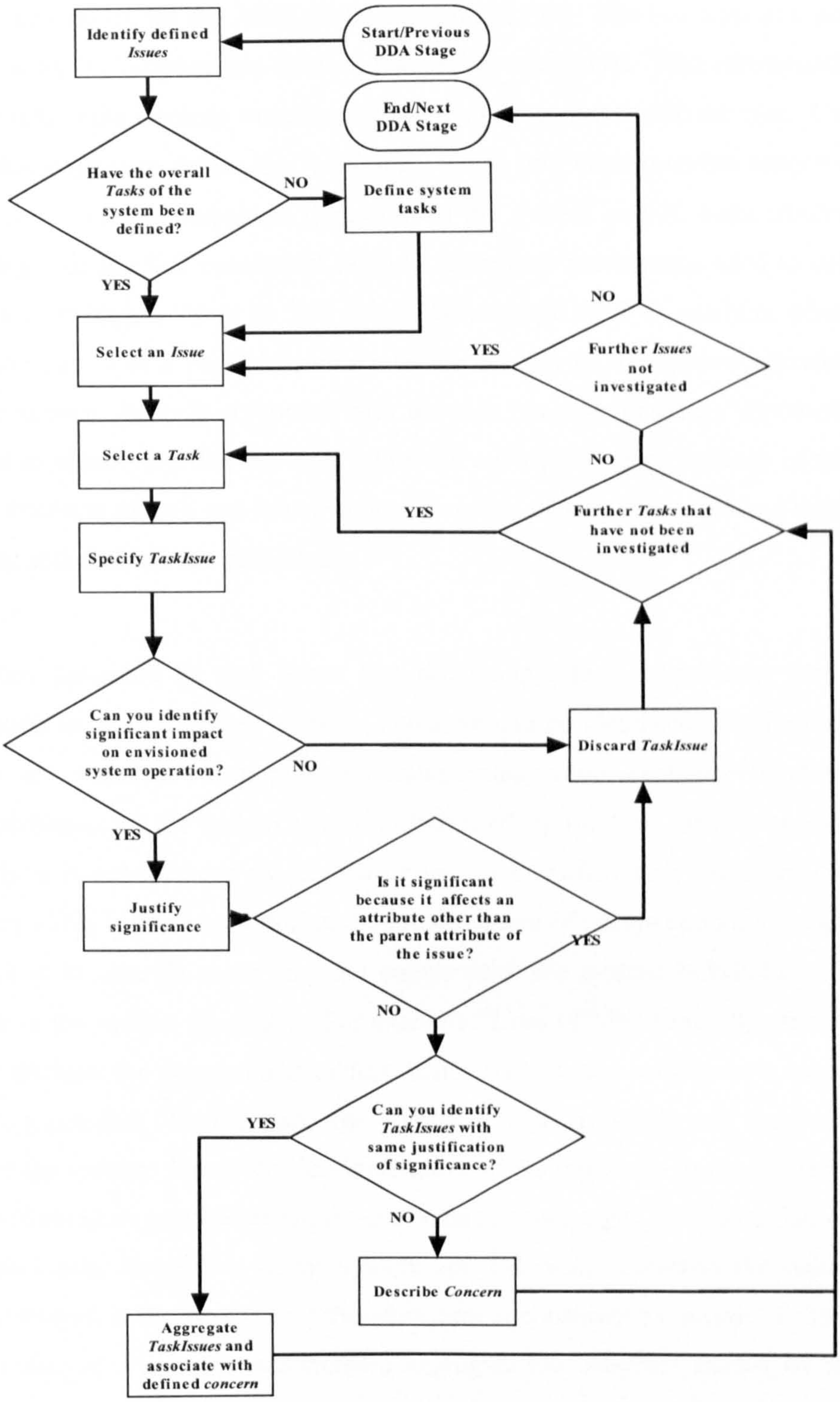

Fig.4.9 –Identification of Concerns Stage

Table.4.5 summarises the identified tasks, the typical issues and the resultant impact from the task issues for the AGO scenario (Appendix B). The box next to a task issue with the letter "C" implies that the task issue raises a concern. This corresponds in the DCM to setting the Boolean variable *isAConcern* of the class *TaskIssue* true. The initial aim of this stage is to define the *TaskIssues*, which will elicit possible concerns of the stakeholders. This prerequisites definition of the system overall tasks ideally at the initial stages of the SoS conceptual design. Otherwise participants need to define the overall system tasks. Next in this stage, participants need to examine whether the *TaskIssues* can raise a possible concern in the context of the system operation. For example latency (issue 2) combined with the task "transfer of forces" (system task 4) will lead to identifying that this may allow the enemy to escape because of the delay. The combination of task and issues is used a means of identifying the end results with which the stakeholders are interested.

Conditions described by task issues that are thought to be significant need further deliberation in order to assess whether a concern can be identified. Participants in the analysis are required to support the initial supposition explicitly identifying the projected impact on the system operation indicated by the task. However this step of the analysis is not intended to focus on how the identified task issue may affect the operation. (This is performed during the identification of failure conditions stage). The objective is to identify cases that can compromise the system stakeholders' primary interests in the system operation. For example "Loss of life" (issue 8, system task 4) directly interests the system stakeholders, hence issue 8 also constitutes a concern. By contrast, a possible "Enemy Sabotage" (issue 7, system task 5) can have significant effect on the system. However, the significance of the task issue lies in the fact that the effect will result in possible casualties due to enemy sabotage. The identified condition is a contributing factor to a safety concern but it does not represent the stakeholders' ultimate interest in the system, and therefore cannot constitute a concern. Following the identification of a concern stakeholders aggregate the concerns caused by the same issues. For example "unauthorised access" is a concern revealed from two task issues that the stakeholders have identified significant for different reasons, which need to be recorded. Justification of significance can be used to further elaborate on the concerns during construction of the dependability case. The steps are followed until all possible task issue pairs have been examined.

Table.4.5 – Definition of Task Issues and Identification of Concerns

Concerns Elicited from Impact of Resultant Task Issues	System Tasks				
	System Task 1 UAVs Patrol Area	System Task 2 Sharing of Information	System Task 3 Suppress Enemy Position	System Task 4 Transfer Forces	System Task 5 Engage Enemy
Typical Issues 1. Unavailability	-	-	-	-	Enemy forces will not be neutralised
2. Latency	-	-	-	Enemy forces escape C	Enemy forces escape C
3. Throughput	-	Systems cannot suppress enemy adequately.	-	-	Systems cannot suppress enemy adequately.
4. Physical Failure	System degradation	-	-	-	-
5. Data Integrity	-	Systems cannot use their data or use wrong data	Suppressing wrong position	-	-
6. Unauthorised Access	-	Strategic/tactical information disclosed to enemy. C	-	Tactical information disclosed to enemy. C	-
7. Assumed Identity	Enemy may sabotage and mislead friendly units.	-	-	-	Enemy may sabotage and mislead friendly units
8. Friendly Fire	System degradation	-	-	Loss of life C	Loss of life C
9. Accident	-	-	-	Loss of life C	-

4.7.3 Definition of Suitable Deviations

One of the challenges applying the deviation guidewords for dependability analysis is the use of the appropriate system models. For example the UK defence standard 00-58 (HAZOP studies) provides guidance on the type of models that can be used to model programmable electronics (referred in the standard as PES). The IEEE STD 1471-2000 [88] recommended practice for architectural description of software intensive systems, suggests that some models are more appropriate in expressing specific system viewpoints (e.g. engineering viewpoint, decomposition viewpoints). The recommended practice identifies a number of concepts that are necessary for models to effectively capture the proposed viewpoints. Instantiation of the deviations should be done using suitable models in order to achieve a meaningful analysis. For example, data flow diagrams are suited to revealing timing concerns whereas human factor concerns can be revealed more easily in use cases. Representation of concerns in system architecture views focuses on particular aspects of the system. Apart from suitable system element types, suitable deviations also entail the guidewords, which in combination to the system element type will reveal the dependability issues. The combination of system element type and guideword should result in suitable deviations that are meaningful and unambiguous in terms of their interpretations. For DDA, we identify exemplar suitable deviations for system element types, in accordance to the Ministry Of Defence Architectural Framework (MODAF) – in which the AGO scenario was captured.

4.7.3.1 Defence Architecture Frameworks (DAF)

The US Department of Defence initially specified DODAF in order to "define a common approach for DoD architecture description development, presentation, and integration for both war fighting operations and business operations and processes" [23]. The framework consists of a number of products organised in views, which ultimately describe the complete operation of the system.

The products describe various aspects of the system and essentially constitute the models of the system. DODAF products are organised in four views; "operational", "systems", "technical", and the "all-views" view which is used as reference to maintain consistency between the views and the products. MODAF is the equivalent framework defined by the UK Ministry of Defence (MoD). The MoD tailored MODAF to its

particular needs adding two more views; the "strategic" and "acquisition" views. MODAF and DODAF models are very similar and the views that are common share the same set of products.

Being primarily a framework for modelling enterprise architectures, MODAF does not have a particular language in which it is modelled. Although the guides describing the framework provide examples for implementing each of the products, it is stated (in DODAF definition documents) that the users can employ any modelling language as they see fit. Furthermore, the MODAF documentation [89] as well as further studies (such as [90]), provides a description of how each of the products could be modelled in UML. Table.2 in appendix B shows a summary of the UML models that can be used to represent MODAF products.

4.7.3.2 Process Walkthrough

Definition of suitable deviations is a systematic process examining the potential of the modelling language – in which the system design is documented – to reveal the identified issues. Fig.4.10 presents the steps of this DDA stage. The steps are applicable for any modelling framework. However, the walkthrough provides examples that were based on use of MODAF. Selection of models is a process that ought to be performed by the involved stakeholders according to a number of criteria. Firstly, the stakeholders need to evaluate the suitability of each system element type to reveal the specified dependability issues. This involves subjectivity from the stakeholders that make the evaluations. One such evaluation is whether the issue can be manifested at the system element type being examined. This requires understanding of what the system element type represents in the system models. In order to achieve this we used the MODAF metamodel (M3). The meta-model of a language documents the concepts, that are modelled and how these are associated with each other. The classes of the meta-model can assist with unambiguously documenting what each class represents in the model language that was used. The MODAF meta-model (M3) provides an insight into all products of all views explicitly identifying the concepts entities and data that each of the products describes. Although both the DODAF and the MODAF meta-models are incredibly similar, in order to avoid confusion due to possible minor difference we have focused on MODAF.

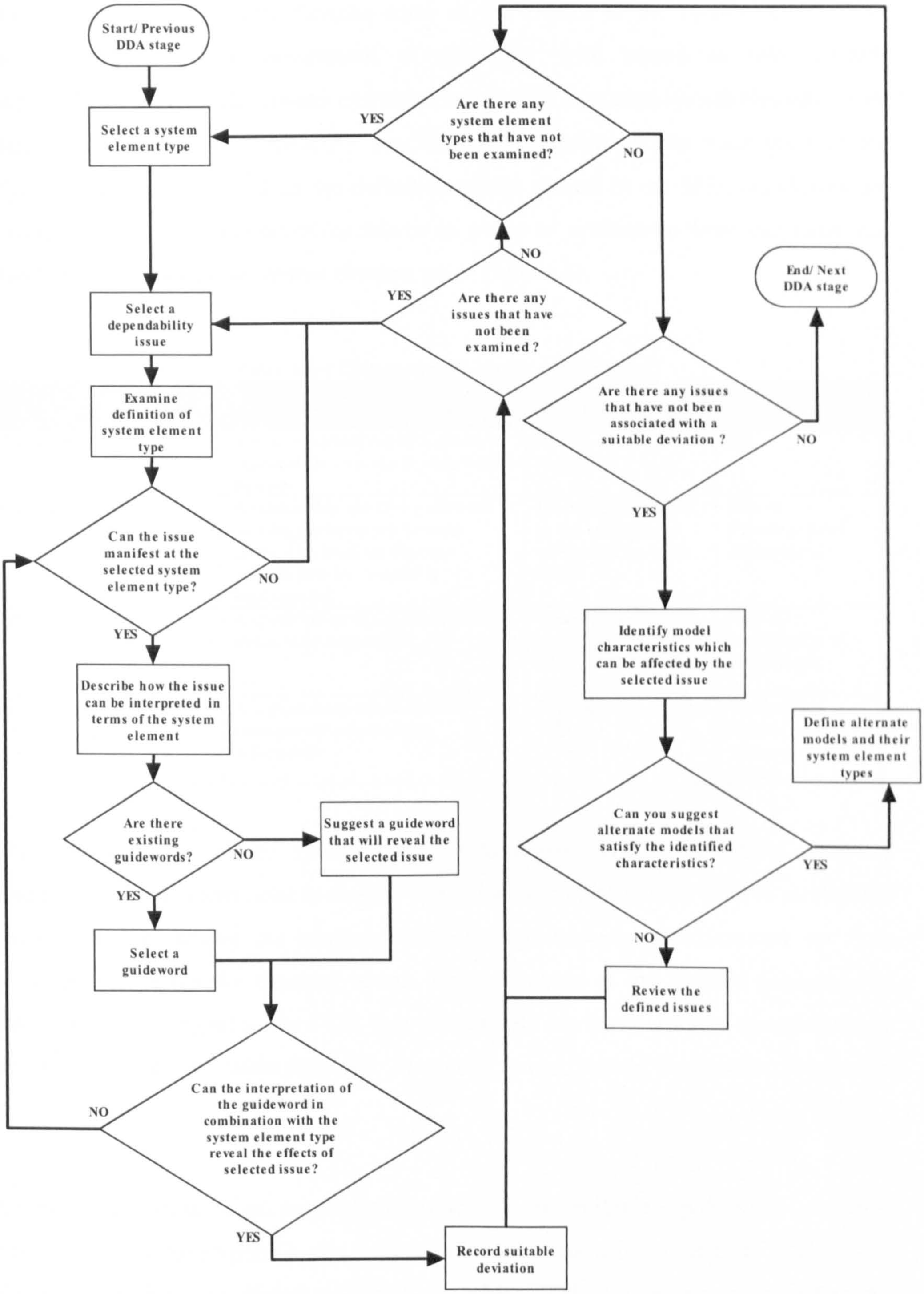

Fig.4.10 – Definition of Suitable Deviations Stage Process

Table.4.6 presents an example of analysing several classes from the node connectivity (OV-2) product. The table contains some of the classes of the product OV-2, their description as well as assessment of suitability with regard to two example dependability issues. The classes examined in the table represent system elements types that appear in the OV-2 MODAF models. After understanding what each of the elements represents (based on the definition of the classes in the M3), the classes are juxtaposed with the dependability issues in order to understand how the issue can manifest on the particular system element type (Table.4.6).

Table.4.6 – Metamodel Elements and Failures

Class	Description	Unauthorised Access	Throughput
OperationalActivity	A process carried out by a person or organisation – i.e. not an automated function.	Unauthorised access to activity	n/a
Needline	A relationship specifying the need to exchange information between nodes. The needline does not indicate how the transfer is implemented.	Unauthorised access to the information exchanged between nodes.	Rate of transmission of information.
InformationExchange	A specification of the information that is to be exchanged.	Unauthorised access to the information exchanged between nodes.	Rate of transmission of information.
Node	A logical entity which creates consumes or manipulates information.	Unauthorised access to node.	Node cannot process the information fast enough.

Once the potential of the system elements to capture the dependability issues is established, participants need to identify the guidewords that will be used to prompt the system element during the analysis. Existing guidewords are examined for their suitability to reveal the required issue. Expressiveness of the defined guideword is important. It is crucial to the DDA that guidewords can be interpreted unambiguously so that analysts can understand the viewpoint under which the system element is prompted.

Upon identification of an appropriate guideword the suitable deviation is recorded. Fig.4.11 shows the properties of a 'suitable deviation' instance in the EMF editor. The deviation has been associated with other instances of the metamodel representing the used guideword, the system element type and the issue for which it has been optimised to reveal.

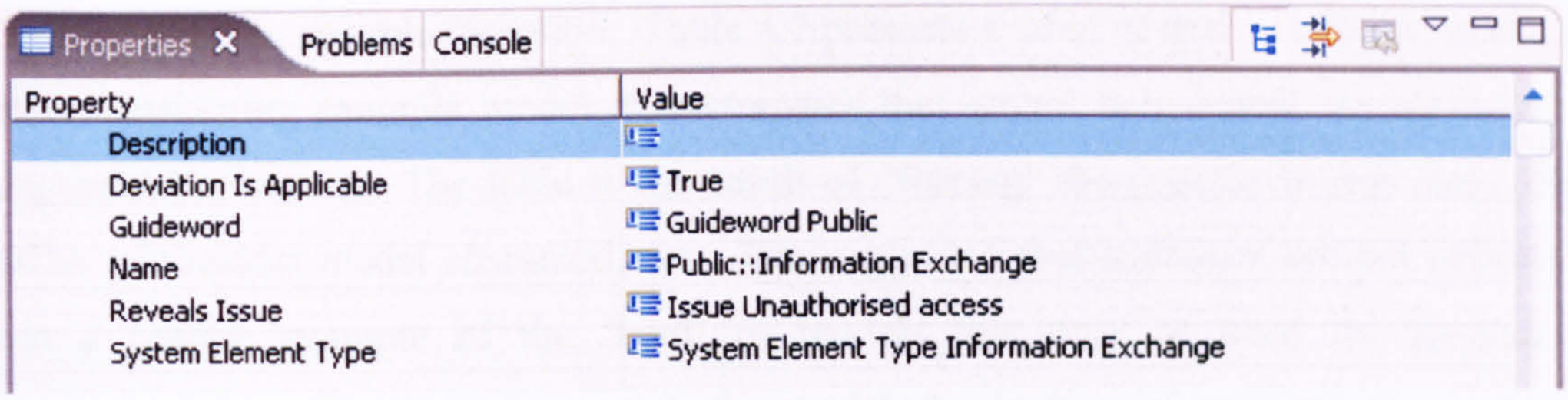

Fig.4.11 – Properties of a Suitable Deviation in the EMF Editor

Fig.4.12 shows the properties of the issue "unauthorised access" in the EMF editor. The issue which was identified from the viewpoint of the security attribute is revealed by a suitable deviation (i.e. Public::Information Exchange) which consists of the guideword "public" and the system element type "information exchange" (the properties of this deviation are shown in Fig.4.13).

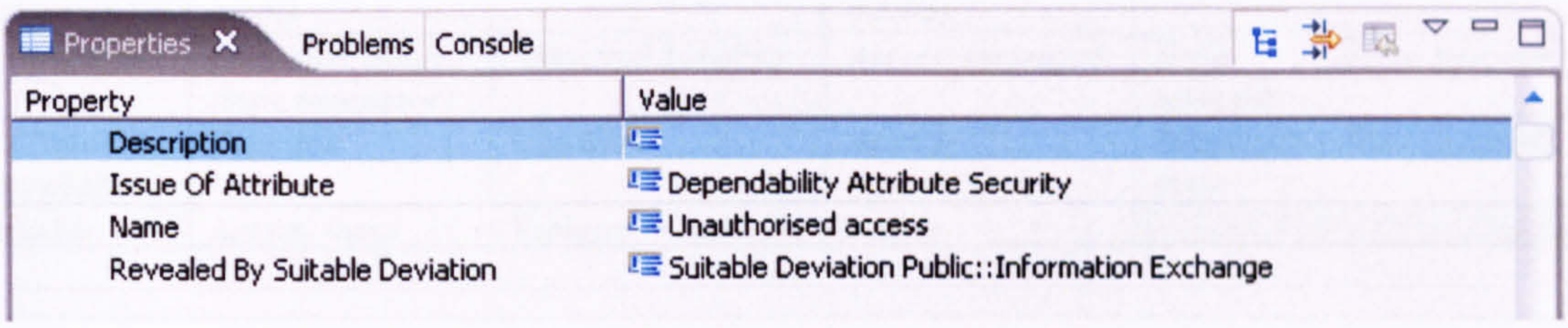

Fig.4.12 – Properties of an Issue in the EMF Editor

Following the examination of existing guidewords and specification of new ones, the process examines the completeness of suitable deviations regarding the identified dependability issues. In some cases there is a possibility that the defined suitable deviations do not cover all issues (i.e. some issues have not been associated to a suitable deviation despite the have been examined). This can be the result in case the available models of the system do not have the necessary characteristics to clearly reveal the issues (combinations of system element types and issues for which the result of the first decision node in Fig.4.10 is 'No'). To more easily understand this, we can consider trying to define a deviation suitable to reveal synchronisation issues, using a model in which temporal characteristics (e.g. sequence of events) are not present.

Following identification that the identified models cannot satisfactorily participate in a suitable deviation, the participants need to identify the models that would help to complete the analysis. Initially the participants are asked to identify the 'inadequacies' of the identified models, by brainstorming the characteristics that the models lack in

order to define a suitable deviation. Table.4.7 presents a table of derived characteristics after considering example model characteristics that would help reveal the identified dependability issues. The table is the result of 'forcing' the process in this stage of DDA to consider model characteristics. The suggested characteristics are not definite but a generic example of the 'kind' of models that may be used for analysis. Subsequent to suggestion of a model, the model should be analysed and the system element types included identified, following assessment of its suitability.

Table.4.7 – Example Model Characteristics Required to Reveal Dependability Issues

Issue	Possible Model Characteristics	Issue	Possible Model Characteristics	Issue	Possible Model Characteristics
Latency	*Temporal diagrams, functions, processors*	**Unauthorised (Function/Data) Access**	*Functions, data, components*	**Physical failure**	*Components*
Deadline miss	*Temporal diagrams, I/O events*	**Repudiation of Transaction**	*Interaction (messaging) models*	**Incorrect value**	*Data, I/O*
Throughput	*Communication lines, processors*	**Assumed Identity**	*Actors, messages*	**Data integrity**	*Data, messages*
Functionality unavailable	*Function*	**Comfort**	*Actors*	**Incorrect state**	*State diagrams*
Mistake	*Actors, input*	**Fatigue**	*Actors*		

Completion of the stage implies that all dependability issues have been associated to a suitable deviation. Failure to associate all issues to a deviation will compromise the completeness of the analysis, as possibly important issues will be overlooked.

4.7.4 Identification of Applicable Deviations

The suitable deviations defined in the previous stage are generic, relating to the identified dependability issues. In this stage of the DDA the applicability of suitable deviations to the system under analysis is examined. The stage is responsible for two main functions; conformance of suitable deviations templates and documentation of applicable deviations.

Templates documenting suitable deviations are created with the purpose of identifying deviations capable of revealing the dependability issues. It may be the case that the available models that represent the system to be analysed do not contain the system element type described in a suitable deviation. Hence, although a suitable deviation may exist for a particular issue, it can not be applied on the actual system being

examined. Possible inconsistencies should be detected and corrected before using the template.

Additionally in this stage all the applicable deviations which will be analysed are created and recorded. The system elements represented by the system element type in a suitable deviation are instantiated and an applicable deviation is created for each one of them. Fig.4.13 illustrates the overall process of this stage.

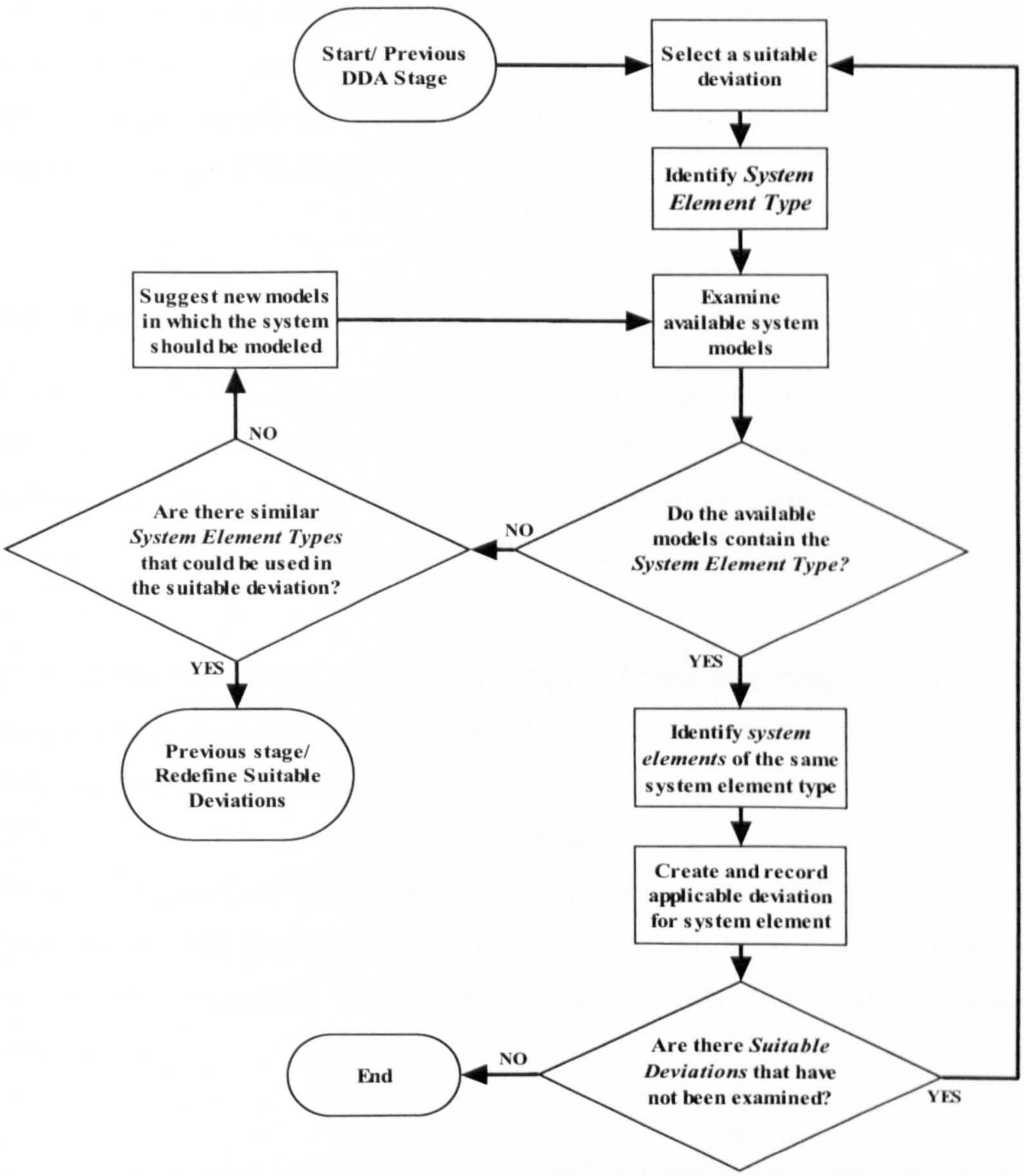

Fig.4.13 – Identification of Applicable Deviations Stage Process

The stage starts with identification of the system element types required by the defined suitable deviations. Next the participants study the models in which the system (under analysis) is represented. Subsequently to successful identification of the **same** system

element types an applicable deviation is created for each system element of that type. Recognition of an applicable deviation is captured in the metamodel by setting the *isApplicable* attribute of the *SuitableDeviation* class to true. The process repeats until all suitable deviations have been examined.

Ensuring that the system element type in the documented suitable deviations and the available models are the same (i.e. represent the same concepts) is highly important in order to avoid inconsistencies. In case a model contains system element types similar but not the same as the suitable deviation, the suitable deviation should be redefined with the available system element type in mind. Upon completion, the defined applicable deviations will be examined whether leading to a credible failure condition during the following DDA stage.

4.7.5 Identification of Failure Conditions

This stage involves failure analysis of the system by examining the applicable deviations. The overall purpose of the step is to identify and understand the relations between all credible failure conditions. The resultant failures will then be analysed in the next stage in order to understand how a failure condition can cause another.

Fig.4.16 shows the process of this DDA stage. At the beginning of this stage the participants identify the available models in which the system is represented and the system elements that are contained in the model. In this example we analyse the AGO MODAF OV-2 (node connectivity) and OV-5 (activity diagram) models, which consist of Table.4.8, Fig.4.15 and Table.4.9 and Fig.4.16 respectively. UML was chosen as the language in which to implement the MODAF models as it is easier to understand, and more broadly recognised in comparison to the MODAF specific graphical representations.

Next, a system element is selected to be analysed, for which there exists an applicable deviation that has been associated with it (during the previous stage of the DDA). The analysis focuses on two types of elements for the respective models; *needlines* and *activities*, which we probe for deviations using the defined guidewords, in order to identify possible failures. As in 'traditional' HAZOP, it is important for the analysts to

understand the model and the intended behaviour of the system, so as to be able to describe the effect of the deviation. MODAF describes a needline as "*A relationship specifying the need to exchange information between nodes. The needline does not indicate how the transfer is implemented*". An activity is described as *"a process carried out by a person or organisation"* [89].

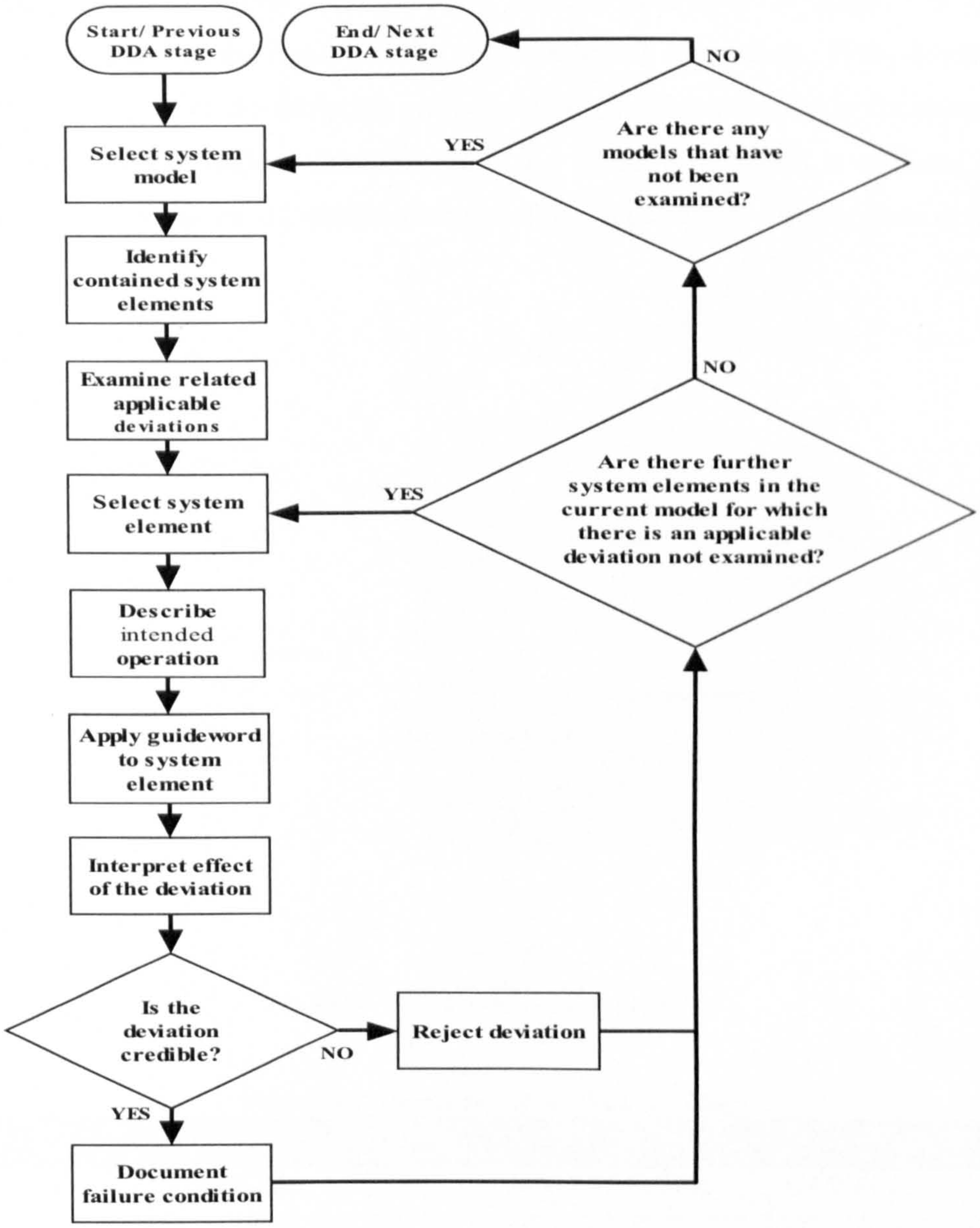

Fig.4.16 – Identification of Failure Conditions Stage Process

Familiarisation with the models and the intended operation of the system is an important facet of the analysis. SHARD also highlights this necessity [76]. This task assists understanding the contribution of the element being prompted to the overall operation of the system. Contrary to previous methods such as SHARD which analyse specific

model elements of the system, the focus of the DDA may differ. For example whereas a needline focuses on abstract data exchange between participating systems, an activity is concerned with functional contribution of each system to the overall mission. Following the description of the intended system operation, participants prompt the system element with the guidewords, as suggested by the respective applicable deviations. The guidewords suggest deviant behaviour of the system element. Analysis participants should examine the resultant system behaviour as suggested by the deviation and describe the impact on the system element's operation. If the deviation is credible, the impact of the deviation on the system element operation is documented as a possible failure condition. Identification of a credible deviation is captured in the metamodel by setting the *isCredible* attribute of the *ApplicableDeviation* class to true.

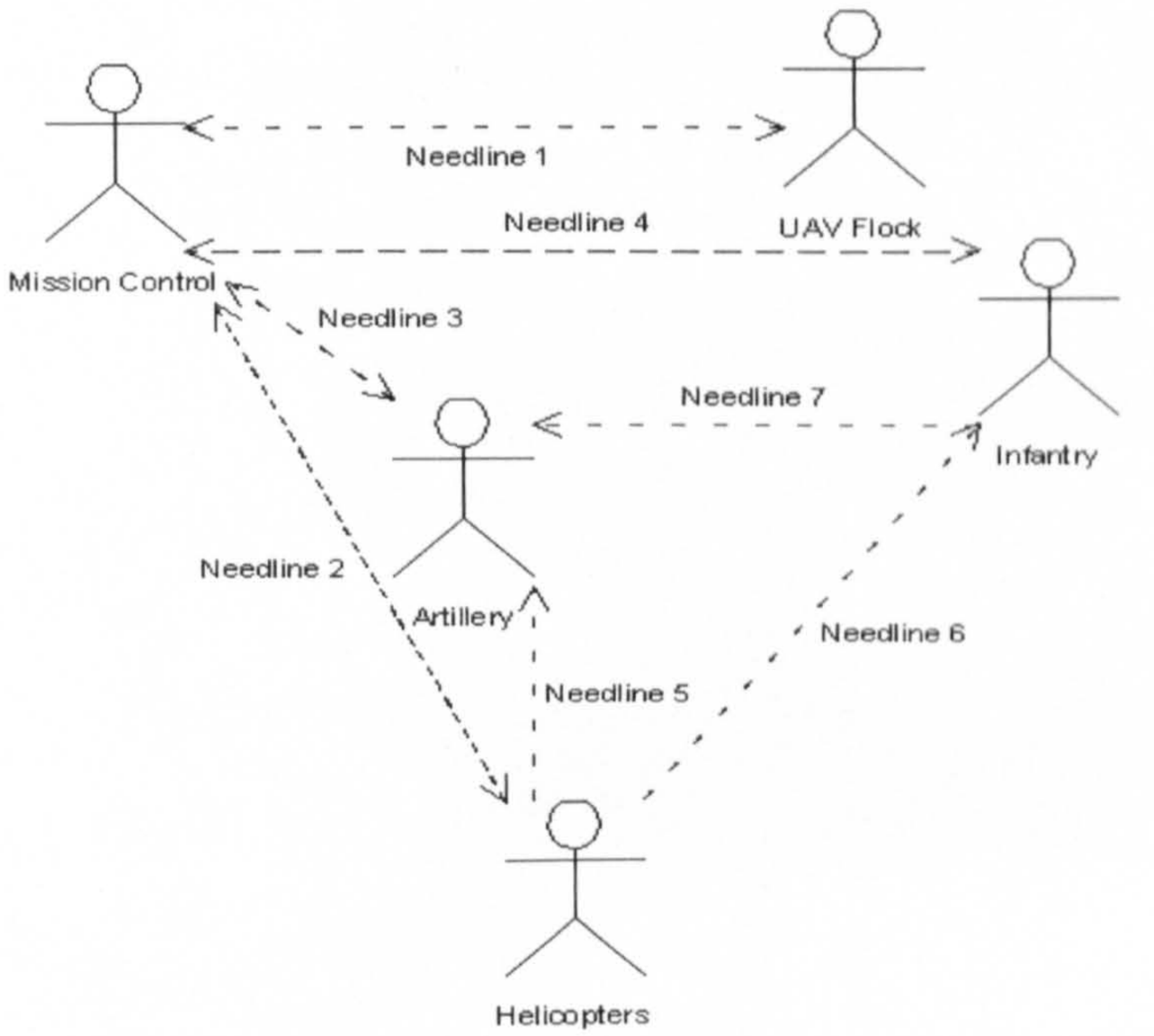

Fig.4.15 – OV2 for the AGO using UML

Table.4.8 – Description of the OV2 Needlines

Needline ID	Information Exchange	Producer	Consumer
1	Flight path	Mission Control	UAV Flock
	(Fusion) Sensor data	UAV Flock	Mission Control
2	Map	Mission Control	Helicopters
	Target area	Mission Control	Helicopters
	(Fusion) Feedback	Helicopters	Mission control
6	Theatre support	Helicopters	Infantry
7	Target location	Infantry	Artillery

Table.4.9 shows summary of this stage in tabular form. The first and second columns define the system element under analysis. In this case all system elements are of the same system element type (i.e. needline). The third column shows the guideword with which the system elements are probed; the parentheses underneath the guidewords indicate the dependability attribute from which the deviation has descended.

Table.4.9 – OV2 Failure Identification

System Element Type	System Element ID	Guideword	Failure Condition	Failure Condition ID
Needline	1	Public (Security)	Disclosure of aircraft position	FC1
	7	Overload (Performance)	Slow transmission of target data	FC2
		Fake (Security)	Artillery receive fake target data with malicious intent	FC3
		Public (Security)	Enemy receives firing intent and target information	FC4
		Omission (Availability)	Artillery will not receive any target data	FC5
		Value (Reliability)	Artillery will receive the wrong location/order	FC6
		Late (Performance)	Delay or possibly loss of request of target data	FC7

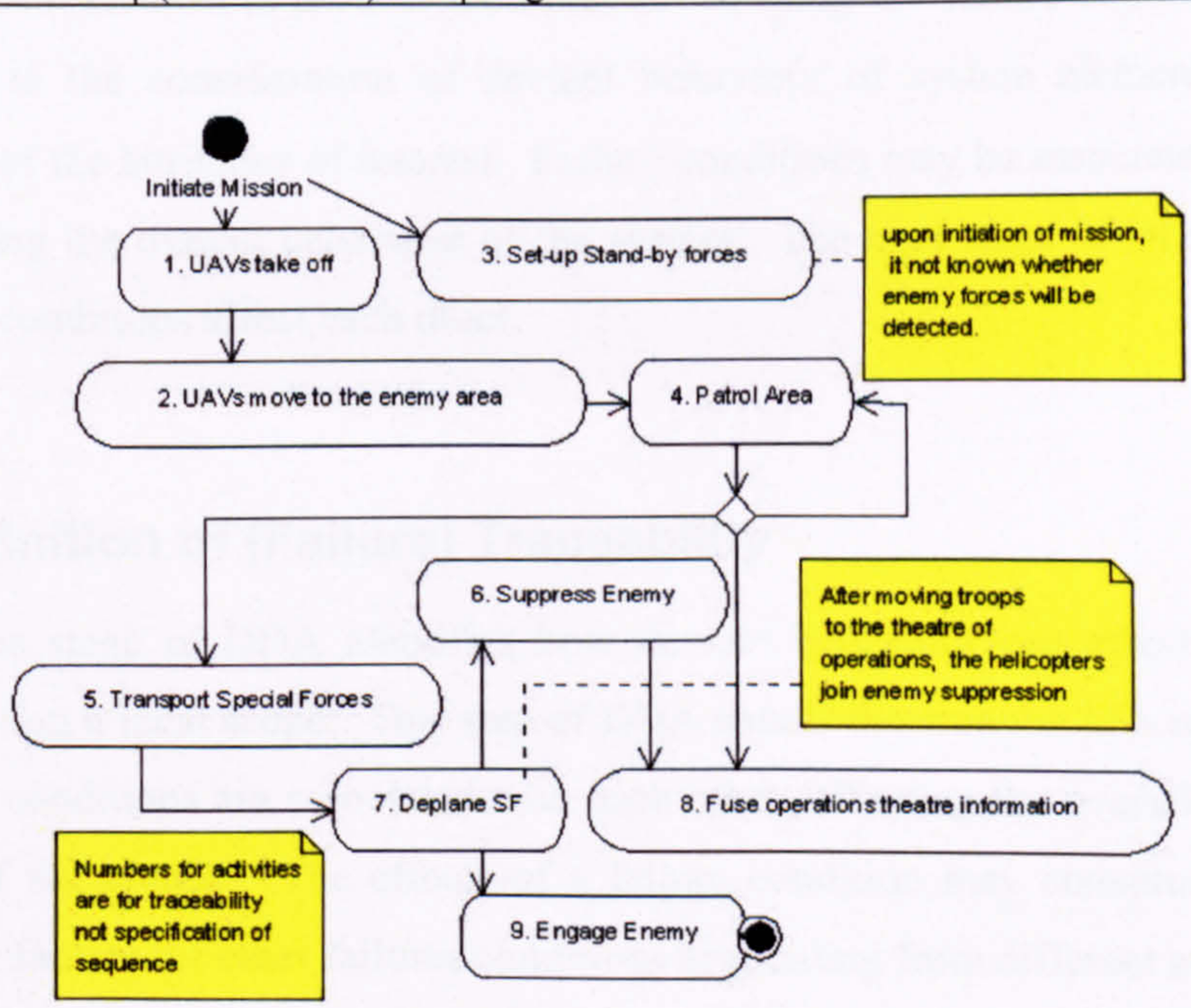

Fig.4.16 – OV-5 Node Connectivity Diagram Using UML

Taking in mind the dependability attribute of origin at this stage contributes in defining the viewpoint for the system's operation, and helps participants to describe the impact of the deviation unambiguously. Finally the last two columns of the table document the

failure condition. Similarly, Table.4.10 summarises the (credible) deviations and failure conditions for the activity system element types of MODAF product OV5.

Table.4.10 – OV5 Failure Identification

System Element Type	System Element ID	Guideword	Failure Condition	Failure Condition ID
Activity	4	Omission	There is no patrolling function available, cannot notify of enemy forces	FC8
	4	Mistake	Users mistakenly identify enemy	FC9
	6	Mode	A location is not suppressed when it is being expected that it does	FC10
			A location is suppressed when it should not	FC11
	6	Late	Delays in suppressing enemy	FC12

Scoping is important at this stage; the description of a failure's effect should be limited to the model that is being analysed. Interpretation of the failure condition depends on how the system element is used in each model. Scoping the failure condition DDA is not limited to the consideration of deviant behaviour of system elements from the perspective of the attributes of interest. Failure conditions may be associated with each other affecting the overall behaviour of the system. The next stage of DDA examines how failure conditions affect each other.

4.7.6 Definition of (Failure) Traceability

The previous stage of DDA identifies how deviant behaviour can affect the system elements within a local scope. This step of DDA entails the creation of a map showing how failure conditions are associated with each other, affecting the overall dependable operation of the system. The effects of a failure condition may constitute causes or contributing factors for other failures conditions originating from different attributes and models. Traceability between failure conditions is necessary in order to understand the associations between failure conditions. This involves consideration of how the system elements are combined composing the overall system, which inevitably requires examination of the system models. Fig.4.17 presents the process overview of this DDA stage. Initially the participants identify a system model (model A) and its respective

failure conditions and a system model associated with it (model B). Next the failure conditions of 'model A' are examined, initially for their potential to directly impact the concerns of the stakeholders (identified in the 2nd DDA stage).

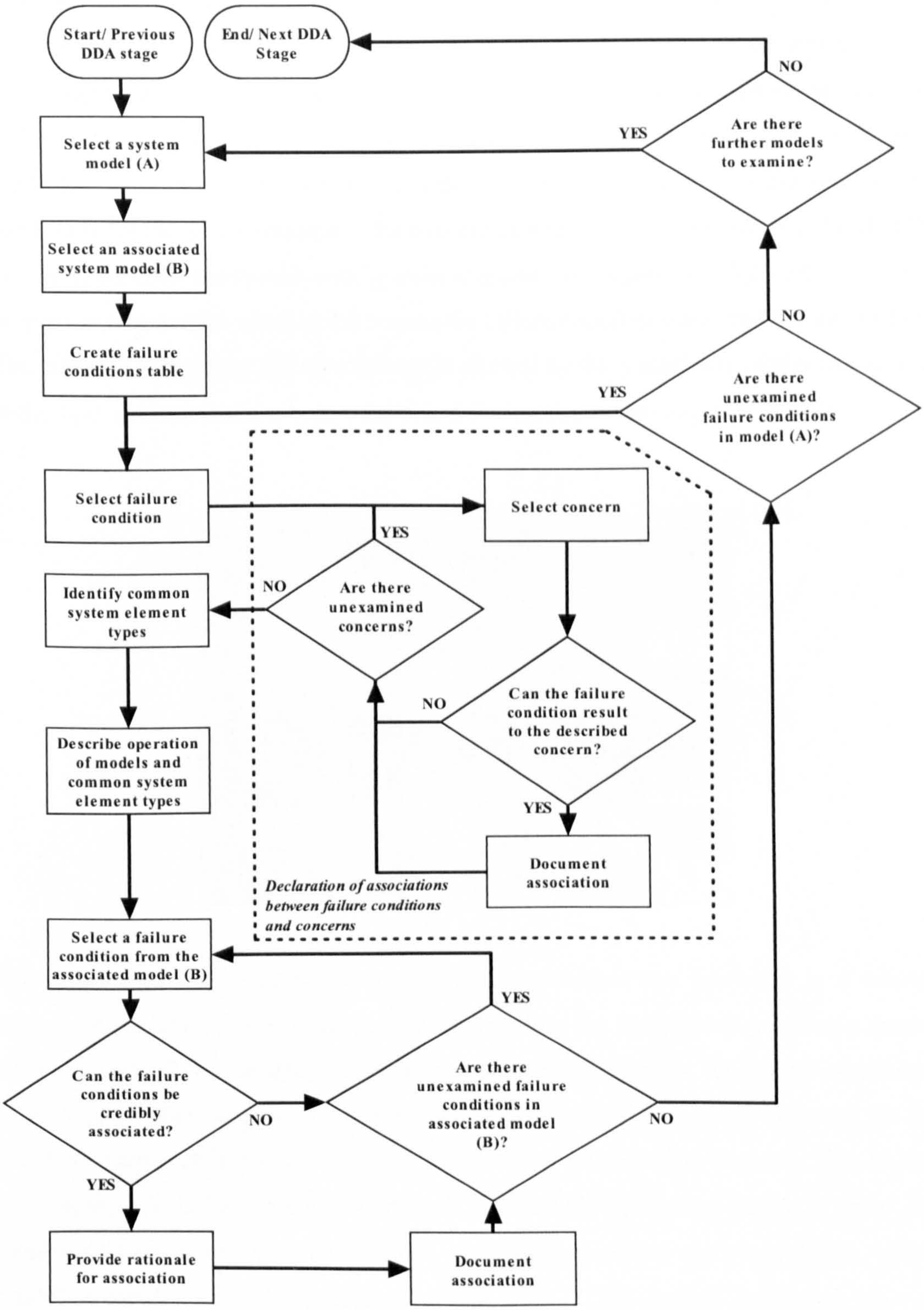

Fig.4.17 – Definition of (Failure) Traceability Stage Process

Associations between failure conditions and concerns are declarative and not the product of an analytical method. This happens because concerns are not associated with a system element but with the overall operation of the system – a fundamental difference of the concepts of failure conditions and concerns. Hence effects of a failure condition cannot be **traced** to concerns using the models of the system. In practice this implies that such association is based on the subjective judgement of the stakeholders, evaluating what constitutes direct compromise of their concerns. Following association with concerns, associations with failure conditions of related models are examined. Table.4.11 provides a summary of possible associations between the identified failure conditions for the AGO scenario. The two dimensions of the matrix are populated with failure conditions associated with system elements belonging to OV2 and OV5. Its purpose is to examine whether the respective failure conditions are credibly associated. The rationale supporting the associations is elicited by the traceability model (explained in the next section) that has been established during the earlier stages of the DDA.

Table.4.11 – Associations Between Failure Conditions in OV2 and OV5.

OV2\OV5	FC8	FC9	FC10	FC11	FC12
FC1					
FC2					X
FC3		X		X	
FC4					
FC5		X	X		X
FC6		X		X	
FC7		X	X		X

Although the associations between the failure conditions are presented in a tabular format, the results are also recorded by instantiating the metamodel (creating a model for the AGO DDA example). One of products of this stage is the failure conditions map. The failures condition map (Fig.4.20) showing the failure conditions map for the AGO scenario) is a visual representation of the associations between the failure conditions. A failure conditions map is created by generating (directly from the metamodel) code for the Graphviz tool which then produces the visualisation of the model, created during the processes of the DDA. The failure conditions map of Fig.4.18 shows the deviations (ellipses) that prompted the identification of the failure conditions (rectangles).

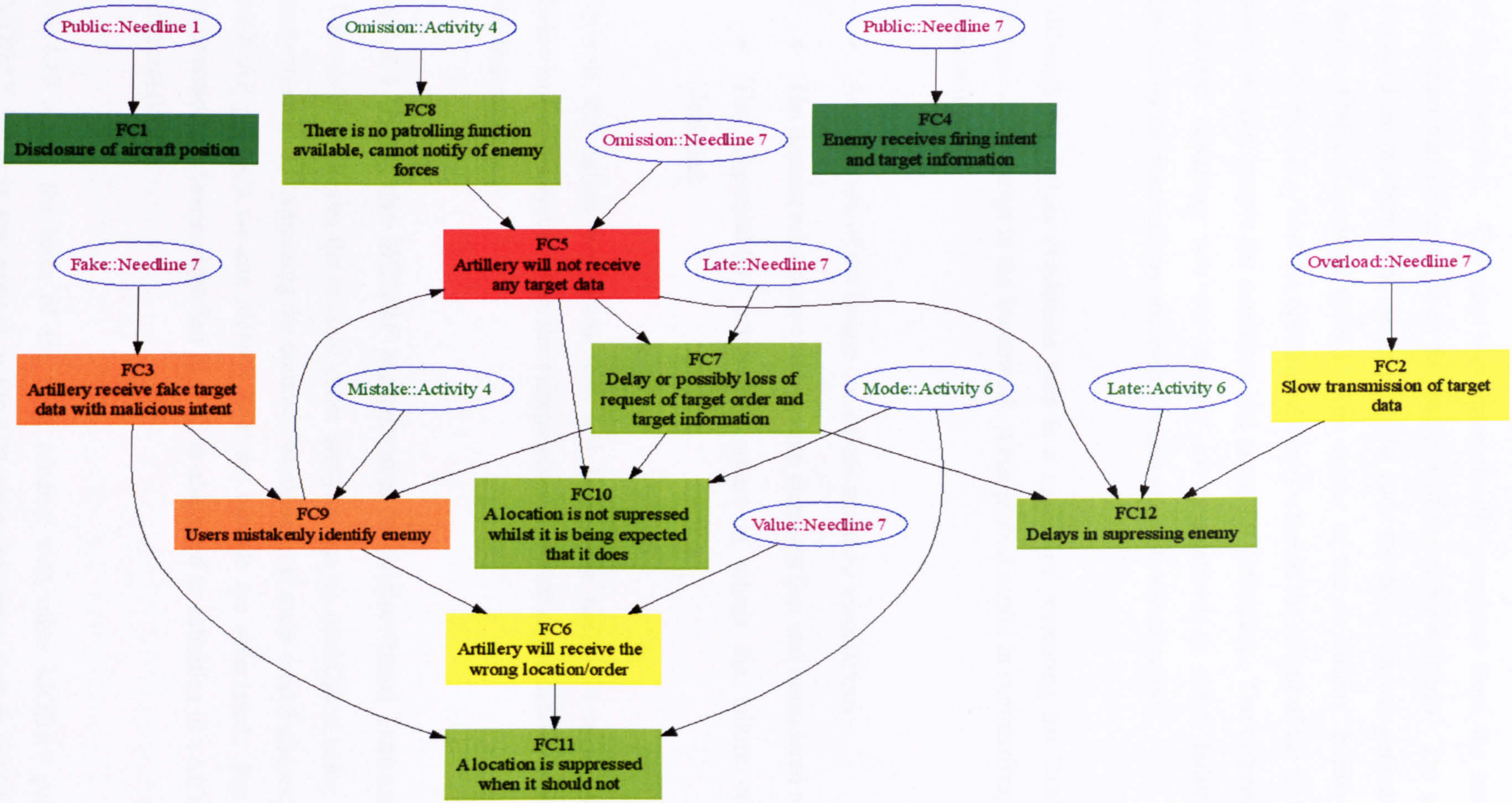

Fig.4.18 – Failure Conditions (FC) Map for the AGO Scenario.

Concerns can also be shown but have been left out in the example to preserve the clarity of the illustration. The map was automatically generated from the instance of the metamodel that corresponds to the analysis of the AGO scenario. The graphviz tool was used for production of the graph. The code for the graph was generated using the Epsilon Object Language (EOL). The colour of the deviations' border denotes the MODAF view (i.e. blue for operational view) whereas the colour of the text the element examined (i.e. purple for needlines and green for activities). The criterion for failure condition colouring was the number of associations to other failure conditions (including the associations not shown in this example visualisation).

Although the failure conditions map is a means of presenting the DDA, and not a fundamental concept in the framework, it has proved useful in summarising the analysis by showing:

- An overview of the failure conditions and their associations
- The deviant behaviour of the system elements that was considered important.
- The (dependability attribute) context in which the failure condition was identified.

Overall the failure conditions map has been very useful in identifying how the behaviour of the system from the viewpoints of different dependability attributes affects the overall system.

4.7.6.1 Using the MODAF Metamodel to Understand Traceability

Traceability between the models under analysis can be established using the MODAF meta-model. By examining the common elements (i.e. meta-model classes) between the MODAF products we can identify how the models are associated. For example, an information exchange identified in OV2 is also used in activities in a MODAF activity diagram (OV5).

Fig.4.19 shows the links of the OV2 product with other MODAF products. The MODAF products that appear in Fig.4.21 share common classes, which are used as prescribed by the specification of each product.

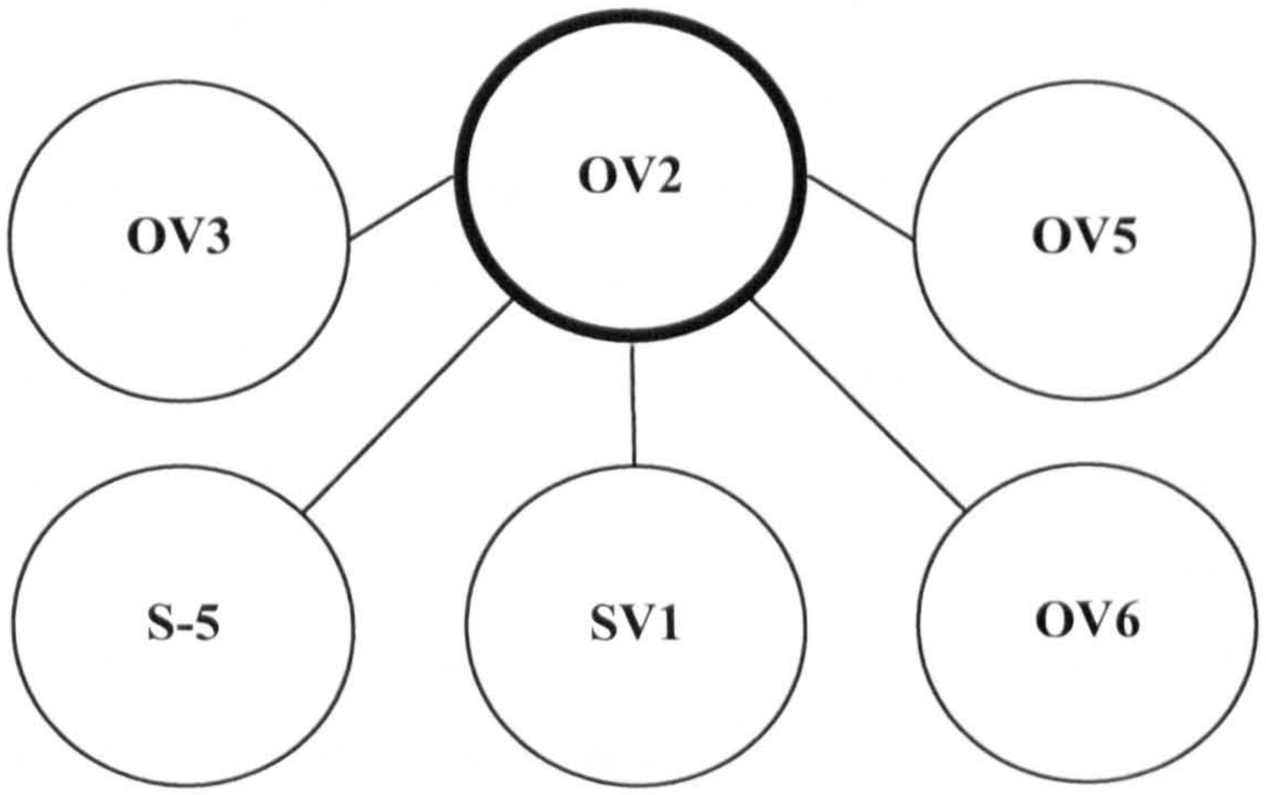

Fig.4.19 – Associations of the OV2 MODAF Product

Table.4.12 summarises the common classes between the MODAF products, which resulted in identifying the relationships of OV2 (Fig.4.21).

Table.4.12 – OV–2 Traceability Matrix

OV5	OV3	OV6	SV5	SV1
-InformationExchange *-Node* *-OperationalActivity* *-ActivityConductedAtNode*	*-InformationExchange* *-Needline*	*-NodeAssembly* *Usage* *-Node* *-OperationalActivity*	*-OperationalActivity*	*-Node*

Although the products are not explicitly associated, the common classes allow us to identify and understand the role of a system element from other viewpoints of the system operation. For example, a performance failure at the needline may give the artillery a command to engage the enemy at the wrong time. Hence the effect of a performance failure regarding the needline constitutes the cause for a reliability failure with respect to the activity. *TextualTraceabilityOfEffect* is the class that represents the justification of how system elements are used in the system and how a failure condition may result in another.

4.7.6.2 Semi-automated Approach for Establishing Traceability

Textual representation of traceability presents certain disadvantages in its use. Firstly, description of traceability is ad-hoc and depends on the DDA participants for clarity. This can result in complex and often weak reasoning. The problems of text based reasoning have been well documented by experiences in the safety case domain in [47].

Moreover identifying the association of a system element with others manually is a tedious and time consuming task. Automation of the task even to some degree can offer significant reduction in the time required to complete the task. This was achieved by building a basic dependency model for system elements. Fig.4.20 shows an extract from the revised DCM with the addition of the class and the associations supporting the basic traceability model. The difference with the initial extract of the DCM capturing DDA, is the addition of the *UsedByTraceability* class. This class serves as an association class between system elements. The association class denotes the dependencies between the system elements during operation. Consider OV2 (Fig.4.17); needline 2 depends on needline 1, hence needline 1 is *used by* needline 2. Associations between system elements are declared during definition of the system models.

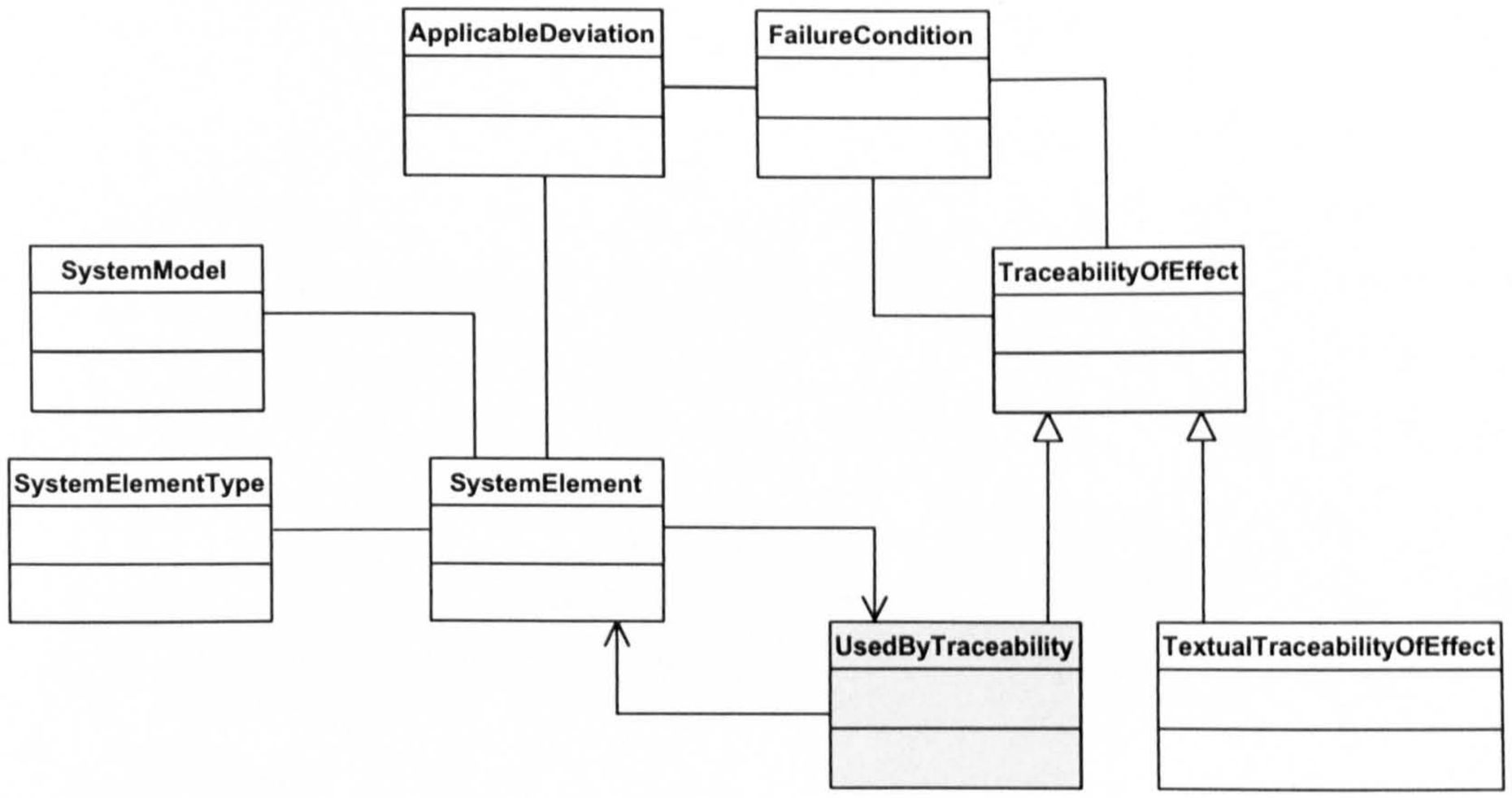

Fig.4.20 – "Used-by" Association in UML

Participants can use a variable degree of granularity in which the model is represented. For example only the needlines could be modelled, omitting the system elements of type *node*. However if more detail is required the intermediate system elements can be added; furthermore, this approach of providing traceability can be extended by 'hardwiring' to the dependability case metamodel, classes of the modelling framework in which the system is represented, that correspond to lower level, and more detailed models. Epsilon scripts were created to benefit from the provided basic traceability in the following manner: by selecting the system elements (consider a hypothetical system element A) the deviant behaviour of which triggered the definition of a failure condition, the script identifies the system elements (consider hypothetical system

elements B and C) that are using system element A, by navigating the *UsedByTraceability* associations. The script then identifies the failure conditions that resulted from deviations associated with system elements B and C. The two resultant sets of failure conditions are suggested to the participants to examine the credibility of possible associations between a pair of failure conditions belonging to each set. The participants do not have to exhaustively check the combinations of all failure conditions. The pairs prompted by the script include only failure conditions that could be associated based on traceability. Providing this degree of automation in the analysis proved to be very efficient, especially during analysis of large scale systems.

4.7.7 Definition of Dependability Profile and Preliminary Identification of Goals

This is the last stage of the deviation analysis. Having understood how unwanted behaviour can affect the operation of the system elements and the system overall, participants elicit tangible dependability requirements. For the system elements, the identified requirements constitute the dependability profile, which can be characterised as a multi-attribute envelope of intended operation. Furthermore, this stage serves as a preparatory stage, in which the participants are involved in a preliminary identification of goals that will form the dependability argument. Fig.4.21 shows the overall process of the final DDA stage. The stakeholders motivated by identification of unwanted (deviant) behaviour define dependability requirements. Concerns and failure conditions are the concepts that capture the unwanted system operation. Consequently, the process can be seen as having two threads corresponding to concerns and failure conditions.

The process starts with selection of a concern or failure condition. Although either can be chosen in the beginning, ideally all concerns should be examined first. Priority of concerns over failure conditions is something that will occur naturally by applying DDA during the system's lifecycle.

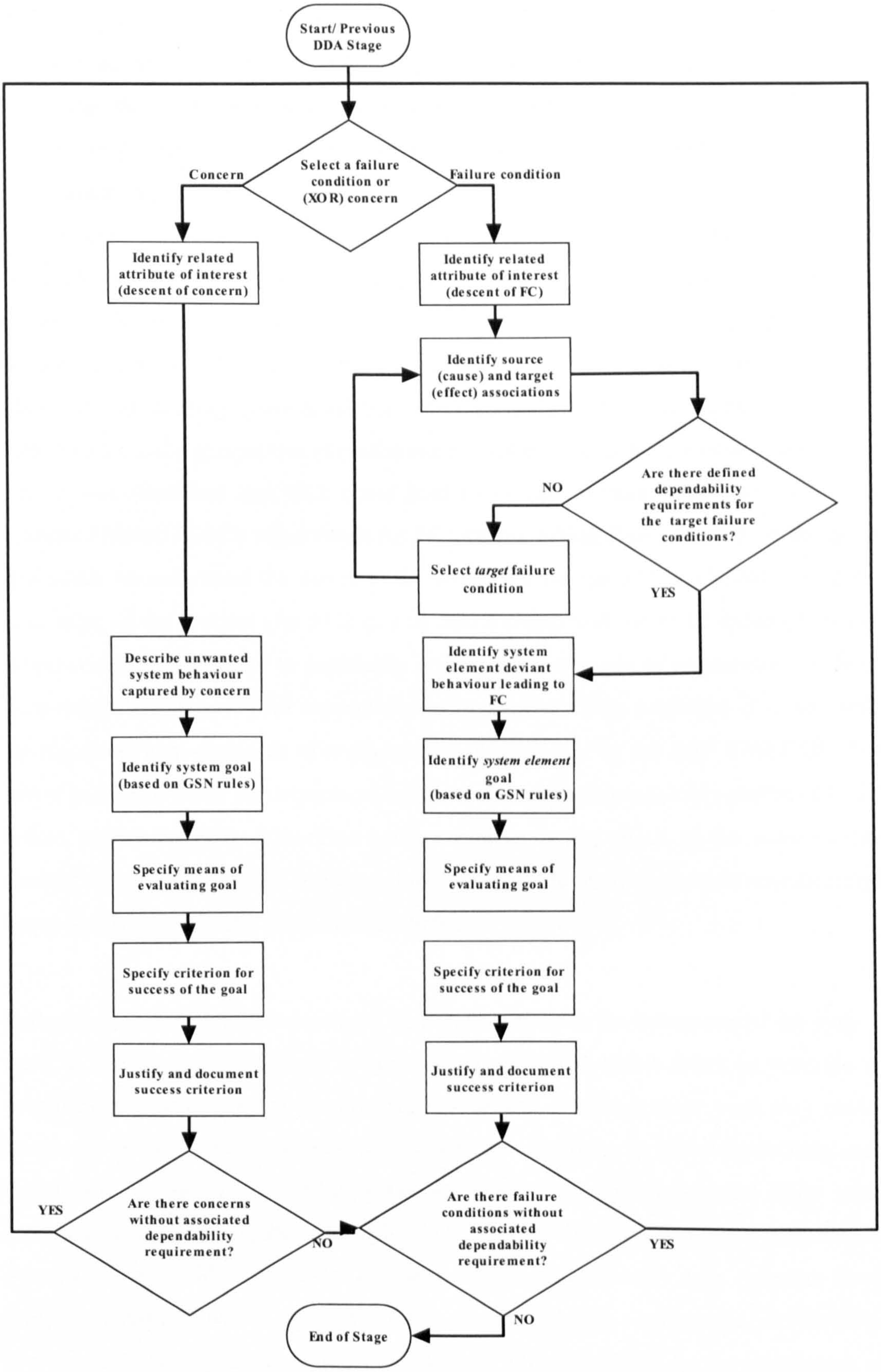

Fig.4.21 – Definition of Dependability Profile Stage Process

Next, the participants should identify the attribute of interest with which the concern or failure conditions is associated, providing the appropriate perspective to the analysis. Following this, the participants identify the unwanted behaviour captured by the concern (e.g. loss of life) or the unwanted behaviour of the system element that corresponds to the failure condition (e.g. public access of needline 1 data). A prerequisite in this step, is that the target associations of the examined failure condition should have an already defined dependability profile. This is a necessary step that allows the dependability requirements to be elicited in the context of operation of the system. Consider failure conditions FC2 (slow transmission of target data) and FC12 (delay in suppressing enemy) of the AGO scenario, which are failure conditions identified from the perspective of performance. Subsequent to the creation of the failure map it was identified that FC2 could lead to FC12. If during the process FC2 is examined before FC12 a requirement for FC2 cannot be justifiably derived. Although it is possible to understand the nature of the requirements that will be elicited during the next steps of the process (for FC2 rate of transmission and for FC12 speed of enemy suppression), it is difficult to justifiably project the target rate of transmission without knowing the target speed for suppression of the enemy. This constraint is in line with the top-down apportionment of requirements as suggested by the ARP 4761 [70]. For this reason the process participants will have to define the dependability profile of FC12 before examining FC2. Concerns are not subject to this check as the requirements elicited from concern would correspond to the highest level of the system operation and hence are defined directly from the stakeholders.

Following identification of unwanted operation, goals for the behaviour of the system need to be identified. This is a preliminary effort for stakeholders to describe in tangible terms goals derived from the DDA analysis. Although these goals may not be integrated 'as-is' in the arguments, they constitute candidates for the dependability case arguments. Stakeholders should define the goals using the syntax and scope rules guidelines described in GSN step 1 [47]. In the case of the AGO scenario a goal elicited from concern would be *'Overall scenario should be acceptably safe'* whereas for a failure condition would be *'Needline should provide acceptable bandwidth'*. Identification of the goal should not be confused with the criteria for the satisfaction of the goal, something also highlighted in the GSN methodology. In the case of FC2, specifying that needline 7 should have 1 Mbps bandwidth is not a correct goal. The goal is the provision of adequate bandwidth and 1 Mbps is the target requirement for the

satisfaction of the goals. The two next steps of the process are responsible for identifying the means of evaluation of the goal (risk for the goal regarding acceptable safety and transmission rate for adequate bandwidth) and the goal satisfaction criterion (1 death in 10^4 hours of operation, 1 Mbps). Finally stakeholders need to present the rationale based on which the acceptance criteria of the goals were specified. For failure conditions this may prove to be easier as it depends on the collaboration of system elements. For concerns such rationale may be justified by appeal to sources such as business objectives, mission plans and legacy systems.

Upon concluding the process the system stakeholders will have identified requirements associated with system elements and with concerns, as well as a set of goals that could be used in the dependability case arguments. The set of the dependability requirements associated with one system element constitute the dependability profile of that system element.

4.8 Other Sources of Requirements

The main purpose to of DDA is to acknowledge the dependability attribute viewpoints and understand how the system should behave with respect to those dependability viewpoints. Application of DDA does not single out use of other analysis methodologies. Instead other method can be used to complement the analysis. This is one of the reasons for the adoption of the dependability profile. Using the dependability profile requirements originating from other methodologies can be recorded and later used in the dependability case. The DCM is extensible to accommodate other processes for requirements elicitation, which however acknowledge and conform to the multi-attribute view (requirements for each attribute) used in this thesis.

On the same tack, the goals elicited during application of DDA are not the only goals that will be used in the dependability case arguments. The goals elicited during DDA contribute in establishing product argument(s). This means that they mainly related to the actual system (i.e. product). According to the architecture of the arguments and the strategies based upon which the argument is developed, various types of goals can be used such as goals asserting the correctness of the system process. For example, consider an argument about quality control of the lifecycle processes (process

argument). The goals elicited during DDA are but a subset of the total goals used in the final dependability case artefact.

4.9 Summary

This chapter has defined Dependability Deviation Analysis, a technique for eliciting dependability requirements. DDA supports identification of system requirements from the perspective of any dependability attribute. In order to understand how a dependability attribute can influence another, the concept of failures maps was introduced. In order to capture and collate the requirements of the SoS elements DDA introduces the concept of dependability profiles. Finally the DDA stages are demonstrated throughout the chapter using the AGO scenario.

Intentionally Blank

Chapter 5

Facilitating Trade-offs Between Dependability Requirements

5.1 Introduction

A dependability case entails arguments about achievement of dependable operation of the system. This involves reasoning about achievement of acceptable system behaviour with respect to the dependability attributes of interest to the stakeholders. Establishing an argument about the dependability of the system involves justification and documentation of the decisions made during system evolution, in order to achieve the required dependability attributes. However, decisions that favour achievement of an attribute may conflict with other attributes. These occasions are common and inevitable especially in large systems, resulting in an impasse. Resolving such situations often necessitates trading-off the achievement of a requirement in favour of another. Decision making is an integral part of the development of a system. This chapter examines concepts applied in decision making, examines application of one such methodology in the context of dependability cases, and finally it introduces the trade-off method (TOM), an argument based method that facilitates trading-off conflicting goals.

5.2 Trade-offs During Evolution of the Dependability Case

Construction of a compelling argument requires explicit consideration of contextual information such as the design of the system or possible assumptions made during the creation of the argument. For example, arguing about a requirement concerning the availability of a system could result in adopting an architecture with redundant components. Hence, that design decision would allow an argument to be constructed combining the availability of the two components, assuming that the components will not fail simultaneously. Similarly, tackling this last concern may lead to adopting a diverse implementation of the two components, avoiding simultaneous failures due to implementation faults.

Evolution of requirements and design takes place in parallel. A representative example of this is the twin peaks model [91]. During the evolution of a system, developers are continuously faced with decisions which need to address the already specified requirements and serve as the basis for the derivation and apportionment of more detailed requirements. Satisfying the specified requirements of a system can be achieved with a number of possible different alternative designs.

Degree of satisfaction of a goal is the degree of achievement of a goal by a candidate design alternative, with respect to the defined criteria of achievement.

Variability on how each design alternative can satisfy the system goals implies that any decision will inevitably result in trading-off satisfying a set of goals in favour of others. **Trade-offs** can be described as a *"balance achieved between two desirable but incompatible features; a sacrifice made in one area to obtain benefits in another; a bargain, a compromise"* [92]. Generally, in order to make such decisions, system developers need to examine the advantages or disadvantages of each design alternative on the operation of the system, aiming to choose the optimal.

5.3 Review of Concepts and Methodologies in Decision Making

There are numerous methodologies and approaches that have been employed by system analysts, not only in the engineering domain but also in other disciplines. Some of the main concepts in decision making are common in more than one methodology. However even subtle differences can signify a distinctive approach towards thinking about decision making, and in specific about managing the process of making trade-offs.

5.3.1 ATAM

ATAM is a method developed by the Software Engineering Institute in Carnegie Mellon University [37]. The purpose of the method is for participants to propose candidate architectures and prioritise them according to which will satisfy the best their goals. The stakeholders' goals that the system needs to achieve are defined during a

complementary process to ATAM, the Quality Attribute Workshops [39]. The aim of these workshops is to brainstorm scenarios about the use of the system, and the required reaction of the system to certain events defined by the scenario. For example, in the context of a network system, in the event of a hardware failure the network should detect and recover the failure within 1.5 seconds. The output of the process is a utility tree, which provides a top down model for directly translating business drivers of a system into quality attribute scenarios. The overall objectives of the system are decomposed into more concrete criteria. Fig.5.1 presents an example extract from a quality attribute utility tree with final criterion from the point of security and, in particular data confidentiality. The resultant criteria are characterised by a priority and risk indication according to the projected importance to the stakeholders' interest and their risk (indicating difficulty) of implementation.

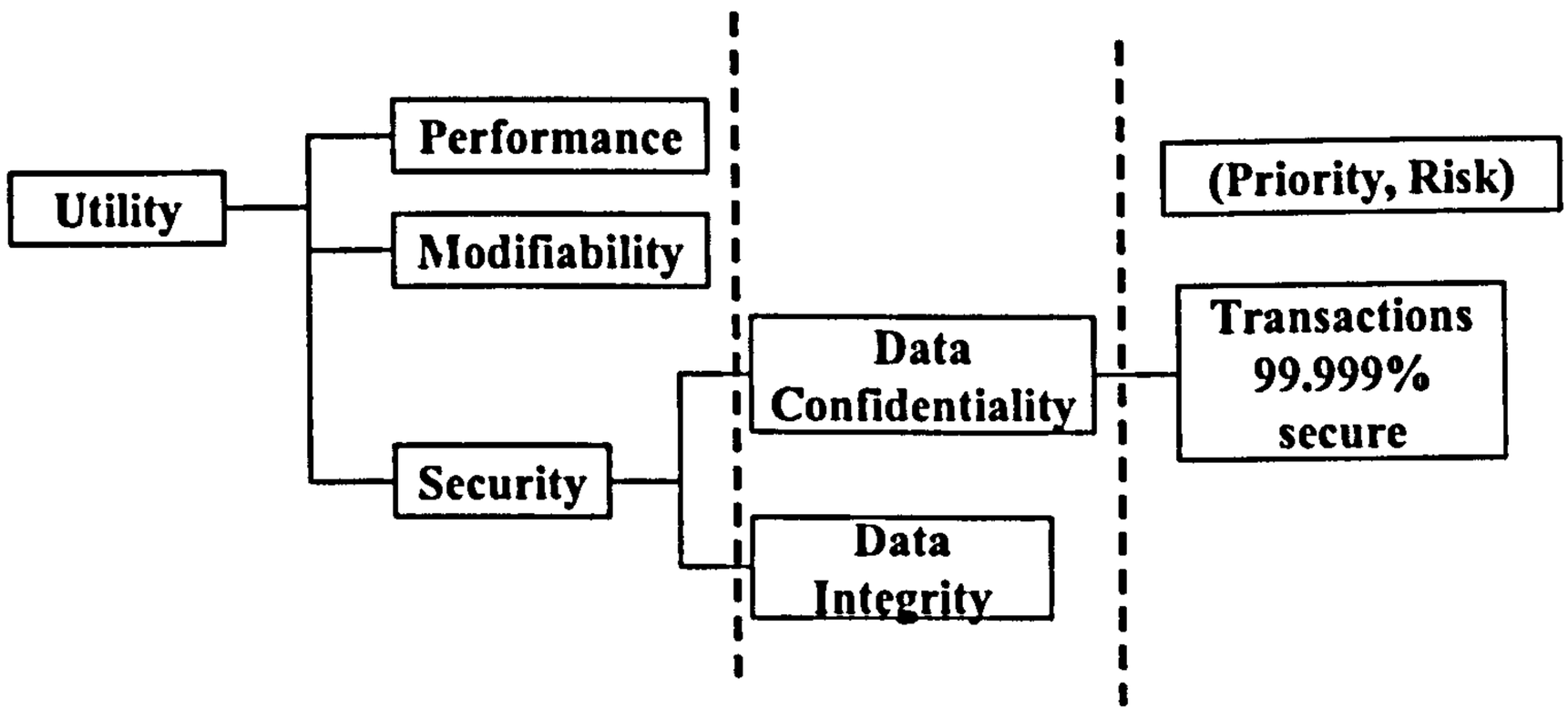

Fig.5.1 – Quality Attributes Utility Tree

The participants of ATAM discuss the possible architectural approaches and how they can contribute to achieving the elicited scenarios. The final selection of the appropriate architectural strategies takes place after voting conducted by the stakeholders, selecting the strategy with the highest score. Voting takes into account the priority of achieving each scenario as well as possible risks. An important activity of ATAM is the identification of sensitivity points. Sensitivity points are architectural decisions which significantly affect the achievement of the attributes. An example of a sensitivity point given in [86] is the level of confidentiality in a virtual private network, which is sensitive to the number of bits of encryption. Identification of sensitivity points allows the participants to understand the architectural characteristics responsible for the

achievement of the stakeholders' goals. The objectives are to analyse and communicate the objectives of a system and understand the means of their achievement.

5.3.2 Cost Benefit Analyses

In general, cost benefit analyses compare the expected return of an investment against the initial monetary value required of the implementation of the alternative. One application of the principles of cost benefit analysis in systems engineering is the Cost Benefit Analysis Method (CBAM) [86]. CBAM compares the benefit from implementing an architectural strategy against its implementation cost. Ultimately, the purpose of the implemented architectural strategy is to achieve the required quality attribute. The return of investment is the calculation of the achieved benefit over the cost required to achieve that (benefit / cost). Return of investment is a numerical measure used in CBAM. In order to calculate it (since cost is also numerical) CBAM suggests quantification of utility. Utility represents the benefit with respect to the required quality attributes resulting from implementing a candidate architectural scenario. Fig.5.2 presents a schema of how the return of investment is calculated. The stakeholders assess the improvement in the system in terms of the required quality attributes and subjectively specify the utility on a scale from 0 to 100.

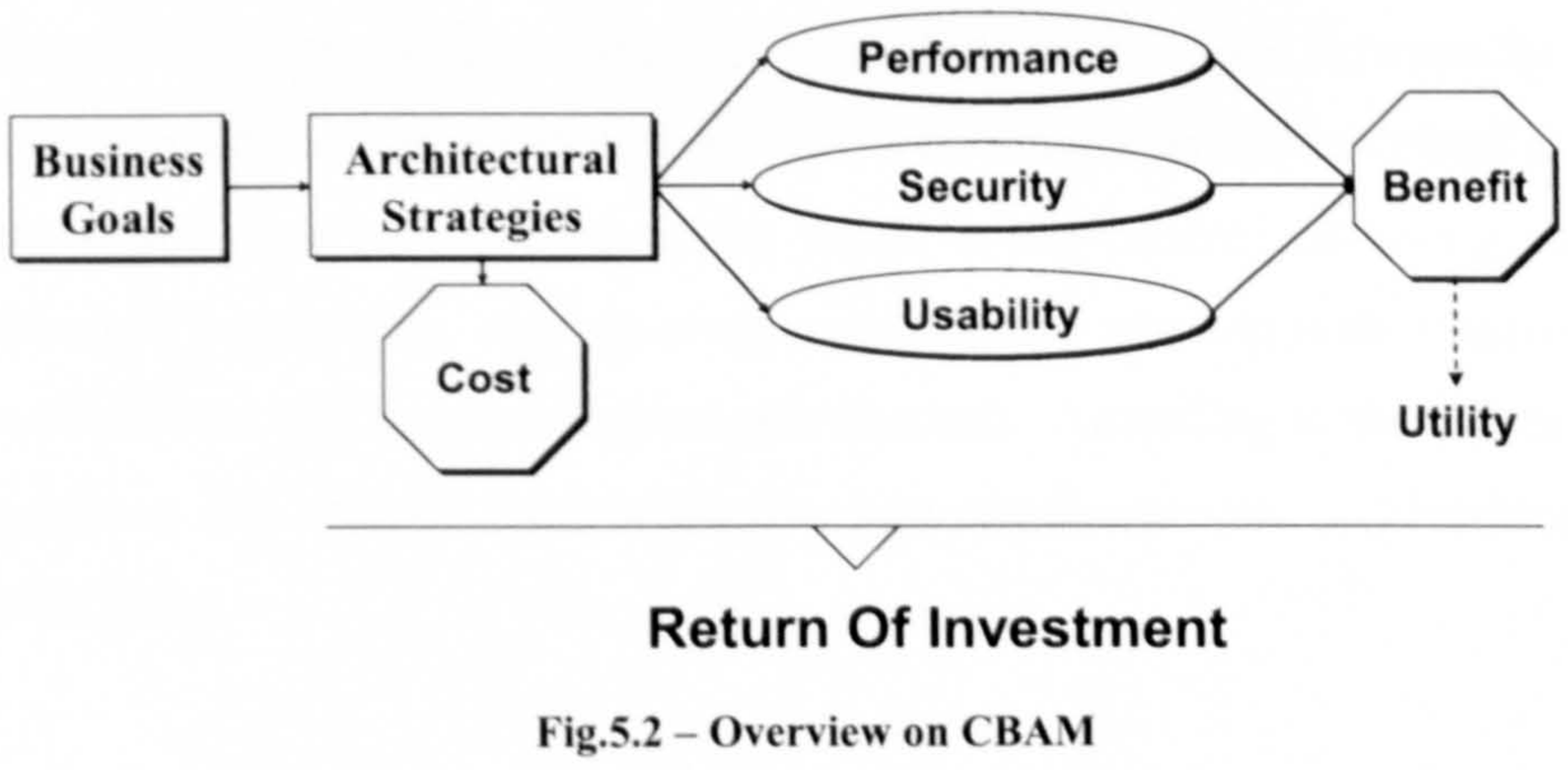

Fig.5.2 – Overview on CBAM

Although importance of implementation of each attribute can be added as part of the analysis using weights, CBAM remains a pure numerical comparison of utility of each attribute against cost. Cost benefit analysis is a straight forward process and although the results are compelling analysts should be circumspect, as the numerical representation of all quality attributes may mask the importance of a quality attribute. A

case in which cost benefit analysis has been misapplied taking into account only economical results is discussed in the ALARP section.

5.3.3 Easy Win-Win

Easy win-win is a process for eliciting and negotiating system requirements [93]. The objective is again to engage system stakeholders in understanding each other's viewpoint and collectively agree a set of compatible requirements. In order to achieve this state each stakeholder describes a set of win conditions. Win conditions are statements about the successful operation of the system as it has been envisioned by the stakeholders. Stakeholders examine the win conditions and identify whether they can impact other stakeholders negatively. A condition which can impact stakeholders negatively is called a lose condition. Identification of win and lose conditions allow the stakeholders to negotiate through the win-win process an alternative set of requirements, ultimately resulting in conditions that would be characterised as win conditions by all.

The method identifies trade-off points. Unlike ATAM win-win points are statements of conflict between requirements and do not suggest how the architecture can influence the achievement of the requirements. Trade-offs are not explicitly analysed and recorded but the method focuses on describing the result of the negotiations between the involved stakeholders over an acknowledged conflict. Win conditions are described in terms of importance and ease of implementation, which is the product of voting among the participants of the analysis. An important aspect of easy win-win is the classification of the win conditions and their subsequent prioritisation. According to the values assigned to importance and ease of implementation, win conditions can be classified in four categories [93]:

- *Low hanging fruits (important, easy)*: Win conditions with a high business importance, which seem to be feasible.
- *Important with hurdles (important, difficult)*: Crucial win conditions difficult to implement.
- *Maybe later (unimportant, easy)*: Low-priority win conditions that may be considered later because of their low difficulty of realisation.

- *Forget it (unimportant, difficult)*: Unimportant win conditions that are difficult to achieve.

The importance and ease of implementation are two criteria that motivate the system stakeholders to implement some win conditions over others. For example, an unimportant and difficult is considered to be something that will not be implemented on the grounds that is difficult to implement, and that the perceived utility of the achievement of the condition is low in comparison to the other conditions. The prominence of this feature of easy win-win lies in the fact that the four categories constitute implicit arguments based on which the decision about the implementation order is justified.

5.3.4 Multi Criteria Decision Analysis

Multi Criteria Decision Analysis is a process that allows making decisions between candidate alternatives in an environment with multiple competing objectives. One such model is the Analytic Hierarchy Process (AHP).

5.3.4.1 The Analytic Hierarchy Process

The Analytical Hierarchy Process (AHP) is a systematic method for making decisions, between a number of possible options with respect to multiple objectives [94]. Fig.5.3 illustrates an AHP the main elements in the AHP hierarchy.

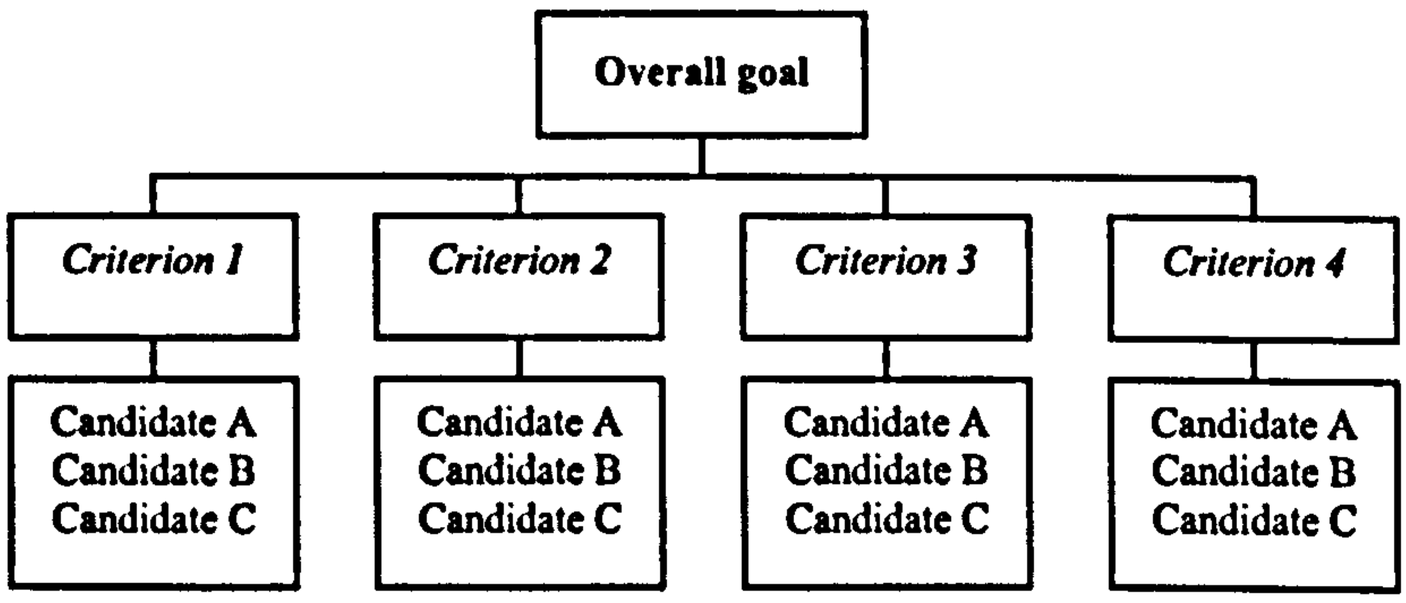

Fig.5.3 – AHP Hierarchy

Initially the participants state the overall goal of the decision making process, which usually would be the selection of the optimum candidate decision. In a process

resembling the creation of utility trees in ATAM, the overall goal is decomposed into a number of criteria (also called the objectives that the alternative needs to achieve) that need to be satisfied, and against which the candidate decisions will be evaluated. After the criteria have been defined the participants make pair-wise comparisons between goals populating a matrix with values indicating the relative importance of the x, y objectives (i.e. 1 = believe that they are of equal importance, 9 = very strongly believe that X is more important than Y). Calculating the eigenvector of the matrix of the criteria, results in a vector that indicates the overall importance of each goal.

Following the calculation of weights for each of the objectives the participants make use of the same process to compare the 'goodness' of the options. For each objective a table is created with pair-wise comparison between the options answering the generic question "which option from the two you think has achieved the objective better". Alternatively scale values can be used. The process will result in a set of tables, the eigenvector of which shows the relative 'goodness' of each alternative with respect to each objective. Combining the table using matrix algebra results in a 1xN (where N the number of alternatives) matrix, showing the overall score of each alternative. Among AHP's strengths is the fact that the participants can made subjective judgements regarding the importance of the objectives, as well as the achievements of the objectives from the options.

5.4 Using AHP for Trading-off goals in the Context of Dependability Cases

This is an exercise showing potential use of AHP as part of the construction of a dependability case. The exercise was created during meetings between the members[vi] of the partnership, as part of which this research took place. The purpose of the exercise was to examine the suitability of AHP to be used in order make justified trade-offs between system goals in the context of dependability case development. In the exercise AHP was used to select the most optimum design for a hypothetical Future Transport Aircraft (FTA). The new transport requirements as described in [95] served as inspiration for the definition of the requirements and the general context for the FTA exercise. This is a hypothetical example with the sole purpose the evaluation of the

[vi] QinetiQ, BAE Systems and Rolls-Royce plc.

concepts (such as application of a utility vector to represent dependability) of the AHP methodology. In accordance to the description of AHP the process involves three main phases:

- Identification of overall goal and criteria
- Prioritisation of criteria and calculation of weights
- Assessment of the candidate decisions

Although it was endeavoured to provide realistic requirements, their accuracy is not of primary importance to the exercise. The data relating to the aircraft were inspired by real examples, but in certain occasions (i.e. minimum stall speed and flight endurance) conjectures were made. However this does not affect how AHP is used but merely the details of the final result, which is not a concern.

5.4.1 Identification of Criteria & Alternatives

The first step in the AHP process is the identification of the criteria, which will be part of the final hierarchy. Criteria represent the requirements that need to be achieved by the candidate design, and constitute the context in which the candidate (design) alternatives will be assessed. In a (GSN based) dependability case the criteria would be the equivalent to goals, representing the system requirements that will have to be achieved by the system (i.e. the aircraft).

Deviation Dependability Analysis (described in chapter 4) was applied to derive the goals and dependability requirements for the FTA. DDA was not used for a full system analysis but to elicit the high level system goals and requirements by identifying the system concerns. A prerequisite for DDA was the existence of the high level of concept of operations. The Future Transport Aircraft (FTA) is a military cargo aircraft. According to its concept of operation, the FTA can undertake multiple roles according to the purposes of the mission, including strategic, tactical and theatre of operations. The strategic role involves transport of big volumes of cargo and personnel between (possibly distant) bases or command centres. During a tactical mission the aircraft is required to deploy equipment and personnel using temporary airstrips, prepared at the beginning and maintained until the end of the mission. Finally in the theatre of

operations role, the FTA provides support such as surveillance and medical evacuation. Fig.5.4 presents the issues with which the tasks of each FTA role were prompted. The issues cover a range of attributes including performance safety, usability.

- Cargo capacity
- Range
- Speed
- Take off landing runway length
- Autonomous ground handling
- Ground manoeuvrability
- Minimum stall speed
- Flight Durability
- Ease of payload conversion
- Hull loss (Safety)

Fig.5.4 – FTA Issues

Using the identified issues the high level tasks of the FTA for each role were examined and concerns were identified.

Table.5.1 – FTA Overall Concerns

Concern	Task Issue	Target Requirement
Hull loss	Unacceptable loss of life (non-hostile).	10^{-8}
Payload	Inadequate transport capability.	60 t
Range	Inadequate range; requires refuelling for distances greater than the achieved range.	3500 nm
Minimum Stall Flight Speed	Difficulty in deploying paratroopers, and airborne delivered equipment. This also impacts the length of the landing strip.	110 kts
Flight Endurance	The aircraft will have to break mission to refuel	10 hrs
Reliability	Probability of fault that will result in aborting the mission.	10^{-5}
STOL	Minimum length of the required runway for short take of and landing (STOL). Very difficult to create long runways in theatre of operations.	2300 ft

By applying the steps of the DDA process, Table.5.1 was created. The table summarises the task issues, the acknowledged concerns, and their respective requirements elicited from the DDA process.

5.4.2 Calculation of Weights

The next step of AHP involves calculation of weights for each of the identified criteria. The weights show the relative importance among the identified criteria. Calculation of weights entails the creation of a matrix, which includes pair wise comparisons of the criteria (Fig.5.5 a). The values represent the relative importance of the criteria (i.e. 1 = believe that they are of equal importance, 9 = very strongly believe that X is more important than Y). A tool implemented by the author in Java, prompted the exercise participants for pair-wise evaluations between the identified criteria, and calculated the eigenvector of the table. The resultant vector is shown in Fig.5.5 (b), and represents the importance of each criterion according to how the participants evaluated each pair-wise comparison.

	1	2	3	4	5	6	7
1|	1	8	5	3	3	4	3
2|	0.125	1	0.333	0.25	0.33	2	0.25
3|	0.2	3	1	0.5	3	3	0.25
4|	0.33	4	2	1	1	2	0.5
5|	0.33	3	0.33	1	1	2	0.5
6|	0.25	0.5	0.33	0.5	0.5	1	1
7|	0.33	4	4	2	2	1	1

(a)

WEIGHTS:

Safety: 0.35453835
STOL: 0.17924733
MinSpeed: 0.12529664
Range: 0.12261105
Endurance: 0.10109467
Reliability: 0.0673232
Payload: 0.049888786

TOTAL: 1.000000

(b)

Fig.5.5 – AHP Criteria Comparison Table & Weights

As expected, when making the pair-wise comparisons, safety received the highest rating followed by the remaining criteria. It is worth noticing that two of the criteria, namely minimum stall speed and range have little difference in their weights, only becoming evident at the 3rd decimal digit of the weight value (the significance of this observation is discussed in §5.5)

5.4.3 Evaluation of Alternatives

Evaluation of the possible design options followed the calculation of weights. In the FTA exercise the focus was on one design aspect, namely the number and configuration of the aircraft engines. Identification of how the characteristics of each option affect the goals (similar to sensitivity analysis as performed in ATAM) was not in the scope of the

exercise, which focused on the effectiveness of AHP in architectural trade-offs; a more detailed discussion about design rationale can be found tin chapter 6. However during the exercise such activity took place in order to understand how each alternative would affect each criterion. Three options were proposed:

- Option A: Two high thrust engines under the wings.
- Option B: Four engines under the wings.
- Option C: Two engines above the wings.

The characteristics of the three options have different impacts on the operation of the aircraft. At this stage assumptions were made about how the three configurations affect each of the criteria. Again, although the participants had rudimentary knowledge of the principles that govern the behaviour of the aircraft, there were assumptions made; the factual accuracy of which was not of primary importance to the exercise.

By using two engines over the wing, option C achieves more lift and therefore the aircraft has lower minimum stall speed. This allows for shorter a runway as the aircraft can take off sooner and needs less space to brake during landing. Moreover when using temporary strips as runways, the engines are protected from debris suction that may damage the engine; especially during short take-offs when the engine is set to the highest thrust, this can be a very serious risk. In contrast to option C, option A has two engines mounted under the wing, providing less lift. Hence higher thrust engines are required, something that would increase the possibility of debris suction. Moreover higher thrust engines would comparatively contribute to higher overall weight, increasing in the same time fuel consumption, which would compromise flight endurance. Finally, option B is a more conventional configuration; the aircraft has lower (compared to the two other options) thrust four engines mounted under the wing. Although more exposed to debris than option C, the low thrust reduces the risk of debris ingestion in comparison to option A. Option B provides the aircraft with the capability to fly with one or two engines shut down or at idle power (i.e. gliding with a small descent rate) preserving fuel and therefore extending flight endurance.

Next in the AHP process, the alternatives are evaluated against the identified criteria. In order to simplify the example and increase accuracy of the exercise, only three criteria were used. Selection of the three criteria was based on the confidence of the participants, in being able to predict how the characteristics of each alternative would affect the criteria during evaluation. Calculation of weights was repeated for the three selected criteria, the results of which are presented in Table.5.2.

Table.5.2 – Weights for the Criteria Subset

Criterion	Weight
Safety	0.7018
STOL	0.1972
Endurance	0.1009

Table.5.3 shows an overview of the normalised results of the evaluation of the three options. Within brackets is the original score in a scale from 1 to 10 that assigned to each alternative with respect to each criterion.

In the case of this exercise, evaluation was based on the participants assigning the 'goodness' of each alternative, by interpreting the degree of satisfaction of each criterion. For example, the evaluation results show that both options B and C would have the same degree of achievement of endurance and therefore were assigned the same grade. Moreover the table implies that option C performance in STOL is nearly fully satisfied the criterion and therefore was assigned a 9 out of 10.

Table.5.3 – Evaluation and Normalisation of Alternatives

	Option A	Option B	Option C
Safety	0.2857 (6/10)	0.3809 (8/10)	0.333 (7/10)
STOL	0.2727 (6/10)	0.3181 (7/10)	0.409 (9/10)
Endurance	0.25 (4/10)	0.375 (6/10)	0.375 (6/10)

The final result of the process is a 1x3 matrix presenting the overall utility of each of the alternatives. This is produced by combining, using matrix algebra, the matrix that results from Table.5.3 and the matrix containing the weights of each criterion Table.5.2.

The resultant matrix contains the overall (normalised) utility of each alternative with respect to all criteria.

$$UtilityMatrix(U) = AlternativesEvaluationMatrix \times WeightsMatrix \Rightarrow$$

$$U = \begin{matrix} 0.2857 & 0.2727 & 0.25 \\ 0.3809 & 0.3181 & 0.375 \\ 0.333 & 0.409 & 0.375 \end{matrix} \times \begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix} \Rightarrow \begin{matrix} 0.2857 & 0.2727 & 0.25 \\ 0.3809 & 0.3181 & 0.375 \\ 0.333 & 0.409 & 0.375 \end{matrix} \begin{matrix} 0.7018 \\ 0.1972 \\ 0.1009 \end{matrix} \Rightarrow \begin{matrix} 0.27936 \\ 0.36787 \\ 0.35219 \end{matrix} \begin{matrix} (optionA \\ optionB \\ optionC) \end{matrix}$$

Fig.5.6 – Calculation of Utility Vector

Fig.5.6 shows the calculations taking place when producing the utility matrix. Option B (0.36787) is the most optimum option followed closely by option C (0.35219); option A (0.27936) scored the lowest.

5.5 Drawbacks in the Numerical Representation of Dependability

As described in chapter 3, dependability is a generic term which encompasses many heterogeneous attributes. This characteristic of dependability causes problems when attempting to represent or even understand dependability as a single concept. In fact, when referring to dependability, it is often done in the context of a particular viewpoint such as reliability and safety. This section investigates the problems when modelling dependability using a numerical representation, and discusses its application (i.e. AHP exercise) in making justified decisions in the context of a dependability case. After presentation of the problem and completion of the AHP exercise, discussions followed regarding the suitability of the characteristics of AHP as decision making tool in dependability cases. Observations by the participants of the exercise as well as subsequent analysis and evaluation from the author, identified several challenges to be overcome, in order for a trade-off resolution method to be used in dependability case framework.

Use of a trade-off resolution method in the context of a dependability case requires clarity in rationale of justifications, and traceability. As explained in chapter 3, a dependability case presents an argument and communicates assurance regarding the

operation of the system. Arguments in the dependability case evolve and are constructed in the context of the system, and consequently design decisions. In order for the final argument to be compelling all decisions made during system development should be justified. Moreover the rationale on which the decisions were based should be clearly communicated in the case. Clarity of rationale and documentation of justification on which decisions are based are important prerequisites for traceability of the argument. Traceability is an important aspect of a strong and compelling argument contributing to the overall system assurance. Reviews of the argument should be easily performed and reviewers should be able to identify the assumptions and justifications accompanying the inferences from the statement of a goal to the provision of evidence supporting that goal. Moreover, developers need to be able to trace the reasons on which their decisions were based, to identify improvements and to control changes by analysing their impact on the overall dependability of the system.

At the early stages of the exercise, participants expressed a certain degree of confusion when challenged with pair-wise evaluation of the objectives. Specifically, in some instances participants were unable to express a clear preference between the alternatives. The main cause for that was the lack of context in which the relative importance of the criteria was evaluated. Participants were asked to distinguish preference based on the abstract notion of each attribute (e.g. safety versus endurance). However there was no analysis supporting this judgement as to how each attribute contributes to the overall operation of the system. Criteria that were considered far more important than others may be subject to exceptions, something that is not captured by pair wise assessment of the criteria. For example, during a surveillance mission endurance may be more important than safety. Arguing about preference between two objectives requires an assessment on behalf of the stakeholders of the consequences of trading off each objective. However, even when relating the objectives to the intended operation of the system, stakeholders often cannot precisely specify a preference. The reason for this is that the involved stakeholders cannot substantially understand what failing to meet the objective will signify for the envisioned operation of the system, unless the degree of satisfaction of that objective is stated.

Making trade-offs involves comparing the achievement of each of the goals (descending from the attributes) with respect to the design alternatives. Being

heterogeneous and representing fundamentally different concepts, dependability attributes cannot be compared directly with each other by a mere numerical comparison. For instance consider two goals representing requirements elicited from the viewpoint of two different dependability attributes. The goals are described using numerical representation of utility, with measures (assuming a percentage scale) of 65% and 70%. This may not necessarily mean that there is an overall difference of 5% in utility. The consequences of not achieving the goals are fundamentally different and cannot be directly compared. Even factoring weights in the utility of each goal cannot represent with accuracy how the two goals can be valued in terms of their impact on the system's operation. Prasad [10] in her thesis concludes that we can only quantify attributes of dependability individually. Even then, Prasad mentions that some attributes such as safety are not easily quantifiable. Furthermore, she continues by claiming that is infeasible to represent dependability as a single metric with methods such as the Multi Attribute Utility Theory. This is a view also put forward by [96] in which it is stated that *"The main problems perceived in use of multi-attribute utility methods are: difficulty of trading-off very different kinds of attributes; subjectivity of the problem structure and weightings used; and consistency from one decision to another"*. Prasad mentions that a representation of utility as a vector could be potentially used. However a numerical approach also showed difficulties in justifying the meta-decisions made during the assessment such as pair-wise judgements, and evaluation of alternatives.

Application of the trade-off method takes place within an argument based framework such as dependability cases. Hence any decision made needs to be justified. Another part of the exercise was the construction of an argument justifying the alternative that was selected that would incorporate steps of the AHP. This eventually resulted in arguing about the assignment of values to the comparisons. Although the overall direction of the comparison (e.g. A is better than B) was captured, the specific value assigned during a pair-wise comparison is something that is assigned subjectively and cannot be justified that it reflects the stakeholders' interests, resulting in weak arguments. Fench et al suggest, with regard to decision making in general, that *"disagreements among groups are addressed via sensitivity analysis and debate not via some mathematical formula which combines judgements in some democratic way and prescribes a consensus decision"* [96].

Traceability of decisions (irrespective of whether they involve trading off of objectives) is an important aspect of a dependability case supporting evolutionary development. The justification and rationale behind a decision and in particular decisions necessitating trades, constitute vital information for the success of further design reviews, system (and case) maintenance, and integration with other systems in a System of Systems. During these activities system developers need to review how a decision was taken and re-examine whether the contextual information used (such as the design of the system) have changed since the decision. Developers are often interested in how individual elements of the system contribute to the overall 'fitness' of the system for its use. Basing a design decision or trade-off on numerical criteria such as the creation of a utility function (using for example AHP) makes it considerably more difficult to trace the decision to individual elements of the system, something that was also pointed out by Prasad [10].

5.6 The Trade-Off Method (TOM)

The Trade-Off Method (TOM), developed by the author, is inspired from the ALARP principle, aiming to facilitate dynamic reasoning about candidate design options (with respect to the system goals). TOM consists of a number of steps, which allow system stakeholders to evaluate design alternatives and systematically make trade-offs balancing all required system properties.

5.6.1 Objectives

The purpose of the method is to produce an argument module, which reasons about optimum and acceptable selection of an alternative. The resultant argument provides context in which the dependability case evolves. TOM aims to:

- *Enable stakeholders to understand and share their perspectives and rationale for acceptability.* Each stakeholder has a unique perspective on the system's operation with system goals that reflect their interests. This means that when eliciting goals related to them, stakeholders cannot necessarily understand the implications the achievement of the goals will have on other stakeholders. Moreover, it is essential that during trade-off discussions, all the stakeholders

share their rationale and understanding of the consequences of compromising all goals on the overall acceptability of the system. There is little usefulness in developing a system that may satisfy the goals of a particular stakeholder but is not fit for purpose with regard to another. It is not the purpose of TOM to provide a 'recipe' for solving disagreements between stakeholders. The final decision of the process must be justified.

- *Systematically examine and understand the trade-offs involved with each alternative and provide feedback to the evolution of the design.* Selection of each alternative involves relative benefits and compromises with respect to other alternatives. Stakeholders identify the motivation for selecting each alternative and reason whether the benefits of an alternative can balance the compromises. Moreover, the strengths and weaknesses of each alternative can be fed back to the design process, which under a particular focus (provided by TOM) may improve the proposed alternatives.

- *Document reasoning, contributing to the traceability of the dependability case.* Evolution of the system entails design decisions, in the context of which the dependability case evolves. The trade-off argument constitutes context to the 'main' dependability argument capturing the justification and rationale, on the basis of which the alternative was selected. Documentation of the reasoning leading to the selection of the alternative provides an explicit reference to contextual information which can be easily reviewed and modified.

5.6.2 Overview of the Method

Fig.5.7 presents the stages of the trade-off method. Initially, the system goals are identified from the dependability case along with their targets. Following the links in the dependability case provided by the metamodel (DCM), participants identify from the products of the DDA process, the rationale for the goals and their target. The aim of this step is to define and justify 'how much' compromise of the goals can be tolerated, still resulting in an overall acceptable system.

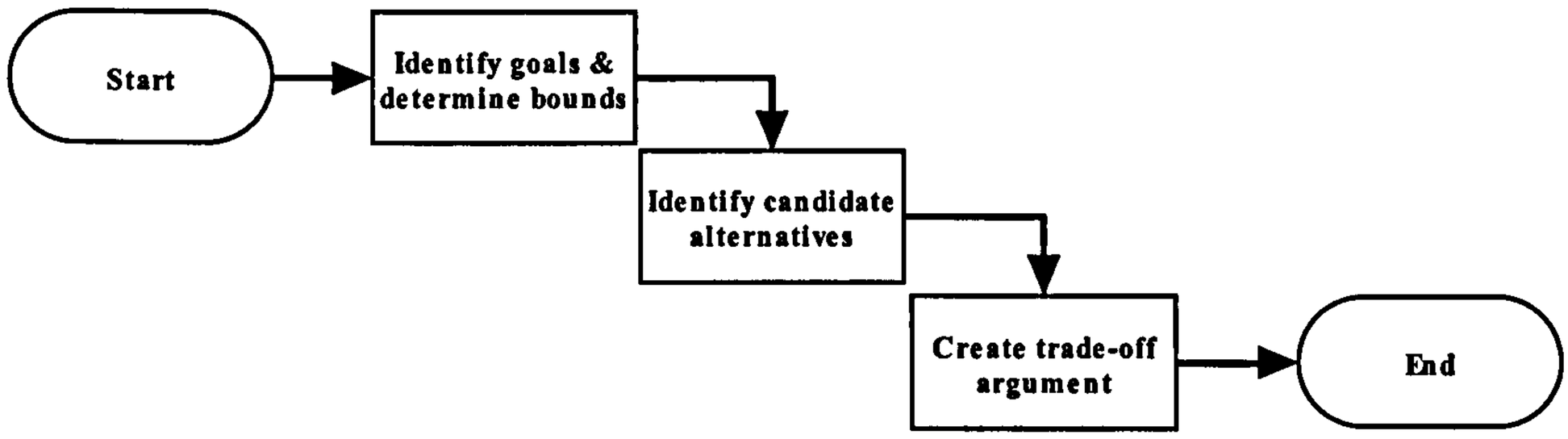

Fig.5.7 – Overall Processes of TOM

Next, participants identify the candidate alternatives that were created during the design process and evaluate the alternatives with respect to the goals. Finally examining the 'goodness' of the candidate alternatives participants identify the involved trade-offs and reason about the selection of the most suitable alternative.

5.6.3 Fundamental Concepts of TOM

The trade-off method introduces a number of concepts used in order to establish the trade-off argument:

- Acceptability of requirements
- Flexibility of requirements
- Willingness to trade-off

These concepts are discussed in the following sub-sections.

5.6.3.1 Acceptability of Requirements

Acceptability of the degree of satisfaction of requirements depends on the context in which the system will operate, as well as on how the stakeholders envision the operation of the system. Achieving stated requirements is often not a 'black or white' situation, but they can also be partially met. The degree of satisfaction indicates the extent to which the requirement has been met; however, it is meaningless if stated out of the context of operation of the system. For example stating a degree of achievement of 90% for a performance requirement of 1Mbit bandwidth is not helpful; unless the stakeholders assess the implications of the 'partial' achievement of the original

requirement to the (envisioned) operation of the system. Implications may mean that the interests of the stakeholder(s) that defined the requirement are compromised, or they can be negligible. Acceptability of requirements helps providing a meaningful interpretation of the degree of achievement of a dependability requirement. Trade-offs especially in large systems are inevitable. In order to be able to make trades stakeholders should avoid looking at the requirements as single targets, and instead be prepared to discuss the acceptability of a broader range of (degrees of) satisfaction. This will result in a space in which a not fully met requirement may still be admissible from the stakeholders.

One of the most compelling examples in which partially met requirements may be admissible is safety. It is common consensus that a safety critical system can never be completely safe. For example the stereotype that the safest airplane is the one that will not fly reflects that belief. Hence the assurance for a safety critical system is based on reasoning that a system is *acceptably* safe for its particular use in its operational context. A well-established approach for defining the acceptability of risks is the As Low As Reasonably Practicable (ALARP) principle [44]. The risk associated with the operation of a system is defined in terms of the frequency and severity of the consequences of the identified system hazards. Fig.5.8 shows an adapted example of the categories for classification of risk used in safety standards such as [46].

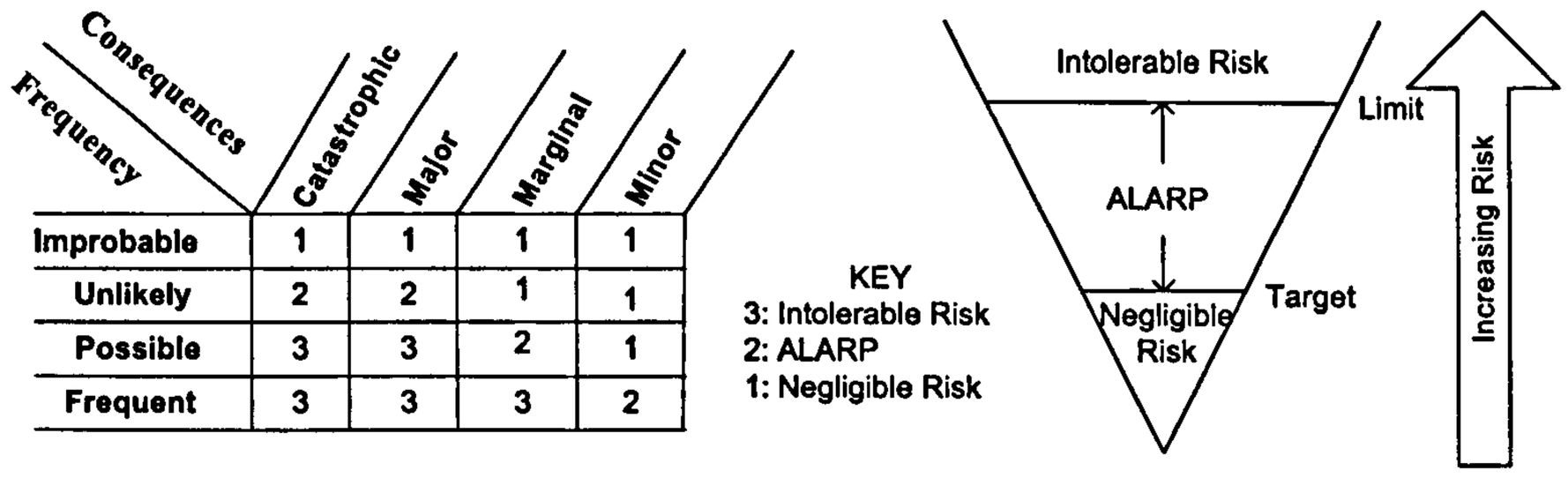

Fig.5.8 – Example of Risk Classification and ALARP

According to the ALARP framework, risk can be classified in three major categories: Negligible, Intolerable, and within the ALARP region. The safety 'target' is to achieve risks that are considered negligible. Additionally, a limit on acceptable risk is defined. Risks exceeding the defined limit are considered intolerable (i.e. unacceptable). The region between these two values (the target and limit) defines the risks that can be

considered to be tolerable if they can be argued to be As Low As Reasonably Practicable (ALARP). A risk is considered to be ALARP if costs associated with further risk reduction can be shown to be disproportionate to the risk reduction that would be achieved.

Defining acceptability criteria for safety is common practice. The requirement for acceptable safety can be flexible depending on the cost required to achieve it. The UK's Health and Safety Executive (HSE) gives examples for targets and limits with regard to acceptable tolerable and intolerable risks to the general public and individuals in various work environments. According to the UK HSE [44], a risk of 1 death in 1,000,000 per year broadly defines the boundary between negligible and tolerable risk. HSE suggest that the boundary between tolerable and intolerable risk should be 1 in 10,000 per year for the public, and 1 in 1000 per year for workers. When considering the limit for the risk limits we take into account the residual risks of every day life, as well as the benefit of the activity to the society.

Security is another example of dependability attribute for which flexibility in requirements is practiced. Security has a unique characteristic that possible threats to the system are caused by malice. The Common Criteria for Information Technology Security Evaluation [61] provides a framework that defines the security evaluation process for a system, with respect to a target of evaluation. The Common Criteria framework defines six security assurance levels, each of which describes a set of criteria to be satisfied and the processes to be followed during the development of the system.

There are other examples of attributes in which the definition of acceptability is important in understanding the benefit of a design decision. For example, the performance of a system (i.e. delivery of functionality in a timely manner) is important for real time systems, where temporal deadlines must be met. In practice, occasional losses can often be tolerated, either because the consequences of the loss are negligible, or because the system is able to react before the next deadline without serious consequences. It has been suggested that the tolerability of missing the deadlines can be specified by 'constraint values' that define the limit for missed deadlines [97]. For example, consider a hypothetical telemedicine system providing real time digital video. The system should decode the video stream at a desired 23 frames per second (fps). A

missed deadline is a frame that has not been decoded before the next one in sequence arrives for decoding. If few deadlines are missed the video stream would not degrade significantly, but if too many are missed this can endanger the procedure. In this case the target is 0 missed deadlines and the limit is no more than 3 missed deadlines in a row, at which the video will still be acceptably clear although with degraded quality. Overall, stakeholders' acceptance may vary according to operational context of the system. In this thesis the term acceptability is defined as follows:

***Acceptability** is the attribute of a goal that captures the interpretation of the degree of satisfaction of a goal, with respect to the impact it will have on the envisioned operation of the system compared with full achievement of the goal.*

5.6.3.2 Flexibility in Goal Based Requirements

In the aforementioned examples, defining the acceptability of a system requires description of the required 'target' and 'limit' values, defining the region in which divergence from the target value is tolerable. When the limit of the acceptability criteria has not been met, the system is considered unacceptable and action must be taken to correct this. However, in cases where a target value for an attribute is not met the system can still be considered acceptable *if justified.* This requires arguing that the consequences of 'increasing' the attribute towards the target are disproportionate to the benefit achieved. Defining the acceptability region also needs to be justified. The acceptability region is only meaningful when considered in the operational context of the system.

*Targets and limits define the **bounds** of a goal that represent a space (of acceptable solutions) in which degree of satisfaction of a goal can be traded.*

Definition of bounds achieves separation of concerns when reasoning about requirements. Goals describe the *core intent* of the requirement (e.g. system is acceptably safe), whereas bounds represent the acceptability criteria of the requirement. As an extension to GSN, this thesis introduces the concept of bounds of acceptability of a goal, comprising of a target and limit values. These two criteria define the bounds

within which a requirement may be considered acceptable provided that the levels achieved for the goal in question are as high as reasonably practicable.

With respect to GSN, bounds constitute contextual information and they are related to a goal with *in context of* associations. Hence a goal can be stated *in context of* a target and limit (i.e. bounds). In terms of representation in GSN, bounds are denoted by GSN context elements, with a solid bar along the top or bottom edge of the context, indicating the target and limit respectively.

Fig.5.9 presents the security requirement goals associated with part of an example dependability case for GIS (Geographical Information System) software. The extract focuses on security. The top-level goal for security (*'SecTop'*), refers to the architecture ensuring adequate security and it is stated in the context of the definition of security (*'SecDefn'*). In order to further decompose the top-level goal, we follow strategy *'ArgScen'* to argue over the identified system scenarios that refer to security. Among the identified scenarios are scenarios describing the response of the system in the event of someone trying to login to the system (authorisation), and the scenario of the system detecting intrusion to the system.

Eliciting the requirements based on the scenarios, results in identifying as goals for security *'AccSec_Auth'* and *'AccSec_Detect'*. These goals state that the system should provide acceptable means for authorisation and that the system adopts acceptable means for detecting system intrusions. The security goal *'AccSec_Auth'* is stated in the context of acceptability criteria *'TargSec_Auth'* and *'LimSec_Auth'* which define the target and limit values of acceptability, and in the context of justification (*'Auth_BoundsJ'*), enabling references to the supporting rationale for the acceptability criteria as defined.

Since the security goal is stated in the context of a target and limit, changing the acceptability criteria or the justification for them will not necessarily affect the 'core' security goal. Definition of bounds can isolate reasoning about requirements from reasoning about acceptability of requirements and explicitly define a space in which design alternatives can constitute an acceptable solution.

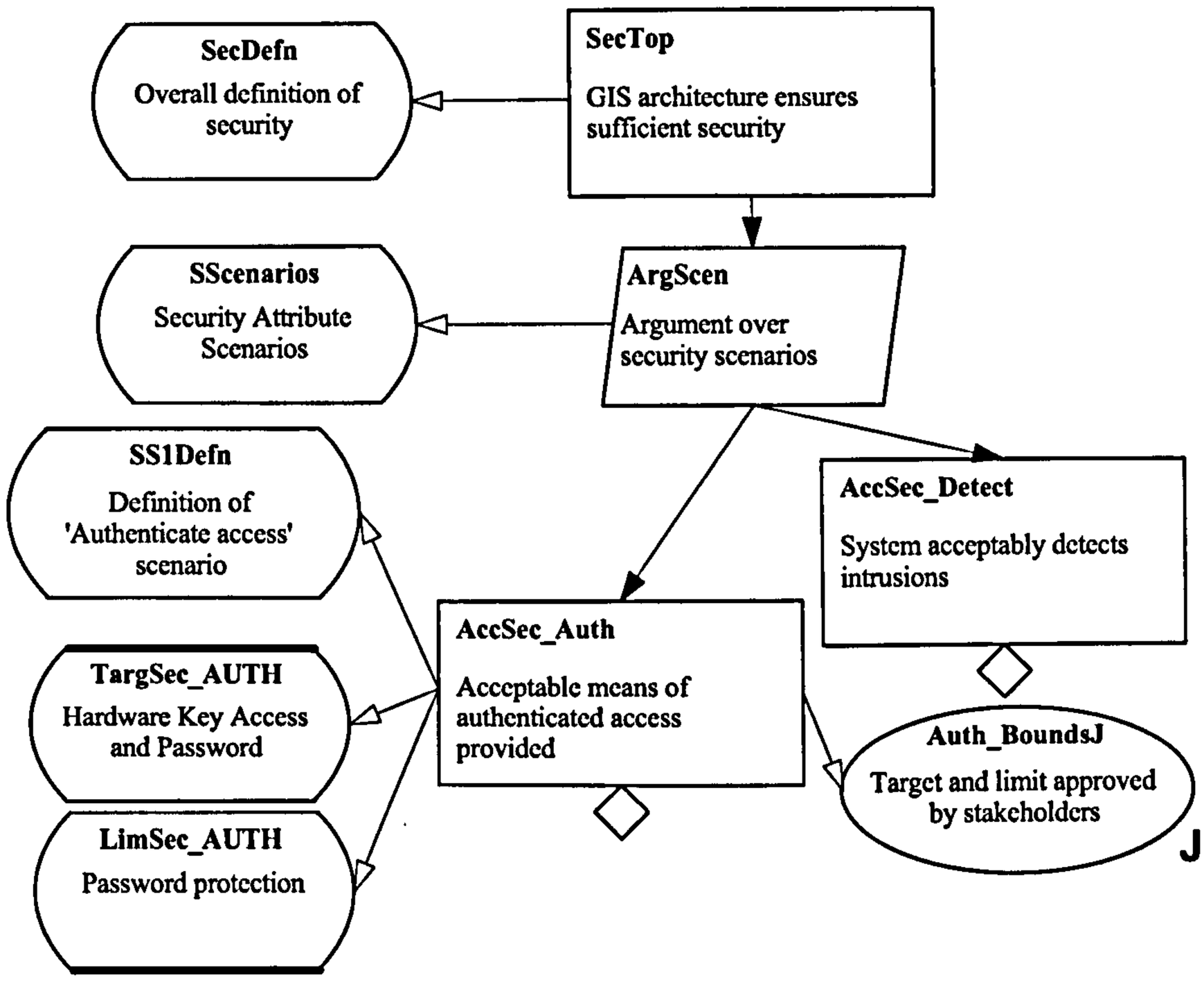

Fig.5.9 – Example of Flexible Security Requirements in GSN

5.6.3.3 Willingness to Trade-Off

Trading-off design goals requires information about how the degree of achievement of the goal is affected by each alternative, the acceptability of the achievement, and evaluation of the importance of each goal.

In ALARP, after the stakeholders have defined the target and limit values of risk, trades between safety and cost need to demonstrate that the cost required to reduce the risk levels of the system is disproportionate to the actual risk reduction achieved. In practice, the tolerable (ALARP) region is often split in two equal sub-regions. Hence when a design is within the ALARP region, it can be identified as being closer to the limit or being closer to the target. This distinction is made in order to help stakeholders to argue about disproportional benefit. For example, it is easier to justify that a risk reduction would be beneficial when closer to the limit value rather than when closer to the target. Extending the ALARP principle for multiple attributes, when two dependability objectives A and B are in conflict, not fully achieving A, may be tolerated if the benefit from *not* compromising B, is greater than the benefit gained from fully

achieving the original requirement of A. In order to implement this principle, willingness to trade-off a goal was introduced in TOM to facilitate trades.

***Willingness** of the stakeholders to trade a (dependability) goal represents the ease with which the stakeholders are prepared to trade the degree of satisfaction of the goal. Willingness is a means of expressing acceptability of a decision alternative with respect to a goal.*

The rationale behind willingness to trade-off is that stakeholders, similar to ALARP, will be more willing to trade a goal if that goal is closer to the target or if that goal is not very important. Accordingly, the minimum required benefit in order to trade-off a goal will vary based on the degree of achievement and the importance of a goal. As discussed, acceptability interprets the degree of satisfaction of a goal. Willingness is specified in relation to the degree of achievement of the required goals by a particular option. Willingness of a goal should not be defined a priori (as in the case of the a priori comparison of importance between goals in AHP), but needs to take place as part of the assessment of an alternative with respect to a goal.

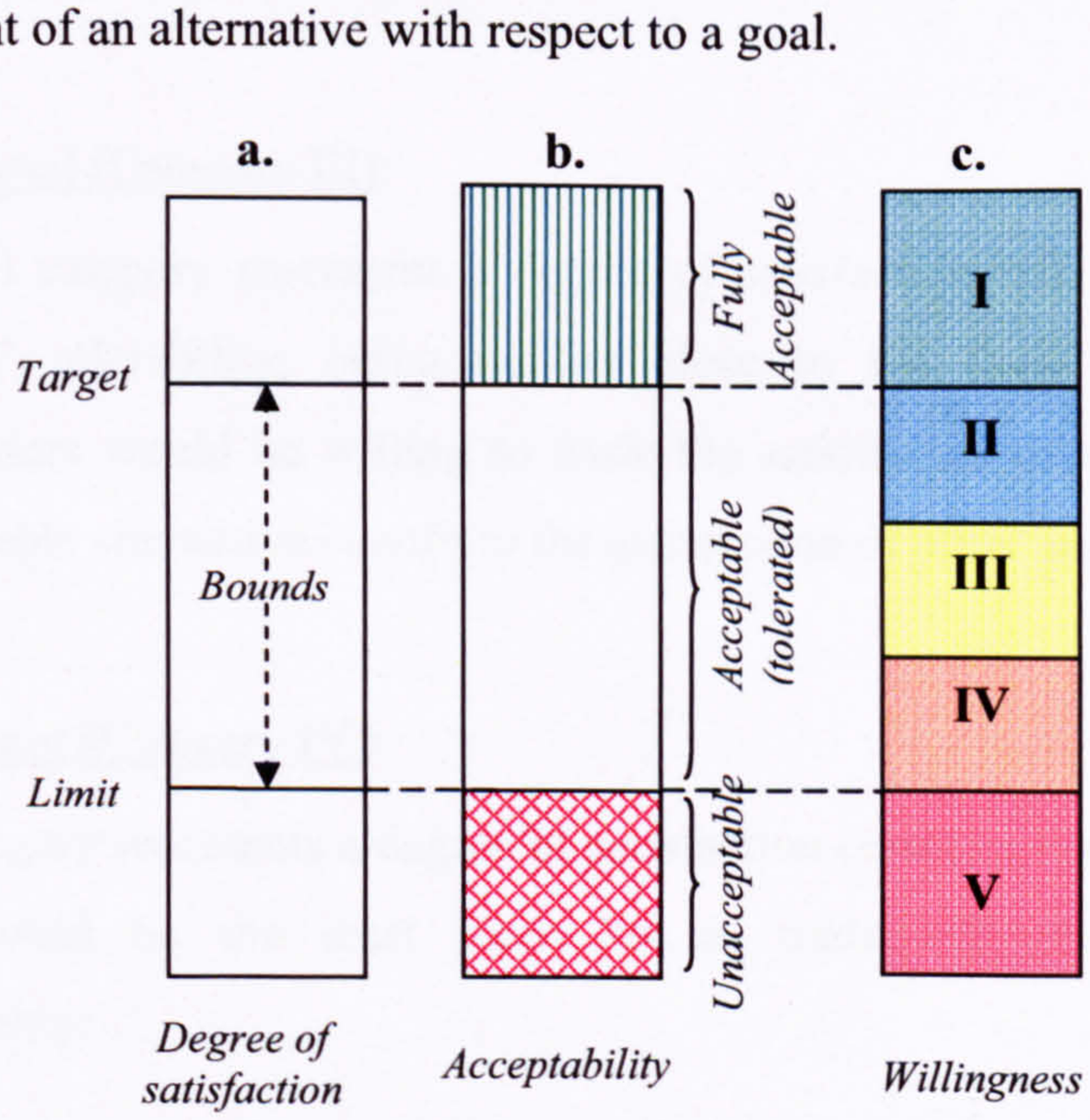

Fig.5.10 – Relating Degree of Satisfaction, Acceptability and Willingness

Fig.5.10 shows the relation between degree of satisfaction, goal acceptability and willingness to trade-off a goal. The left bar represents the range of values for the degree of satisfaction of a goal. However, the values alone are not meaningful; the middle bar

(b) captures the acceptability to the stakeholders, interpreting the various degrees of satisfaction. Finally the right bar (c) illustrates the categories of willingness, expressing the levels of acceptability of a goal. There are five categories of willingness defined to represent the acceptability of the degree of satisfaction of a goal:

1. Unconstrained (Category I):

This category represents a degree of satisfaction that exceeds the defined target. These goals would be the first to consider trading, given that the requirement of the stakeholders represented by the goal has not been compromised in any way. Stakeholders would not need significant gains (in terms of the benefit to the satisfaction of other goals) in order to be willing to trade the goal.

2. Probable (Category II):

This category represents a degree of satisfaction falling short of the target value. These goals would be probable candidates for trade-offs, if some benefit (to the satisfaction of other goals) can be gained.

3. Potential (Category III):

The third category represents a degree of satisfaction falling in the region of 'medium' tolerability, being neither close to the target nor limit values. Stakeholders would be willing to trade the satisfaction of these goals given a considerable alternative benefit to the satisfaction of other goals.

4. Hesitant (Category IV):

This category represents a degree of satisfaction closer to the limit value. These goals would be the least likely to be traded-off given their marginal acceptability.

5. Ineligible (Category V):

This category represents a degree of satisfaction that is considered unacceptable (i.e. falling short of the limit value). An alternative that results in a degree of satisfaction in this category will be rejected and cannot be traded-off.

The degree of satisfaction of the goal is not the single consideration when assigning willingness to goals with respect to an alternative; another consideration is the relative importance between the goals. For example, ALARP assumes that importance between safety and cost should be biased in favour of safety. This is the reason that the system stakeholders are required to demonstrate that the benefit from compromising safety (and therefore cost saving) is *disproportional* to the compromise itself. Willingness to trade-off incorporates importance of the goal. Importance of the goal is not stated explicitly using a metric, but is implicitly included in the assignment of willingness.

Stakeholders will be more willing to trade a goal in favour of a more goal. By describing willingness to trade-off stakeholders implicitly include assessment of how important the goal is to them. For example, if a performance and safety goal were described with the same level of willingness (with respect to an alternative), this would mean that stakeholders after considering the acceptability of the degree of satisfaction of the goals (step of the TOM process), they are equally willing to make a trade-off. A major difference with the example using the AHP method is that stakeholders only evaluate the specific instance of the goal in relation to the candidate alternative.

Altering the distribution of the willingness categories within the acceptable space of the goal, stakeholders can compensate for the goal's importance. An example of an occasion in which this could happen is when the operational context of the system changes making the goal more important to the system stakeholders. Fig.5.11 includes instances of goals (bars d and e), which show two additional examples of assignment of willingness. In the examples, the spaces, which represent lower willingness to trade the goal, cover most of the range with the bounds of acceptability of the goal.

This is illustrated best in the goal captured in bar e. The importance of the goal makes the stakeholders hesitant to trade it, leaving only a small window in which the goal can be traded without some significant benefit (yellow and aqua spaces), and an even smaller window in which the goal can be traded with some benefit (aqua space only).

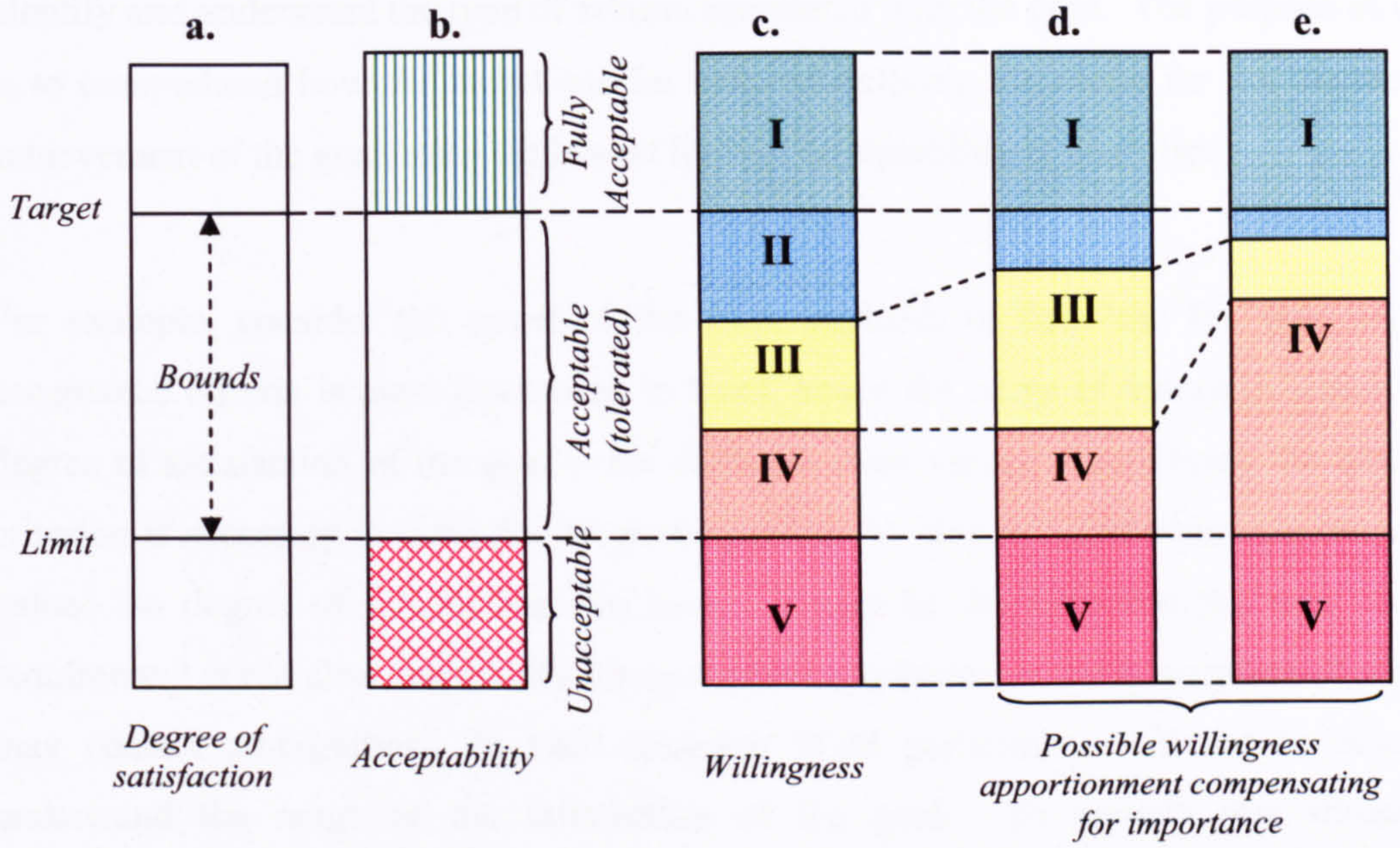

Fig.5.11 – Compensating Willingness

5.7 Method Walkthrough

This section presents in detail the stages of the methodology, illustrating its application on the FTA example.

5.7.1 Determination of Bounds

The first stage of the trade-off method is responsible for eliciting and capturing the degree of satisfaction of the goals as well as the acceptability, described in terms of willingness, which corresponds to the different levels of the degree of satisfaction.

Fig.5.12 illustrates in detail the process of this stage. Initially, the stakeholders identify the goals that the system needs to satisfy, which are defined during application of the Dependability Deviation Analysis (DDA). At this stage the goals elicited during DDA are stated in the context of the target (one of the two elements of bounds; the other is limit). The target represents the actual requirement accompanying the intent represented by the goal, as described in chapter §5.6.3.2. During application of TOM the tolerability of the stakeholders with regard to the requirements is examined, resulting in definition of the limit (complementing the already defined targets). Following identification of the goal, it is important for the participants in the method to

identify and understand the type of criteria associated with the goal. The purpose of this is to comprehend how the target can be reduced defining the range for the degree of achievement of the goal, until the lowest limit of acceptability is identified.

For example, consider the target of the FTA payload; in this case the type of the acceptance criteria is mass (measured in tons), hence the range of values in which the degree of satisfaction of the goal is the different mass values. Identifying the type of criterion is necessary in order for the participants to be able to comprehend the range of values the degree of satisfaction can have. It may be the case that the type of the requirement is not clear, especially for qualitative requirements the description of which may contain ambiguities. In such occasion TOM participants will not be able to understand the range of the satisfaction of the goal. To remedy this situation, participants should refer to the 'specification of dependability profile' DDA stage, in order to understand the rationale based on which the goal and the target were elicited. Payload constitutes a quantitative criterion; however requirements can also be described using qualitative criteria. In the case of the GIS software example the security goal was stated in the context of qualitative criteria. The intent of the goal was the provision of 'adequate means of authentication'. The type of the criterion is the functionality required for authentication. Reducing the degree of achievement to identify the acceptable limit, the degree of satisfaction involved 'less' authentication functionality, thus making the system less secure. In this example target and limit were not quantitative but they were qualitative. Forcing the criteria to be expressed quantitatively is not always the best approach. A possible quantitative representation of this goal could be a statistical metric showing the probability of malicious actors being authenticated successfully. However such a metric may not always be accurate or even possible to define. Specifically for security there are practical problems introducing a metric; one of them is that the statistical sample may not be suitably large to make conclusions since the actual security attack is unknown until detected.

Bounds can be described in qualitative terms by the stakeholders, provided that they understand and explain how the target can be reduced, defining in this way a region of the degree of satisfaction of the goal. In this example the combination of functionality that allows hardware key authentication and password authentication is considered to be more effective than password authentication, which is also the limit of the goal.

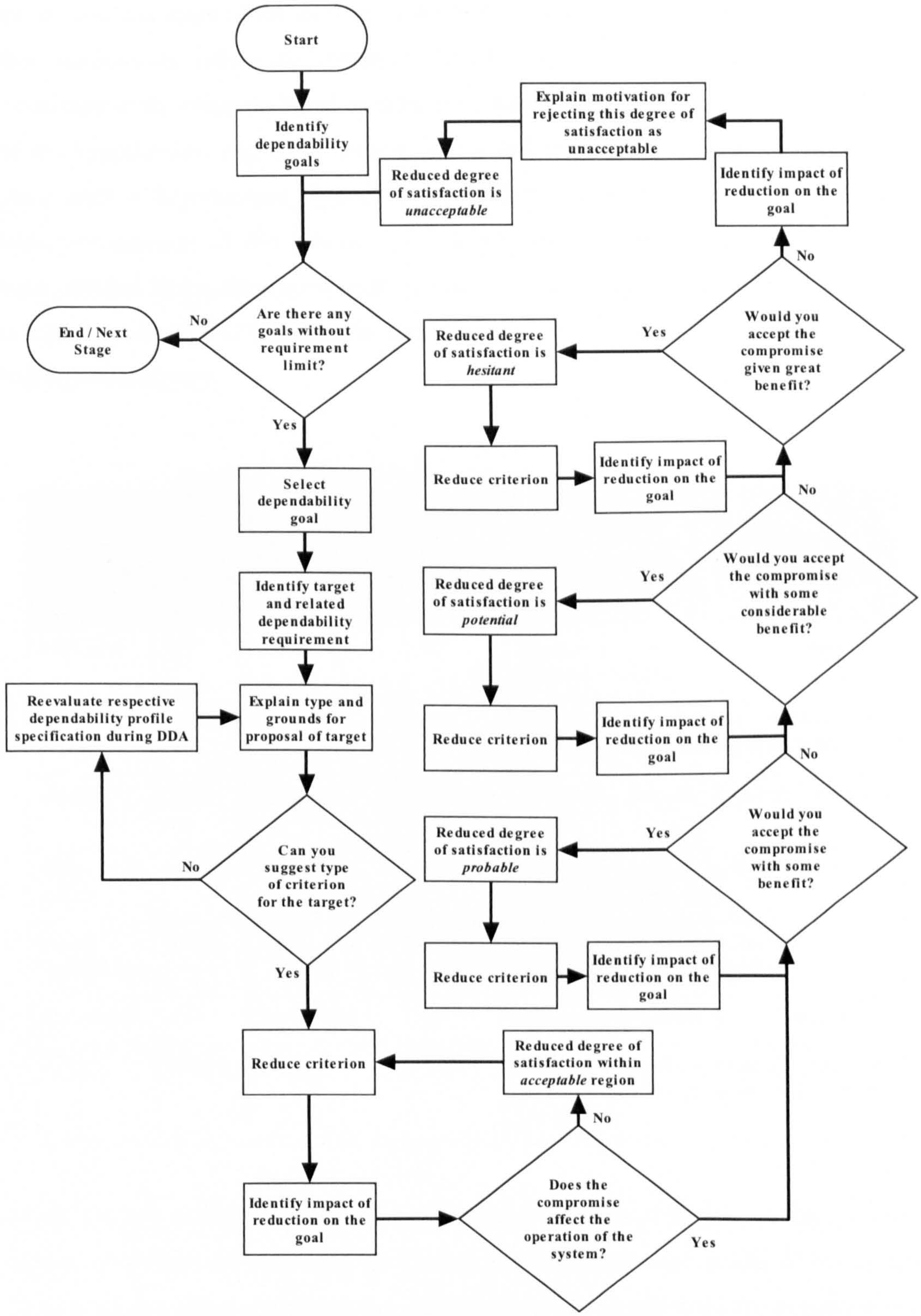

Fig.5.12 – Determination of Bounds Process

Table.5.4 shows the FTA concerns after the completion of this step of the trade-off methodology. Column c documents the type of the criterion that was identified during requirement elicitation shown in column b. Subsequent to identification of the type of

the criteria this stage of the method involves elicitation of the goal's limit. To achieve this, participants reduce the criterion hypothesising a new degree of satisfaction. According to the rationale based on which the target value was elicited the acceptability of the hypothesised degree of satisfaction is described. This iterative process takes place until a hypothesised degree of satisfaction is described *not* acceptable. An important outcome of this process is the justification of the limit (target has already been justified during the requirement elicitation). Justification of the limit is elicited by combining reduction of the criterion and allocation of the new degree of satisfaction to a willingness category.

Table.5.4 – Bounds & Limits of the Identified FTA Concerns

a.	b.	c.	d.	e.
Concern	**Target**	**Type of Criterion**	**Limit**	**Limit Justification**
Hull loss	10^{-8}	Probability	10^{-7}	Maximum probability of losses of previous systems.
Payload	60 t	Mass	50 t	Weight of armoured vehicle and equipment required for start of operations.
Range	3500 nm	Distance	2800 nm	Minimum distance between refuelling points.
Min. Stall Speed	110 kts	Speed	140 kts	Maximum speed for deploying equipment without being damaged.
Flight Endurance	10 hrs	Time	8 hrs	Average time of most operations which the FTA will have to support.
Reliability	10^{-5}	Probability	10^{-4}	Minimum probability to abort mission
STOL	2300 ft	Distance	3000 ft	This is the maximum length of runway the engineering unit can construct on time for the mission

Allocation of a willingness category involves identification of the impact that a reduced degree of satisfaction will have on the intended operation and utility of the system. Participants are asked to evaluate the compensation that would make the compromise of the degree of satisfaction tolerable. This question is put forward in a step-by-step approach, in which the participants evaluate whether the acceptability of a degree of satisfaction belongs to one of the prescribed willingness categories.

Taking a step-by-step approach is more preferable as, in this way, the participants consider the entire range of the degree of satisfaction of a goal.

Consider the payload requirement for the FTA. The target was defined to be 60 tons, with the rationale for choosing this value being that this was the maximum weight of a typical armoured transport vehicle with related equipment. The impact of gradually reducing the payload capacity is a reduction in the FTA's capability to transfer the vehicle and all of its equipment. Hence, only the necessary equipment for the vehicle's immediate operation could be transferred. This may be a reduced capability that is still considered tolerable. Further reduction would result in the FTA being able to transport only the vehicle without any equipment. Continuing to reduce the payload capacity will ultimately result in a FTA that cannot transport an armoured vehicle. The payload capacity at which this occurs was considered to be the the limit for the payload goal. Table.5.4 shows the limits and justifications for all requirements in the FTA example, elicited by applying the process.

5.7.2 Identification of Alternatives

This is a relatively straightforward stage of the trade-off method. The purpose of this stage is for the stakeholders to identify the alternatives and record the acceptability of each alternative with respect to the goals.

Initially the stakeholders identify the candidate alternatives. Construction of the alternatives is not within the scope of the trade-off method, but the product of design evolution methods discussed in chapter 6. After identifying the degree of satisfaction of each goal with each alternative, a willingness category is assigned to each goal-alternative pair. Categories of willingness are assigned based on the correspondence between degree of satisfaction and willingness identified during the previous stage. Evaluation of the alternatives is captures in a tabular form, the trade-off table, illustrated in Table.5.5. Each of the shaded cells constitutes a result of evaluating an alternative with respect to a goal. By browsing through the cells, stakeholders can review the acceptability of each goal, identifying whether an improvement can be made.

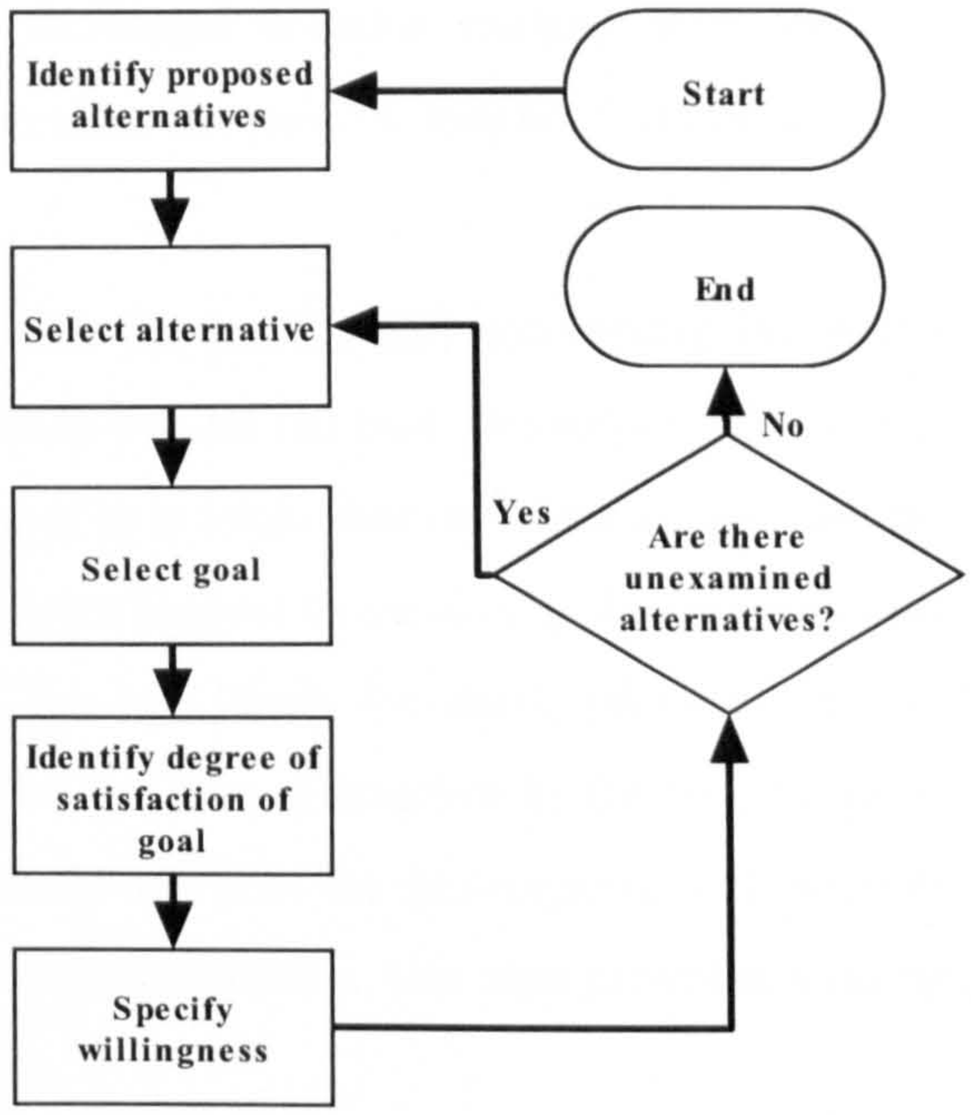

Fig.5.13 – Identification of Alternatives Process

Table.5.5 –Trade-off Table Format

Attributes / Alternatives	Goal 1	Goal 2	Goal 3
Option A			
Option B			
Option C			

Moving vertically in the table the participants of the method can identify the compromise or benefit from switching between candidate options. Moving horizontally the participants identify how an option affects the goals that need to be achieved. Combining these two actions it is possible to identify which goal was the motivation to select another alternative and what is the impact of this improvement. If the improvement involves reducing the acceptability of another goal, then a potential trade-off occurs.

5.7.3 Construction of the Trade-off Argument

Examining an option can lead to opportunities of improving a goal that has not met the target criterion (given that some other option improves the acceptability of that goal). This can lead to arguments in favour of adopting the option that improves that goal.

This stage of TOM facilitates decision making after the consequent trade-offs are evaluated. Fig.5.14 presents the process followed in this stage of TOM.

The process begins with the participants examining the populated trade-off table to identify what they perceive to be the best alternative. Characterisation of an alternative as best can be thought of as a token that indicates at any time the alternative considered as the most preferred with respect to satisfying the dependability goals. The better the alternative is overall, the less likely the participants will prefer another. Selection of what is perceived to be the best alternative at the beginning of the stage, reduces the number of potential trade-offs that the participants will evaluate. Instead of randomly selecting an alternative for evaluation, this step provides a starting point to this stage of TOM.

There are no specific rules for the selection of the best alternative. As a 'rule of thumb' at the beginning of this stage the alternative that has the fewest goals in the lower willingness categories will initially be considered the best alternative. A constraint that applies to the selection of the best alternative is that the acceptability of all goals must be within the tolerance region (i.e. above the limit), thus selecting an alternative that is overall acceptable. This means that an alternative with a goal categorised as 'ineligible' cannot be selected as best alternative, neither at the beginning of the stage nor during later parts of the stage.

Following identification of the best alternative, the participants examine the acceptability (described as willingness) of the alternative with respect to the dependability goals. The objective is to identify goals, the acceptability of which is in need of improvement. Improving such a goal motivates the stakeholders to choose a different alternative, which achieves higher acceptability with respect to that goal (these goals are referred to as motivating goals). Stakeholders should not seek to improve a (motivating) goal if it is in a higher willingness category than another goal. This would imply that the goal characterised with 'higher' willingness is more important. This is in conflict with the definition of willingness, making the resultant trade-off argument weak.

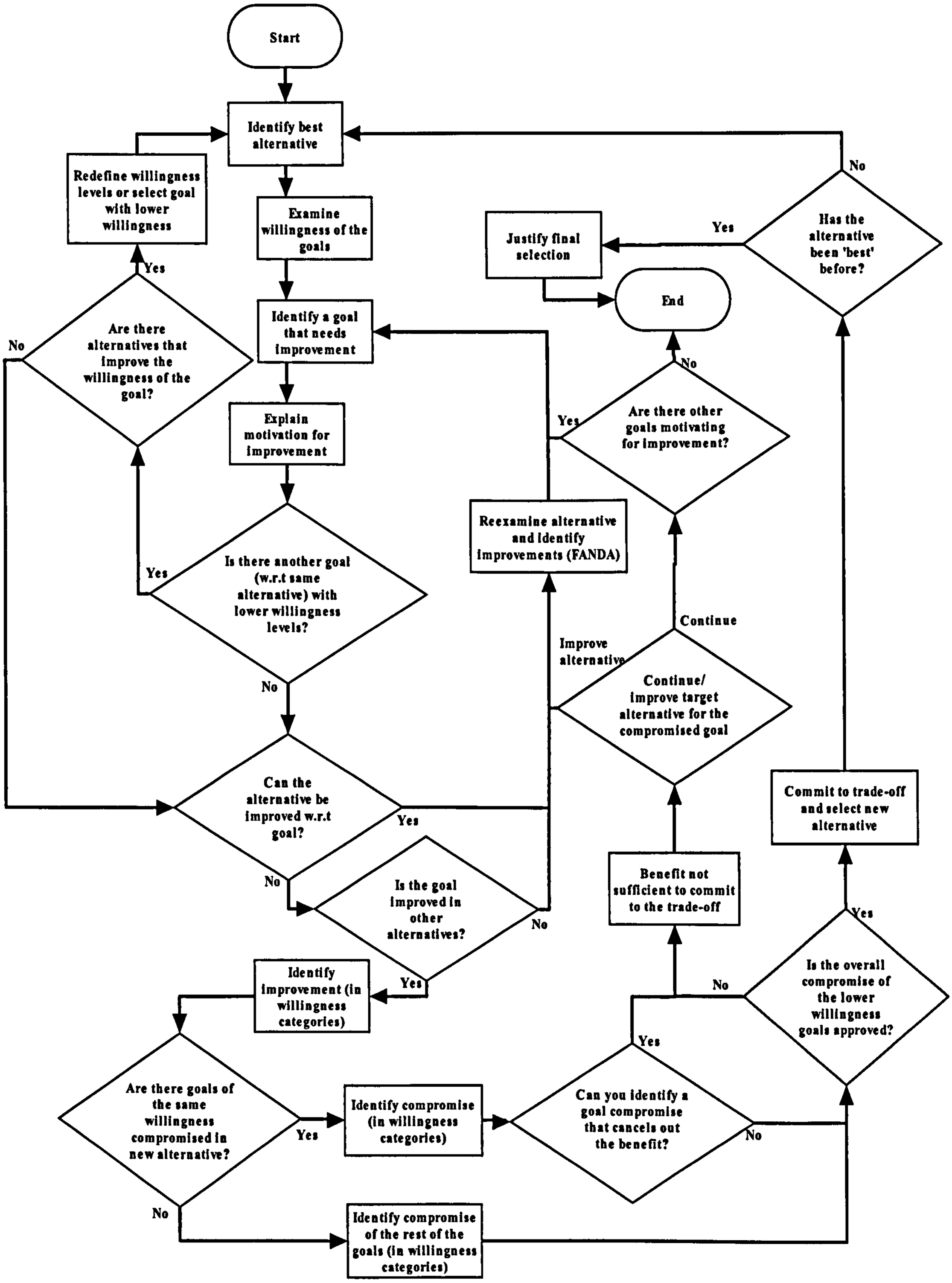

Fig.5.14 – Constructing the Trade-off Argument

If participants continue to feel a stronger motive to improve a goal with higher willingness level than another, the willingness levels of the goals must be re-examined as shown in Fig.5.11. Improving the degree of satisfaction of a motivating goal can be achieved in two ways. Firstly, participants may improve the design of the alternative. Altering the design is outside the scope of the trade-off method and the Factors, Analysis and Decision Alternatives (FANDA) method should be used (as presented in chapter 6). Alternatively, a goal can be improved by selecting another alternative (termed the candidate alternative) that best improves the willingness of the motivating goal. Improvement of a goal can entail compromising other goals.

Following identification of a candidate best alternative, participants evaluate whether they are prepared to accept the consequent trade-offs. Initially participants identify the goals of the same willingness which will be compromised by the selection of the target alternative. Selection of the target alternative is based on the participants' acceptance of the involved compromises. If a compromise is identified that cancels out the benefit from selecting the target alternative, feedback is provided to the design rationale process (by invoking FANDA) in order to improve the target alternative. If goals in the same willingness category as the motivating goal are not present, participants examine whether they can afford the rest of the goals being compromised. TOM provides a systematic analysis of the trade-offs involved in selecting each alternative. It facilitates stakeholders to argue about the selection of an alternative, by allowing them to reason about the motivations for selecting an alternative as well as objections for not selecting others.

However TOM does not provide a definite answer about which alternative should be selected. The final decision lies with the stakeholders who need to justify their selection. The argument about the selected alternative records the information on which the decision was based. Table.5.6 shows the resulting trade-off table after the first stages of TOM. In the first the acceptability (expressed in willingness) of the goals with respect to the degree of achievement of the goals was identified and in the second stage the alternatives for the FTA were evaluated against the goals.

Table.5.6 – FTA Trade-off Table

TRADE-OFF TABLE	Safety	STOL	Endurance
Alternative A	*Hesitant (cat. IV)*	*Potential (cat. III)*	*Ineligible (cat. V)*
Alternative B	*Probable (cat. II)*	*Potential (cat. III)*	*Hesitant (cat. IV)*
Alternative C	*Potential (cat. III)*	*Unconstrained (cat. I)*	*Hesitant (cat. IV)*

Following the process of the third stage of TOM a best alternative needs to be selected in order to start studying the involved trade-offs. Using the rule of thumb, alternative C was described as the best alternative as it had fewer goals in the lower willingness categories. Moreover, by examining the table it is immediately noticeable that alternative A is an unacceptable alternative, since the endurance this configuration will result in is below the lowest acceptable limit of the degree of satisfaction for the endurance goal. Identifying that a particular goal is unacceptable can lead participants to examine the design rationale of the alternative. The FANDA method was invoked, identifying how the characteristics of alternative A contributed in 'unacceptably' satisfying the endurance goal.

As a result alternative A was modified with the new design accommodating engines with slightly lower thrust that also weigh less. Changing these two factors was considered to improve the endurance of the FTA. During this step improving an alternative with respect to a goal may compromise it (the alternative) with respect to other goals. However in the example it was considered that other goals were not affected to a degree in which the willingness was changed. Table.5.7 shows the trade-off table after modifying alternative A and identifying alternative C as the best alternative. Examining the best alternative, safety was identified as a motivating goal for selecting another alternative (Table.5.8). Although endurance has lower willingness level cannot be improved as all other alternatives achieve the same level of willingness. Safety is improved only in alternative B (Table.5.8), which was decided that it would be the target best alternative.

Table.5.7 – Identification of Best Alternative

TRADE-OFF TABLE	Safety	STOL	Endurance
Alternative A	*Hesitant (cat. IV)*	*Potential (cat. III)*	~~*Ineligible (cat. V)*~~ *Hesitant (cat. IV)*
Alternative B	*Probable (cat. II)*	*Potential (cat. III)*	*Hesitant (cat. IV)*
Alternative C	***Potential (cat. III)***	***Unconstrained (cat. I)***	***Hesitant (cat. IV)***

Table.5.8 – Identification of Benefit

TRADE-OFF TABLE	Safety	STOL	Endurance
Alternative A	*Hesitant (cat. IV)*	*Potential (cat. III)*	~~*Ineligible (cat. V)*~~ *Hesitant (cat. IV)*
Alternative B	*Probable (cat. II)*	*Potential (cat. III)*	*Hesitant (cat. IV)*
Alternative C	***Potential (cat. III)***	***Unconstrained (cat. I)***	***Hesitant (cat. IV)***

In order to select the target alternative as best, participants need to examine the compromise involved (Table.5.9 & Table.5.10). As a consequence of improving the safety levels of the FTA, the acceptability of the STOL capability is compromised from category I to category III.

Table.5.9 – Identification of Compromise

TRADE-OFF TABLE	Safety	STOL	Endurance
Alternative A	*Hesitant (cat. IV)*	*Potential (cat. III)*	~~*Ineligible (cat. V)*~~ *Hesitant (cat. IV)*
Alternative B	*Probable (cat. II)*	*Potential (cat. III)*	*Hesitant (cat. IV)*
Alternative C	***Potential (cat. III)***	***Unconstrained (cat. I)***	***Hesitant (cat. IV)***

Table.5.10 – Evaluation of Best Alternative

TRADE-OFF TABLE	Safety	STOL	Endurance
Alternative A	*Hesitant (cat. IV)*	*Potential (cat. III)*	~~*Ineligible (cat. V)*~~ *Hesitant (cat. IV)*
Alternative B	*Probable (cat. II)*	*Potential (cat. III)*	*Hesitant (cat. IV)* OK
Alternative C	***Potential (cat. III)***	***Unconstrained (cat. I)***	***Hesitant (cat. IV)***

As already discussed, TOM does not provide a definite answer; final selection lies with the stakeholders. According to whether they approve or not the compromises involved with a goal improvement the final alternative selection is made. However justification of the final selection is required, as this is essential information to be recorded for the overall dependability case. In this case compromising STOL was not considered acceptable as the reduced take-off/landing theatre capability during operations was considered to outbalance the safety improvement.

5.8 The Trade-Off Argument

Upon completion of TOM, participants will have created arguments that ultimately guide them to selection of the best alternative. The arguments contain information about a number of issues addressed during the TOM processes, including the following:

- Identification and rationale of bounds
- Acceptability of the alternatives
- Preference between competing alternatives

The arguments resulting from TOM provide important contextual information to the dependability case. In order to use the arguments within a dependability case, modular GSN is used to capture them. Fig.5.15 shows the structure of the Trade-off argument module. The top level goal of the argument (*TradeOff*) claims selection of the most suitable decision among the candidate alternatives. The goal is stated in context of the dependability goals that the alternative needs to satisfy (*TradeOffC1*), and the candidate alternatives created during design evolution (*TradeOffC2*). In this thesis FANDA –

described in Chapter 6, is a design rationale method used to create the alternatives used in TOM.

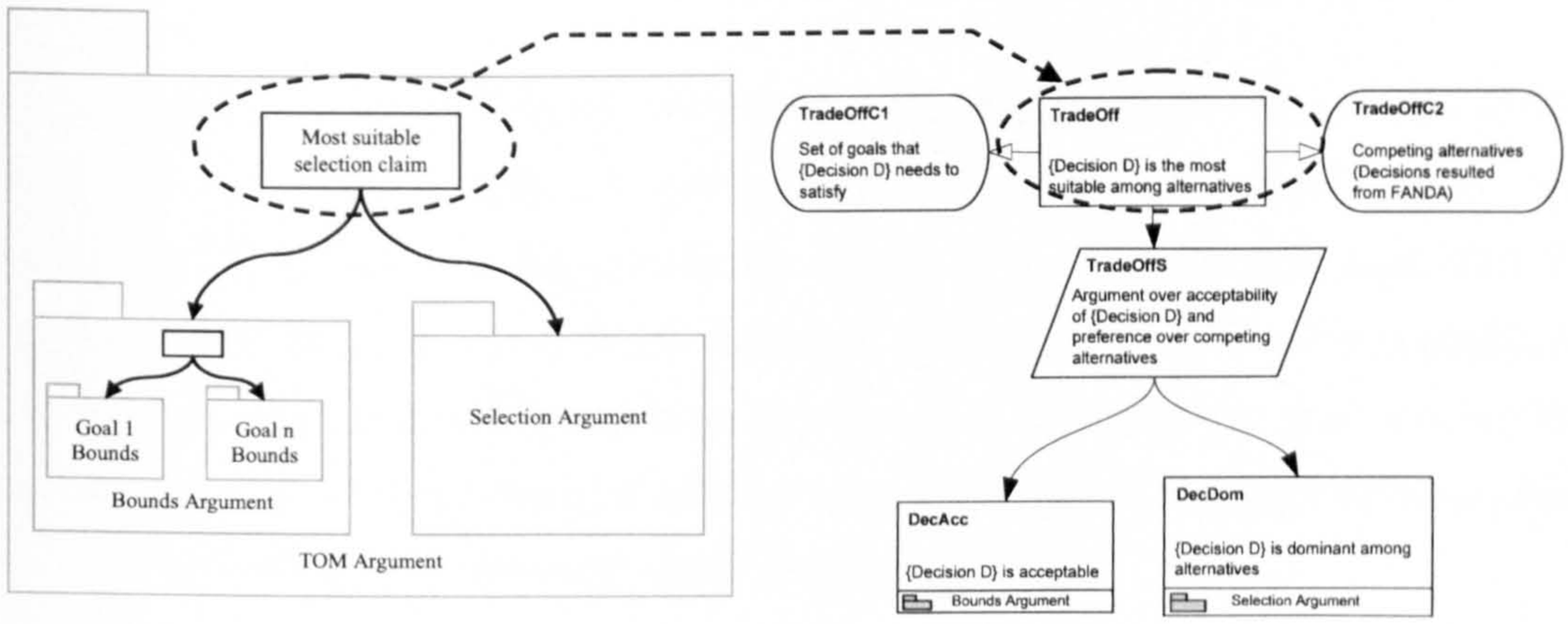

Fig.5.15 – Overview of the Trade-off Argument

The *TradeOff* goal is substantiated by two away goals (*DecAcc* and *DecDom)*, which correspond to separate argument modules (*Bounds Argument* and *Selection Argument*). The *Bounds Argument* is further decomposed to argument modules arguing about the apportionment of acceptability of a goal with respect to its degree of satisfaction. The left half of Fig.5.15 shows the overall structure of the trade-off argument. Decomposing the *Bounds Argument* to smaller modules about the acceptability of each goal, was adopted after considering the overall dependability case architecture (presented in Chapter 6). When a goal in the dependability case is stated in the context of a Target and Limit, it is also associated with an away (argument) context justifying the derived Bounds.

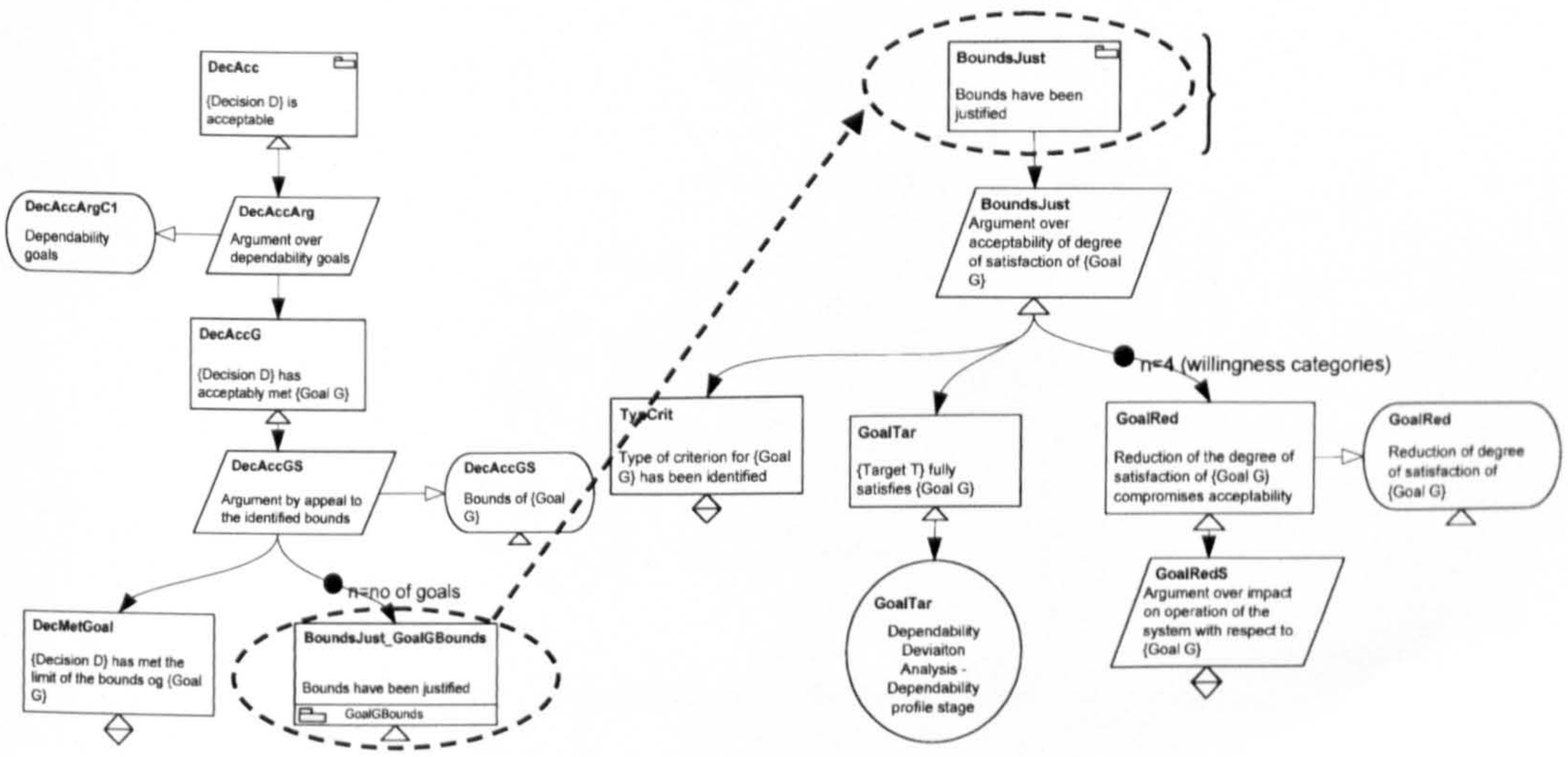

Fig.5.16 – Bounds Argument and Goal Bound Argument

By partitioning the trade-off module as shown in Fig.5.15, the *Goal Bounds* arguments can easily be referenced from other goals. Fig.5.16 shows the details of the *Bounds Argument* (left half), and the *Goal Bounds Argument* (right half).

The top level goal of the *Bounds Argument* claims that the decision selected during TOM is acceptable. This is substantiated by the goals *DecMetGoal* and *BoundsJust_GoalGbounds* – the latter being an away goal. The first goal states that the decision meets the limit, which is the minimum criterion for a goal to be acceptable. The second goals corresponds is instantiated for each dependability goal arguing the justification of the apportionment of willingness to the degree of satisfaction of the goal. The right half of Fig.5.16 shows the argument module.

The argument is decomposed based on identification of the acceptability for the various levels of degree of satisfaction of the goal. The top level claim is supported by the goals *TypCrit, GoalTar* and *GoalRed. TypCrit* documents the type of the criterion describing the target, and *GoalTar* the rationale for setting the target – elicited during the last stage of DDA. The goal *GoalRed* is instantiated for each willingness level capturing the impact a reduction of the degree of satisfaction of the goal has on the operation of the system.

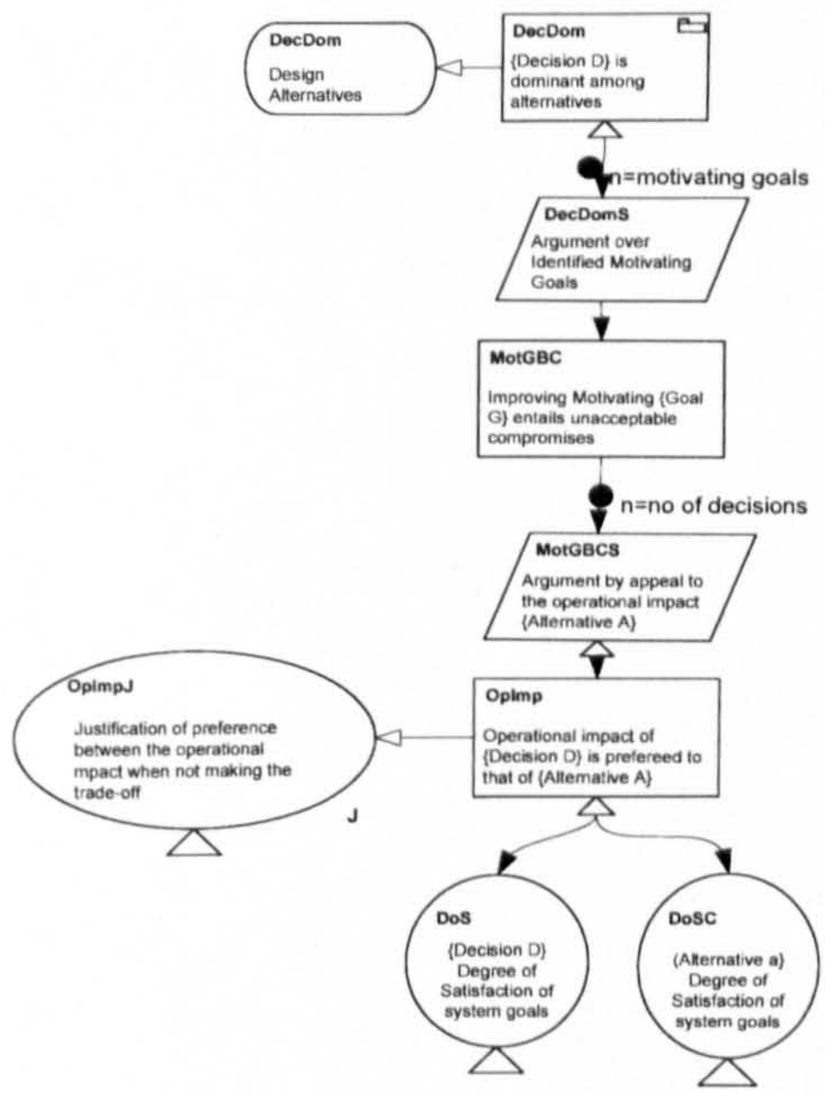

Fig.5.16 – Selection Argument

Achievement of the limit is not alone a sufficient argument for choosing an alternative. Participants also argue about the preference of the selected alternative with respect to the others. Fig.5.16 illustrates the *Selection Argument* of the trade-off argument module. The overall goal (*DecDom*) claims that participants identified the selected alternative as best among the candidates. The top level claim is supporting an argument stating that despite the presence of goal motivating the participants to select another alternative, the selected alternative was preferred. The final decision is stated in the context of a justification element stating why one alternative was preferred over the others.

5.9 Summary

This chapter has defined TOM; a qualitative method, for systematically identifying and arguing about trade-offs. TOM introduces the concept of bounds capturing an admissible space in which goals can be traded-off. In order to relate the degree of satisfaction of a goal to system operation, the concept of acceptability is introduced. Selection of a goal involves identification of benefit and compromise using the trade-off table. Finally, the chapter defines GSN arguments arguing about the selected decision, in the context of which the development of the dependability case takes place.

Intentionally Blank

Chapter 6

Evolution and Architecture of Dependability Cases

6.1 Introduction

Accepted practice in safety cases suggests that a case should be constructed incrementally in parallel with the system. In particular, safety standards such as the Ship Safety Management Handbook JSP430 [12] and the Defence Standard 00-56 [8] explicitly require development of the safety case to start at the beginning of the system lifecycle.

Evolution of the system and the argument involves making decisions about the architecture and the design of the system that need to be justified and documented. Interaction between the argument and the design process exists during the evolution of the system. The *evolving argument* should serve to *evaluate* the *design's* fitness to satisfy the stated dependability goals. A design that directly addresses the stated goals will result in a strong argument. If the stakeholders involved in the development of the argument deem that the argument is not satisfactory, changes to the design will have to be made. The top levels goals of the argument correspond to what needs to be claimed for the final system and accordingly appropriate goals are elicited according to the stage of the system development. Goals in GSN are specific claims that a system has achieved a particular requirement. Being able to explicitly represent and interrelate all the elements of an argument, GSN helps to articulate post-conditions for the initially identified requirements of the system in question.

As part of this decomposition, GSN captures the strategies based on which the goals were decomposed, the rationale for the argument approach adopted and the context in which the goals are stated. The argument of a dependability case can constitute the interface between evolution of requirements and design, providing a systematic way of documenting, tracing and reasoning about decisions. This chapter presents how GSN

arguments can be used as a design and assurance driver to evolve requirements and design in parallel. Additionally the chapter describes how we can establish a modular compositional dependability case relating to the structure of System of Systems.

6.2 Processes Participating in the Evolution of the Dependability Case.

Overall, the dependability case framework incorporates the three different methodologies which are proposed in the earlier chapters of this thesis. The participant methodologies collaborate during the evolution of the dependability argument, which itself is based on the Goal Structuring Notation (GSN) methodology and notation.

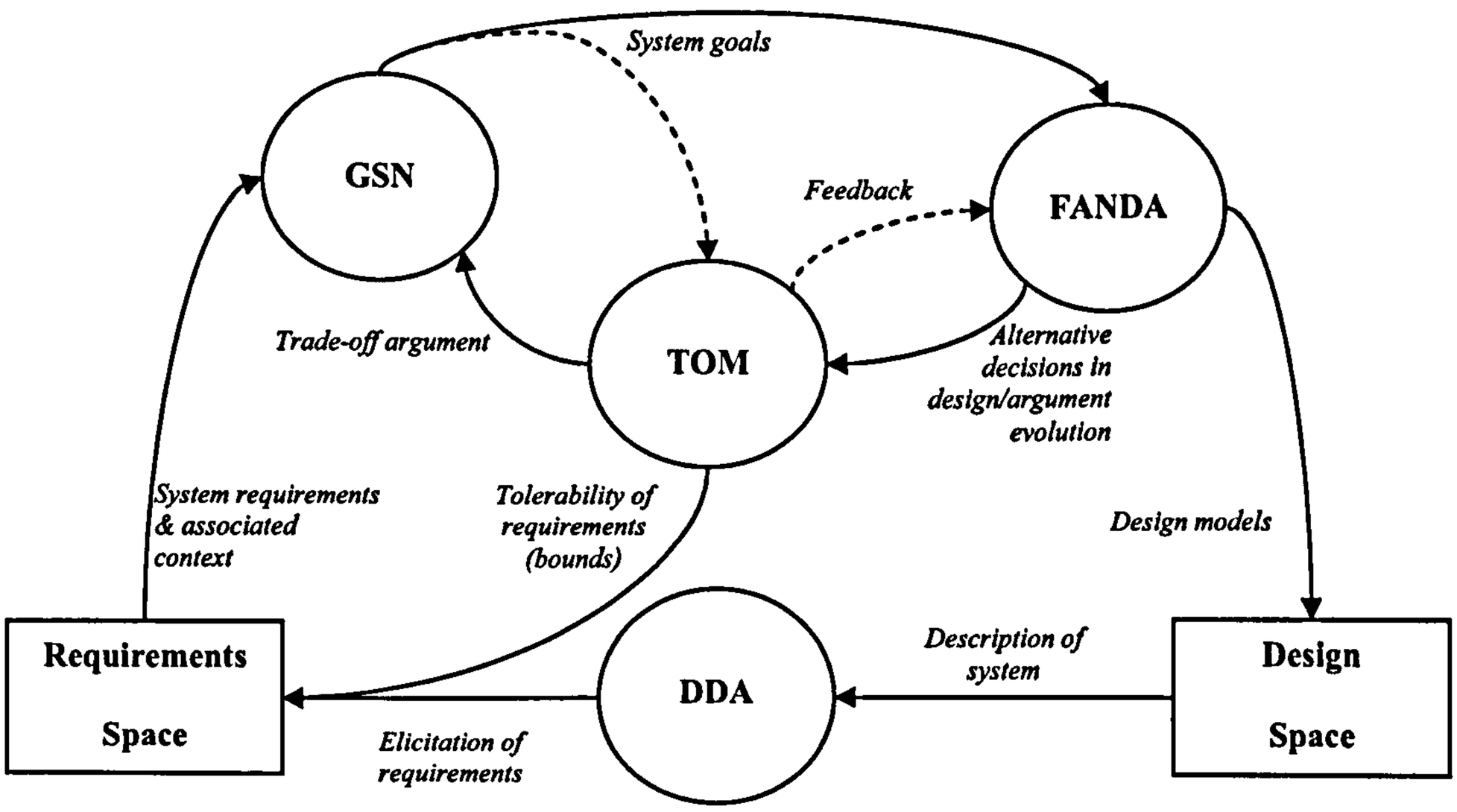

Fig.6.1 – Methods Supporting Argument Based System Evolution.

Fig.6.1 presents the collaboration between the methodologies to support argument based co-evolution of design and requirements, by showing the flow of information between them during system evolution. The methodologies participating in the evolution are the following:

- *Dependability Deviation Analysis (DDA)*: DDA (described in chapter 3) provides a method for the systematic analysis of the system identifying the effect of system deviations on the normal operation of the system. In the context

of dependability cases, the purpose of DDA is to elicit the required and acceptable behaviour of the system with respect to dependability and to clearly identify associations between failures from the viewpoint of each dependability attribute.

- *Goal Structuring Notation (GSN)*: GSN (reviewed in chapter 2) consists of a notation and method pertaining to the development of structured arguments. GSN is used to construct the core of the dependability case – the dependability argument. The dependability case framework has been built using GSN exploiting its ability to explicitly reference contextual information.

- *Factor ANalysis and Decision Alternatives (FANDA)*: FANDA (described in this chapter) complements GSN in supporting the development of arguments. The method can be thought of as a catalyst between the argument and the design. It assists analysts in examining the system goals, recording and managing rationale, and eliciting candidate options for the design decisions taking place during the system lifecycle.

- Trade-Off Method (TOM): TOM describes a systematic way of establishing and using a space of admissible requirements, to trade-off achievement of dependability goals, hence overcoming a potential impasse that resulted from conflicts between the goals. TOM ultimately creates arguments of preference between candidate decisions, in the context of which the dependability argument evolves.

The starting point of the evolution is considered to be the definition of the overall concept system operation – e.g. the definition of the concept of operations (CONOPS) and operational scenarios and roles of a system. This information constitutes the high level model of the system (e.g. OV-1 in DODAF). The models are used in DDA, during which analysts identify the attributes of interest and elicit dependability requirements for the system elements identified in the models. The GSN method is used to identify and create an argument about the dependability of the system. The requirements elicited during DDA constitute the required dependability behaviour of the system. The goals are used by the FANDA method to elicit design rationale and

identify the candidate design alternatives that satisfy the goals. The identified design alternatives are fed (along with the GSN goals) to the trade-off method, which will facilitate selection of the most suitable alternative. The first stage of TOM can also be called from the GSN in order to define the bounds. Moreover TOM can also provide feedback to FANDA in order to improve the identified alternatives. Following the selection of the most suitable alternative, the design space is updated with the models that reflect the design decisions. The new models or system elements can be analysed using DDA, thus starting a new evolution cycle for the argument and the system.

6.3 Influence of GSN on System Development

Creating a compelling dependability argument requires references to the product and the (system) development process. Showing that a system is safe may require references to decisions concerning, for example, the elimination or mitigation of identified hazards. Other goals may need to be supported by arguments regarding the operation of the system or elements of the system with respect to other dependability attributes. For example, claiming that a system is reliable may be argued from the design decision to adopt redundant units. An argument is closely related to the context within which it is stated. In this case the context would be the design decision to use redundant units. Had the argument construction been left for the end of the lifecycle, developers may have not decided to adopt a redundant design. This would eventually lead to construction of a weak argument concerning the system's reliability, or may even result in an unreliable system as the 'inefficiencies' of the design would be apparent.

6.3.1 Argument Strategies

A GSN goal structure evolves through a process in which goals are decomposed into sub-goals, until the sub-goals can be supported by evidence. Goals can be decomposed in a manner that analysts perceive is suitable in order to support the parent goal. Strategies are used in GSN to describe *how* the argument has been decomposed. Identification and elaboration of strategy takes place during steps 3, 4 and 5 of the six-step GSN method, as described in chapter 2. During these steps the developers of the argument need to identify the strategy based on which the argument should evolve, the contextual information necessary to make this decision, and the children goals that need

to be defined to support the parent goal, given the selected strategy. Fig.6.2 presents an example of strategy describing how goal *G3* is achieved by its sub-goals.

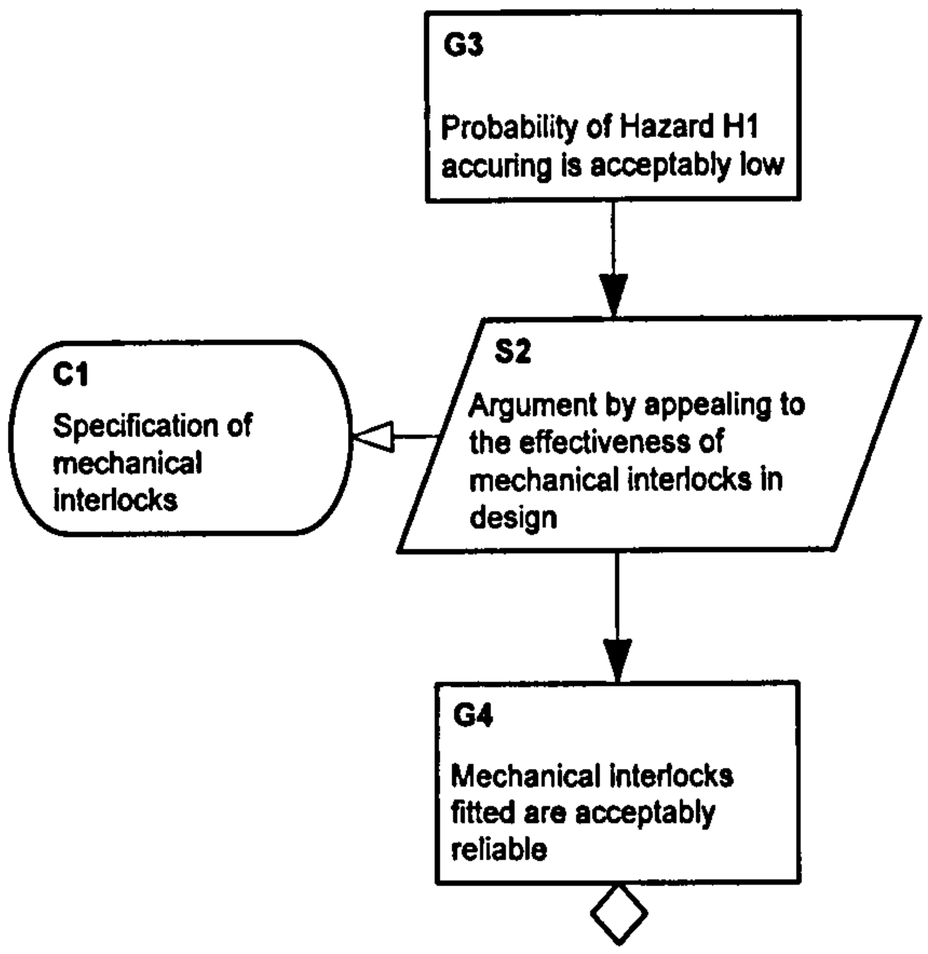

Fig.6.2 – Example of Strategy in Goal Decomposition [47]

The strategy argues over the effectiveness of a decision related to the design of the system. Strategy should communicate the approach which is used to decompose the argument. However decisions regarding the development of the argument are often closely related to decisions regarding the system. In the case of the example of Fig.6.2 the strategy is to argue by appeal to the effectiveness of the interlocks – stated in context of the interlock design decision. Kelly [47] remarks that although a strategy can refer to design decisions (as in the case of Fig.6.2), it should not be a mere reference to the design decision. Instead, this should be done using the design decision as context. GSN strategies are not only limited to describing design approaches. For example, a system argument may be developed to argue about the quality of procedures and processes of the development lifecycle.

Definition of argument strategy is a pivotal point during the evolution of the argument. Regardless of the particular focus of an argument the process of evolving the argument cannot be performed in isolation of the system. The process includes evaluation of the goals that need to be achieved and contextual information such as the evolution of design of the system up to that point, previous experience and architectural tactics. Deciding upon a strategy to develop an argument can influence many facets of the system lifecycle and ultimately affect the system itself. The decision for the adoption of a strategy (and its subsequent influence on development of the system) can originate out

of a set of possible alternative decisions. Whilst GSN documents all the necessary information required to review and understand an argument, the process resulting in the identification of the appropriate argument strategy and the information related to this decision are not recorded.

A situation in which the documentation of decisions becomes important is the activity of changing part of the argument. Changing an aspect of the system such as design, test data, goals will have a ripple effect to the argument [47]. Hence decisions that at the time that were taken were thought to be the best option may need to be reviewed. This requires reviewing the rationale as well as the argument, based on which they were preferred over other possible decisions in the first place.

In this chapter, FANDA is presented as a method to help developers understand and record the evolving dependability goals, the means of achieving the goals, and the resultant decision alternatives during the processes of parallel evolution of argument and system architecture and design.

6.4 Factor Analysis and Decision Alternatives (FANDA)

FANDA facilitates the interaction between the argument and the design. Eliciting design alternatives is an inductive process that uses inferences to associate goals with recognised factors that affect them. The combination of these design factors will form the proposed alternative. The process can use information collected from activities such as brainstorming, reuse of previous and general domain experience, application of patterns and experiments. The purpose of this step is to give developers the necessary impetus to produce candidate alternatives optimised for the satisfaction of the dependability case goals. FANDA provides the following functions:

- Brainstorming and elicitation of alternatives affecting the design of the system & the satisfaction of the dependability case goal.
- Documentation of decision rationale concerning the achievement of the goals
- Identification of competing decisions

- Relative impact of and identification of degree of satisfaction of decisions on goals
- Collation (and digest) of information regarding argument construction

FANDA consists of a number of elements that have been introduced in the DCM, as well as of a process that helps to instantiate and use FANDA.

6.4.1 Overview and Structure of FANDA Elements

Fig.6.3 depicts in UML the principal elements of FANDA as well as their structured, as defined in the dependability case metamodel.

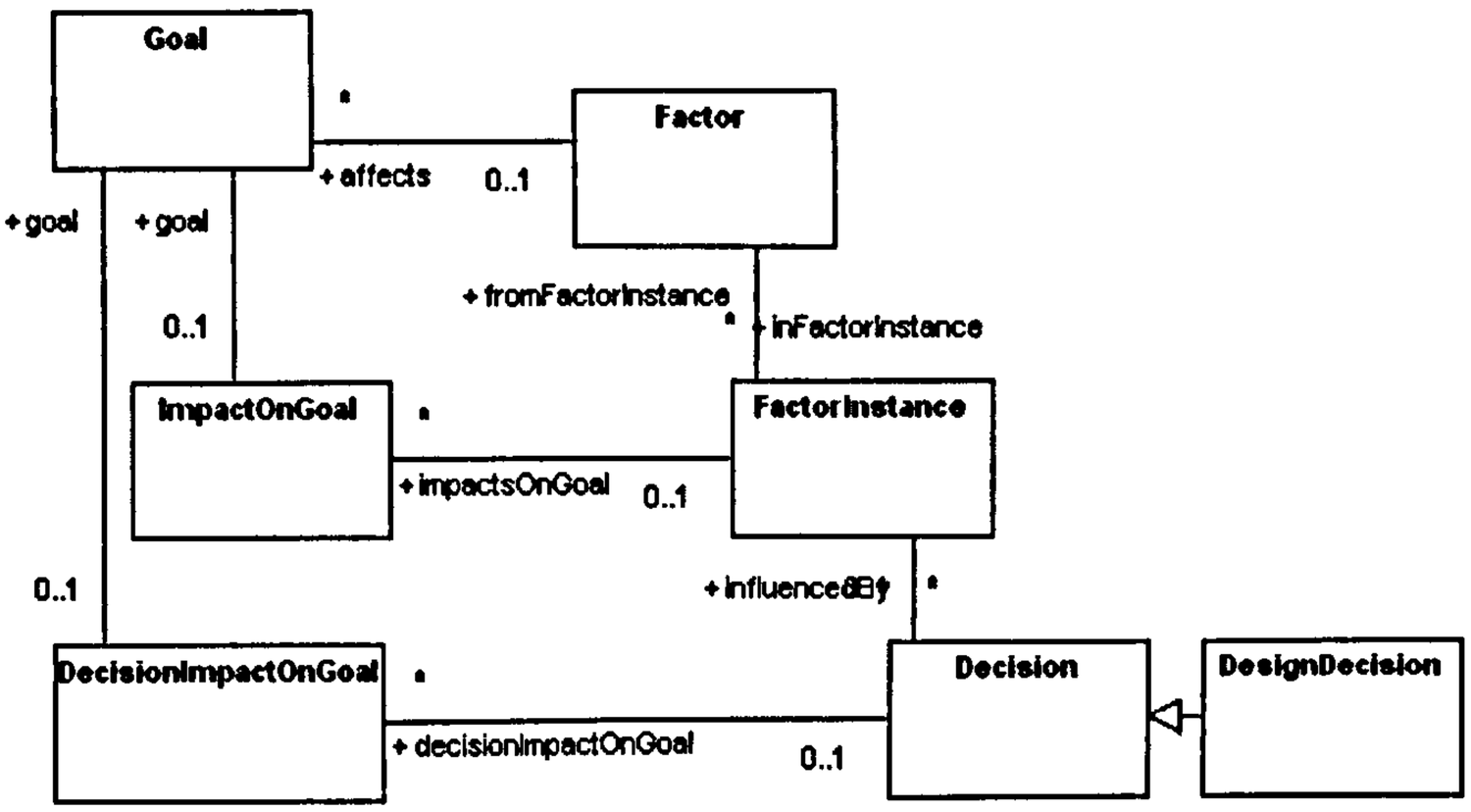

Fig.6.3 – FANDA Metamodel Description

6.4.1.1 Goal

The class 'goal' represents the GSN goals that exist in the dependability case (described in chapter 3). Goals participate in FANDA, as they are the epicentre (and starting point) of the process around which the alternatives are proposed.

6.4.1.2 Factor

Factors are a concept analogous to sensitivity points [37]. Their sole purpose is to identify possible characteristics that may affect a goal. In the context of this thesis, factors relate to design characteristics that are considered to influence the goal. For

example *redundancy* in a design is a factor that can affect a goal regarding the availability of the system, or the *strength of encryption* of data affects the security levels of a system. According to the goals that refer to, factors can be generic statements. For example identification of redundancy as a factor, refers to the architectural layer of the system, and would take place during the initial stages of the system development.

6.4.1.3 Factor Instance

Although a factor identifies an association of a feature with a goal, it does not provide an indication of how the observation is used in the system under development. A factor instance essentially is a concrete proposal for using a factor in a particular way, in order to achieve the goal. For example with respect to strength of encryption a factor instance could be 128-bits encryption, or with redundancy the factor instance could be use of triple modular redundancy.

6.4.1.4 Decision

Decisions represent the actual alternatives created from the FANDA process that constitute the proposed solutions regarding with respect to the dependability goals. It is these decisions that are evaluated during the trade-off method. A decision consists of a number of factor instances. Hence it is related to all goals that the designers need to achieve. For example, a decision would be triple modular redundancy architecture using 128 bit encryption. This thesis focuses on *design decisions*; however other types of decisions can exist. For example there can be decisions regarding the process followed to develop a system and not the product (design) itself. Such decisions could be modelled in the DCM by extending the decision class.

6.4.1.5 Impact on Goal

Factors acknowledge an association between the design characteristic that they represent, and a goal. However they do capture *how* the factor will affect the goal. Impact on goal is an association class between the factor instance and the goal. Given a concrete use of a particular factor, the impact captures how the proposed factor instance affects the goal. Impact of goal has two types of attributes: type of impact and magnitude of impact. The type of impact captures whether the factor instance affects a

goal positively, negatively or it may be neutral. The magnitude captures the extent to which the factor instance affects the goal. Magnitude of impact is described in a qualitative scale including low, medium and high.

6.4.1.6 Decision Impact on goal

Having identified factor instances and their impact on the goals (decision) alternatives are proposed. Decision impact on goal captures the impact of the proposed alternatives on the goals. Contrary to impact of goal which is mainly used to trigger brainstorming, decision impact should be accurate. In essence, decision impact mirrors the degree of satisfaction of an alternative with respect to the goal, used during TOM.

6.4.2 FANDA Process

The FANDA process provides a systematic way of examining the required dependability goals and eliciting the decision alternatives. The FANDA process is structured using the six hats method [98], which helps stakeholders reach consensus.

6.4.2.1 Six Hats Method

The strength of the method lies in the fact that decisions are not achieved through a process of debate. Instead, participants identify a number of perspectives from which a problem is viewed, focusing on each in isolation. This increases the productivity and effectiveness by structuring brainstorming, which often is an ad-hoc activity. Each of the perspectives represents a thinking hat switching the participant's mindset to particular thinking philosophy, forming an attitude that the stakeholders can follow. The six thinking hats are labelled as colours for ease of remembering their function, namely:

- **White hat:** The white hat is concerned with facts and figures. The purpose of this perspective is to identify the available data that can be used to support positions. The white hat aims to be neutral and objective and therefore personal opinion is dismissed in white hat thinking. However, facts that cannot be verified are permissible and are denoted as beliefs. Further investigation and collection of data about a belief needs to take place. Apart from the availability

of information that supports a fact, the likelihood of a fact being true is also an important attribute. The concept of likelihood separates the concepts of a generally acceptable truth and a fact. For example, it can be said that that it is generally true that avoiding loops will decrease complexity of a program; however this may not necessarily be a universal rule and there may not be adequate evidence to support this claim. The two concepts (acceptable truth and fact) have been incorporated in the DCM.

- **Red hat**: This hat provides an emotional view and allows feelings on an issue without having to justify the position. It is suggested by De Bono that there should not be any attempts to justify an opinion. Intuition in expressing an opinion is represented in this perspective.

- **Black hat**: Black represents caution and its purpose is to focus on the negative and weaknesses of an idea. Under black hat thinking participants are asked to answer questions such as "what can go wrong". Black hat thinking can also identify a suggestion that is not supported by facts. De Bono further advocates that black hat thinking should not degenerate into an adversarial argument.

- **Yellow hat**: Optimism and thinking about the positive aspects of an issue are covered under the yellow hat. Yellow hat thinking explores possible value and benefit.

- **Green hat**: Creativity and proposal of new ideas is the focus of the green hat. This perspective is involved with what can be done and what alternatives can be produced regarding an issue.

- **Blue hat**: The blue hat is responsible for organising the overall process and use of the other hats. It can be said that the author of this research defined the use of the hats in FANDA from the perspective of the blue hat.

The six hats methodology influences the FANDA by constraining the attitude of the participants during each of the FANDA processes, focusing only on the necessary perspective at each stage.

6.4.3 Overview of the Process

FANDA consists of three stages, elicitation of factors and factor instances (1st stage) and goal wide examination of factor instances (2nd stage) and elicitation and examination of decisions (3rd stage), illustrated in Fig.6.4. Each stage is further analysed using dedicated process diagrams in subsequent sections of this report.

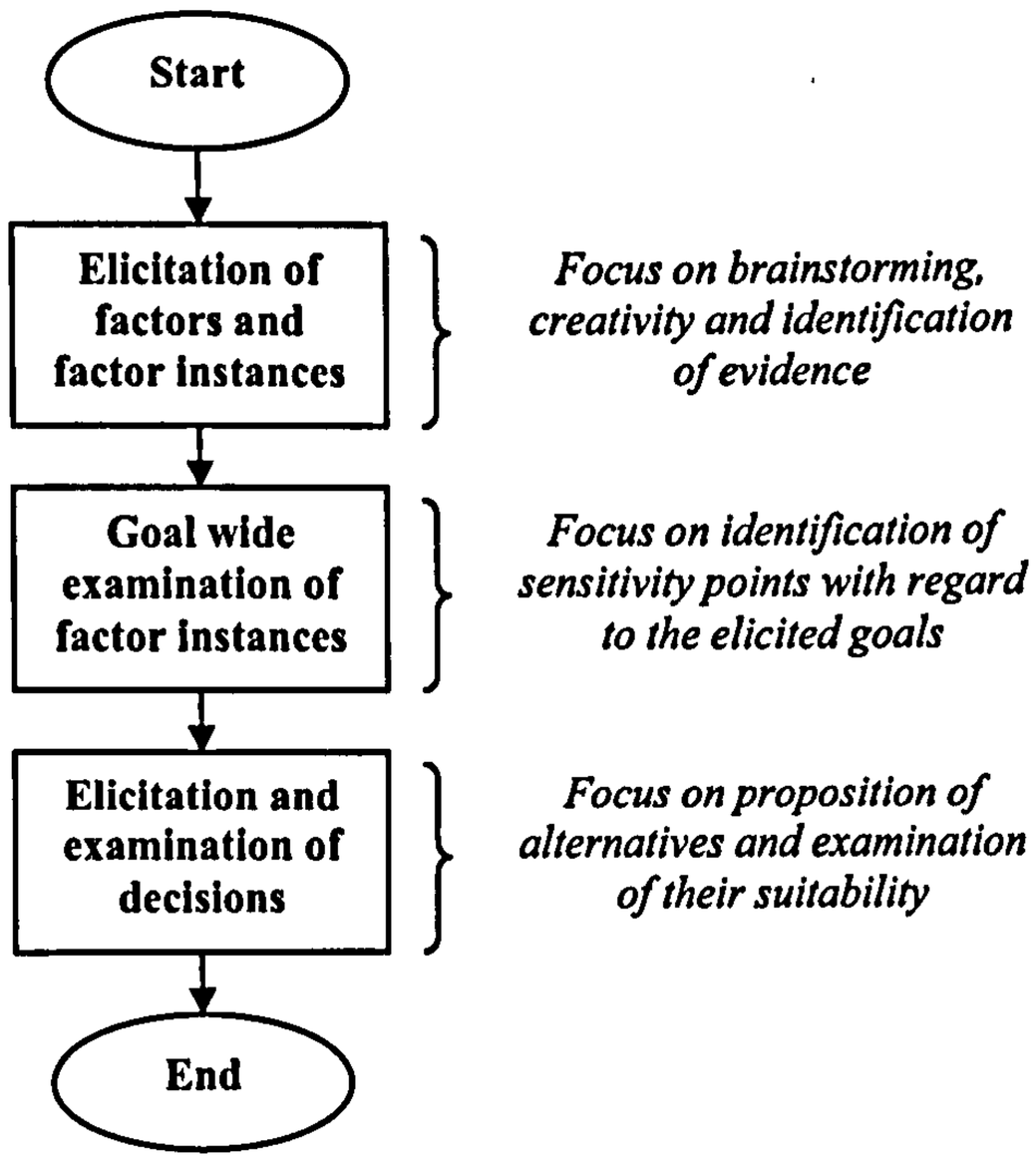

Fig.6.4 – Overall FANDA Stages

The objectives of the first stage are:

- To identify the goals that the alternative is required to satisfy
- Identify the factors affecting the goals
- Suggest ways forward using the factor
- Explore possible factor instances and identify their impact on the goal

- Examine the availability and plan collection of evidence supporting the factor instance's impact on goal

The focus of the first stage is limited on a single goal. Not taking into account the impact a factor may have on other goals, the participants of this stage aim to maximise creativity. Critical thinking is part of the second stage, which examines the factor instances with respect to the entire set of the dependability goals. The objectives of the second stage are:

- Identify the impact of factor instances across all goals create a rationale map
- Identification of evidence suggesting negative impact

Following identification of the impact of factor instances across all goals, participants have created a (as mentioned by De Bono) 'rationale map'. The map captures possible ways forward identified by the participants and both the positive and negative impact that they will have on the goals. The last stage of FANDA involves:

- Creation of decisions
- Specification of their impact on all goals, according to the factor instances that they contain
- Identification of insufficient evidence and planning of evidence collection

6.4.3.1 Elicitation of Factors and Factor Instances Stage

Fig.6.5 presents in a flowchart diagram the process for instantiating factors and factor instances. The process starts by importing, from the argument, the dependability goals that participate in FANDA, and selecting a goal that will be examined. By examining the goals individually the scope of the process is reduced and the appropriate focus is given on an individual problem at a time.

Identification of the goals includes examining the context of a goal, as well as any inherited context (when a goal is decomposed using the GSN method, the children of that goal are also stated in the context associated with the parent goal). Hence any

information from the context associated with a parent goal needs to be acknowledged as it may provide useful information.

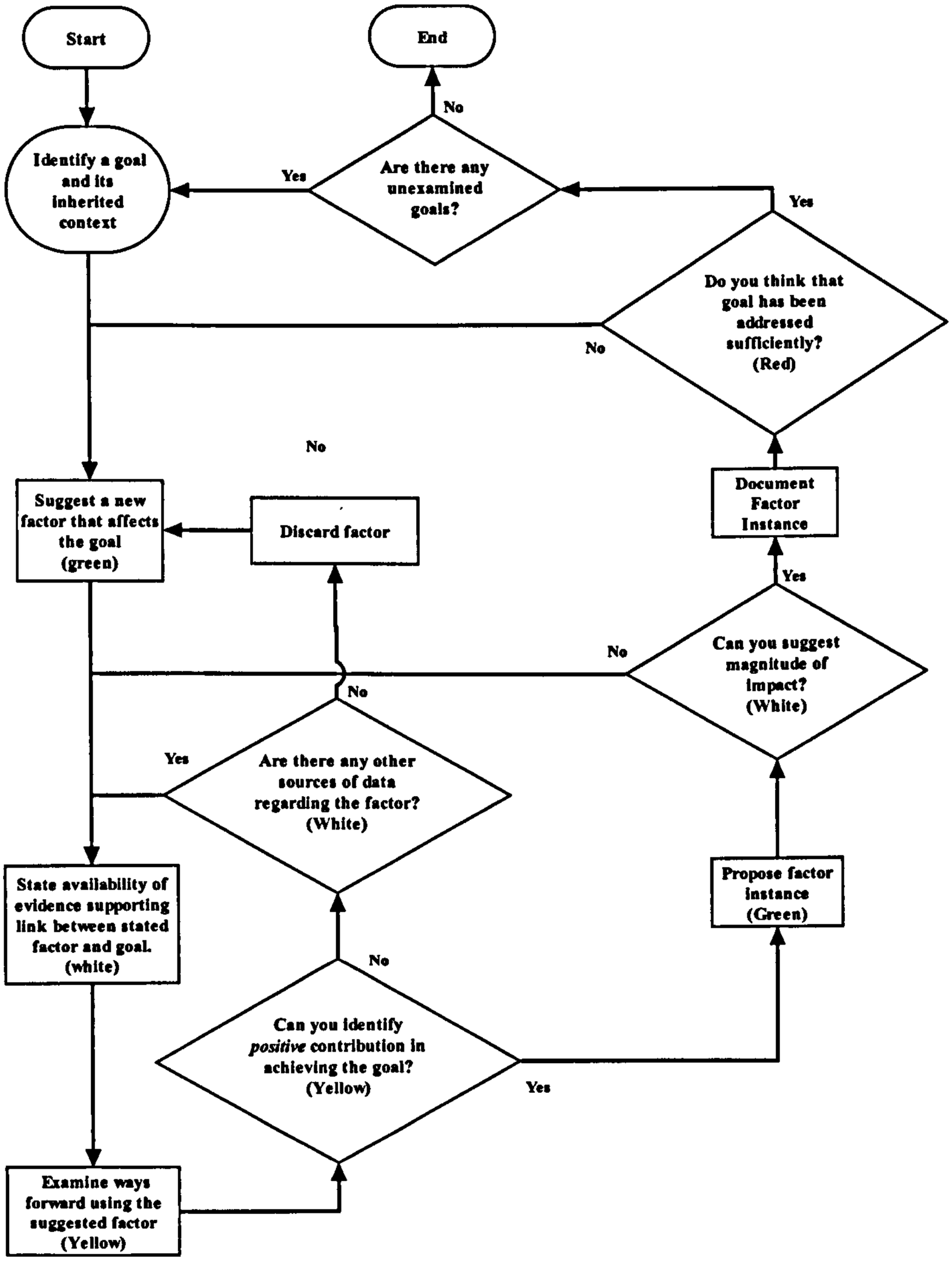

Fig.6.5 – Process for Instantiating Factors

Following the selection of a goal, participants suggest factors that possibly affect the goal. At this stage, a factor can be anything that it is thought to affect the ability to address the goal. Using a 'green hat' attitude the participants need to assert factors that they think can influence the goal without necessarily analysing at this stage the validity of the claim. Inspection of the validity and usefulness of a factor takes place gradually throughout the process. Initially an assessment of available facts regarding the factor needs to be performed. Participants need to look for sources of facts regarding the

factor and its associated goal such as experimental results, and data from previous experience and proofs. Where a factor is suggested but there is insufficient evidence to support it, the association between the fact and the goal should be characterised as belief and not as claim. This means that there is uncertainty whether the association is true. Hence the association between the factor and the goal is expressed using likelihood. Participants need to plan how sources of evidence can be further verified to support the association with more certainty. At this point of the process, as suggested by the white hat, interpretation of the facts in order to conclude as to whether a factor will affect the goal positively or negatively should be avoided. The task is solely limited to identifying the availability of evidence regarding the association between the goal and the identified factor.

Instantiation of factors should first take place using a positive and optimistic attitude. In other words the participants seek factors that will contribute to the satisfaction of a goal. This is reflected in the next step of the process, in which a hypothesis on how the factor can be used to support the satisfaction of a goal is suggested. Although this step needs to take place with a positive attitude, a suggestion needs to be supported by rationale, explaining the available evidence supporting the hypothesis. Caution should be given not to criticise the hypothesis but to maintain a focus on whether the suggested factor can contribute positively to the goal given the available evidence. Identification of positive contribution and definition of the rationale is a stepping stone for proposing a factor instance. Proposing a factor instance entails a concrete specification of how the factor (which was identified as having a positive contribution to the goal) could be used to achieve the goal. The definition of the factor instance is accompanied by definition of the magnitude of impact. This captures the degree that the factor instance is believed to contribute in achieving the goal (low, medium, high), which is an attribute of the *ImpactOnGoal* class in the FANDA metamodel (Fig.6.3). Identified sources of information can provide evidence supporting the decision. Eventually the factor instances that have high impact on goal will be given extra consideration when an alternative an alternative will be specified. It is these factors that will influence the dependability attributes of the proposed alternative.

The process ends with the participants determining whether the identified factors, along with the factor instances, can sufficiently help support the goal. A thorough analysis

and justification for having addressed the goal adequately is not required. Completion of the stage (for each goal) is a subjective decision of the participants, based on whether the brainstorming activity suggesting factors as well as elaborating on their use, has been effective enough.

6.4.3.2 Goal-wide Examination of Factor Instances Stage

During the first stage of FANDA factor instances are proposed driven by identification of a factor that can (positively) contribute in achieving the goal. Factor instances will not have a positive impact on all goals and may affect certain goals in a negative manner (black hat mentality). This part of FANDA involves examining a factor instance with respect to the rest of the goals, also identifying negative impact. The main reason for which examination of possible negative does not take place during the instantiation of the factors (first stage of FANDA) is to avoid rejecting a factor instance on the grounds of negative contribution. A factor instance and the corresponding factor and approaches should not be rejected before examining how it can contribute to the overall decision.

Identifying potential negative contribution of a factor instance and collecting the appropriate evidence, which takes place during this stage is an important activity. The concept is similar to the requirement of Defence Standard 00-56 [6], which asks for the identification of possible counter evidence that could have a negative impact on the creation of a safety case. Fig.6.6 presents the process for examining the factor instances with respect to all required goals.

The process begins by selecting a proposed factor instance. Using the appropriate mindset as described in the six hats methods, participants need to identify whether the factor instance will have, in their opinion, a negative or positive impact on a goal. Impact in this case can also be described as neutral. The possibility of neutral impact is considered during the use of the positive thinking (yellow) hat. This is in accordance to maintaining an overall positive attitude for the benefit of creativity. Failure to identify impact on a goal implies insufficient information about the relationship between the factor instance and the goal. In case the impact is considered to be neutral this should still be supported by evidence. Lack of evidence does not imply a neutral impact.

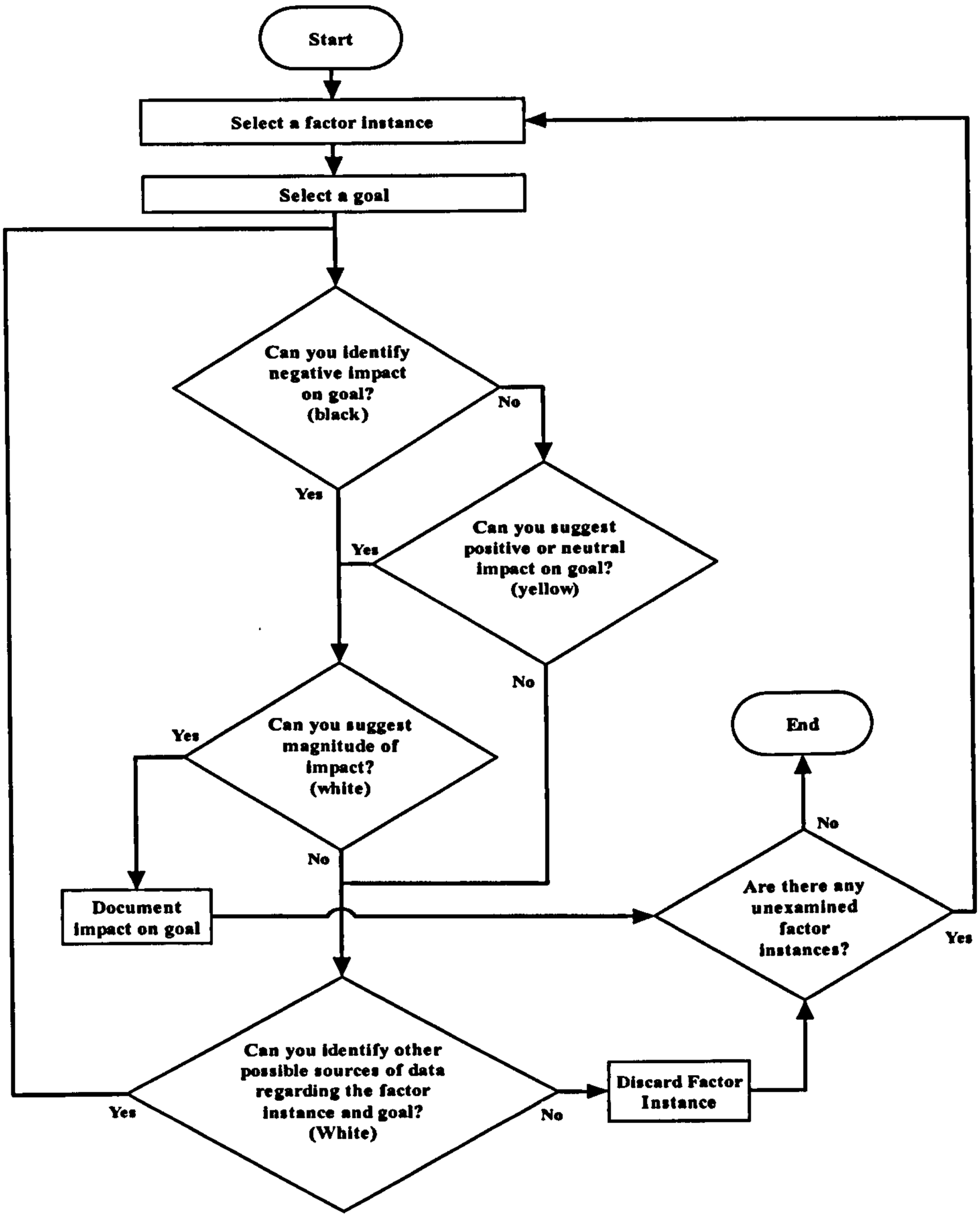

Fig.6.6 – Goal – wide Examination of Factor Instances

Identifying available sources of evidence that can support a suggestion regarding the impact of a factor instance is necessary for evaluating of the overall decision.

6.4.3.3 Specification of Decisions Stage

Following completion of the second stage, participants have created an overview of the factors that affect the required goals, and have created factor instances using the identified factors. The last stage of FANDA involves composition of factor instances and proposition of candidate (design) decisions.

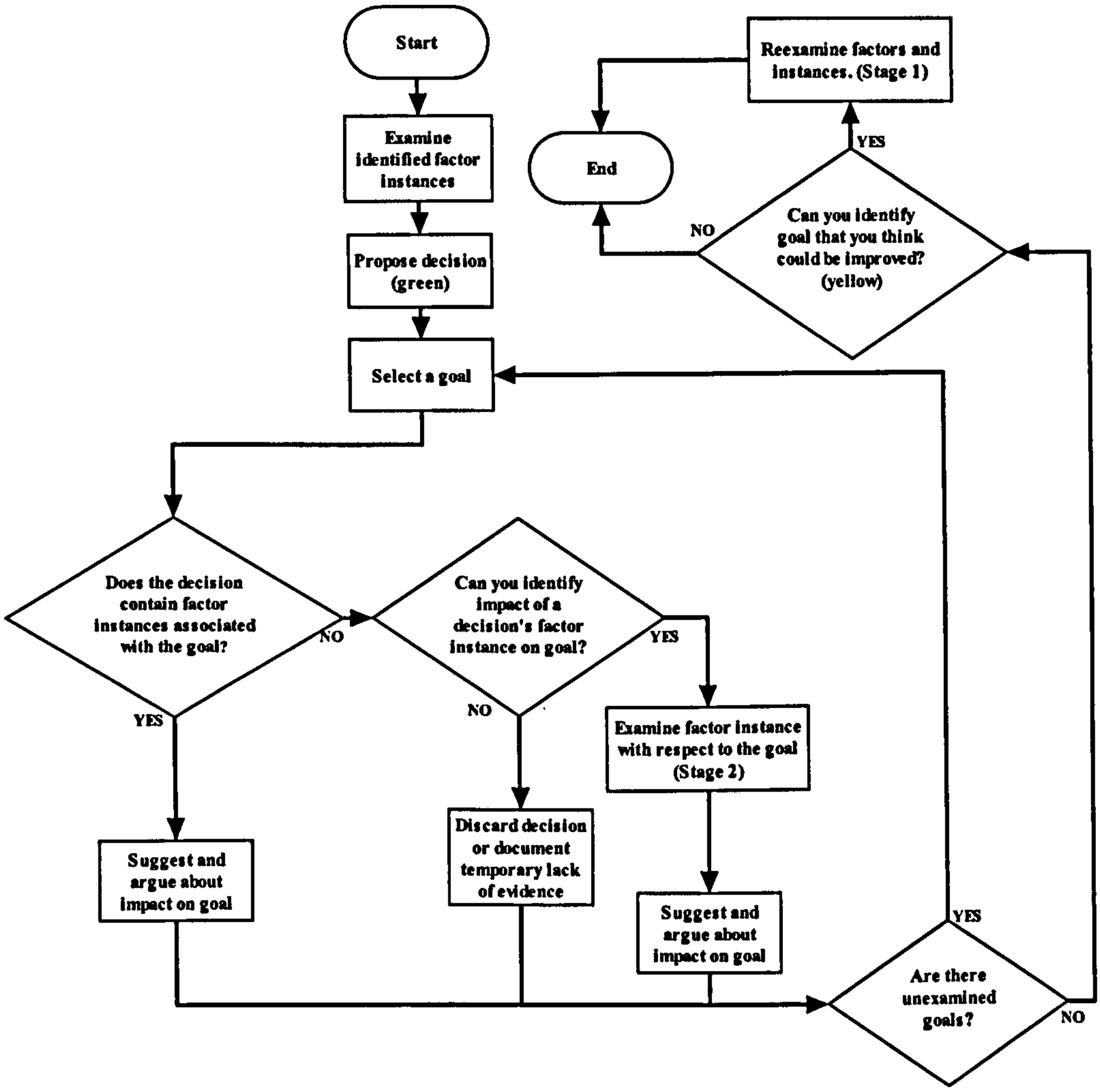

Fig.6.7 – Specification of Decisions Process

Fig.6.7 shows the process of the last stage of FANDA. There are no specific guidelines as to how the decisions should be made. De Bono suggests that by identifying, understanding and discussing the impact of each factor instance on each goal, a 'rationale map' is built that contributes in enhancing the perception of the problem. Following proposal of a decision participants investigate whether there are associated factor instances belonging to the decision with the goals. By examining the associations between factor instances and goals already defined during the previous steps of FANDA can identify the impact of the decision on the goals. There is the possibility that the proposed decision does not have any factor instances associated with one of the goals. In this occasion participants should identify sources of evidence that can help establish the association. In order to do this, the impact of the decision's factor instances should be assessed using the steps of stage 2. Failure to identify impact of decision on a goal

implies inconclusive sources of evidence that will later support the argument and hence are discarded.

6.4.4 Availability of Evidence during the System Lifecycle

With FANDA participants specify factor instances and decisions, and define their association with the system goals. During FANDA, participants enter an incremental process of establishing a collection of 'micro-arguments' (although not captured in GSN as with the main dependability argument) about the association of factors, factor instances and decisions with the goals. Evidence supporting these micro-arguments are an essential aspect of the FANDA process. However, availability of evidence varies according to stage of the system lifecycle. At the early stages of the lifecycle, availability of evidence is more limited compared to later stages. Booch [99] suggests that defining an architecture requires *"...suboptimal decisions made partly in the dark"*. Initially the claims in FANDA may be supported by beliefs (e.g. based on previous experience or common practice), which need to be verified by evidence as the development of the system progresses. Apart from identifying available information, it is essential during FANDA to think about evidence collection required to support decisions made at an earlier stage of the lifecycle based on beliefs or limited evidence.

6.5 Architecting the Dependability Case

At the end of its development, dependability case will combine claims regarding a number of perspectives of the system development such as the trade-offs, the product and the process. Consequently there will be differences in the arguments supporting these claims. Techniques such as modularising arguments in GSN, allow the overall dependability case to be represented as a series of interrelated modularised arguments. According to how the links between the arguments are defined the overall case can result in different layouts (architecture). This section defines a possible dependability case architecture. There are three main concepts are which have been used by the author to structure the dependability case:

- Use of a high level dependability argument
- Use of the dependability profile to structure the supporting arguments

- Use of GSN contracts to compose the overall dependability case

The following sections describe how these concepts contribute to the proposed dependability case architecture.

6.5.1 High Level Dependability Argument

A dependability case communicates assurance that the system will provide acceptable operation with respect to the stakeholders' envisioned dependability requirements. The overall dependability argument constitutes a description of the stakeholder's dependability goals. By constructing the high level dependability argument, stakeholders determine the dependability attributes of interest for the system, their respective requirements and associated acceptability criteria (i.e. the bounds of the goal), which are elicited during DDA. The high level argument predominantly captures the requirements of the system in context of its envisioned operation.

One of the advantages of using modular GSN is the achievement of separation of concerns between the argument modules. Fig.6.8 shows application of modular GSN to separate high level dependability argument from other supporting modules.

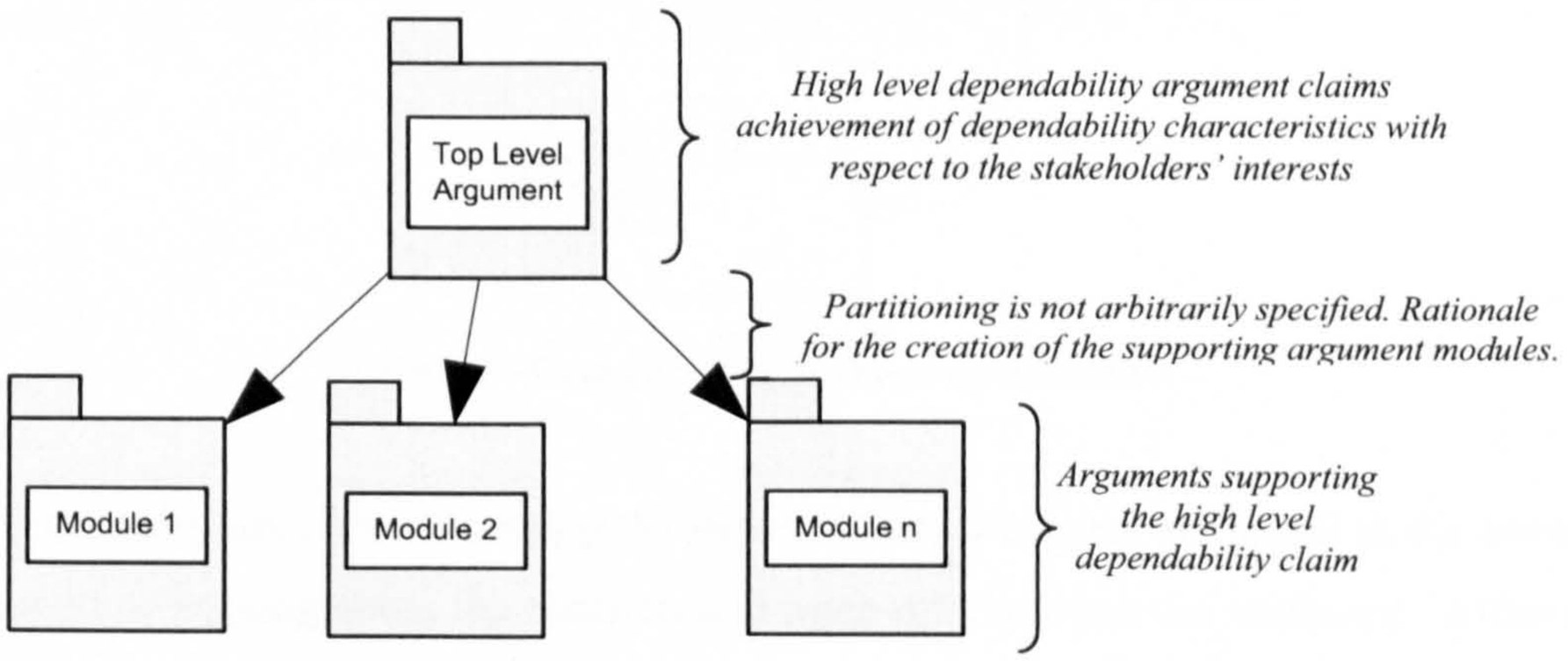

Fig.6.8 – High Level Dependability Argument and Support Modules

The focus of the high level dependability argument is on the high level dependability requirements of interest to the system stakeholders, which also define the system's high level envisioned operation.

Supporting modules provide the arguments for achievement of the requirements identified in the high level dependability argument. Partitioning of the dependability case cannot be arbitrarily specified. The rationale for a particular way of partitioning the dependability case needs to identified, examining the (logical) relationship between the high level argument and the arguments supporting it. A possible way to architecting the dependability case is to use the high level dependability argument as a means for separating individual arguments regarding the dependability attributes. Fig.6.9 shows the basic structure of the resulting dependability case using a high level dependability argument. The high level dependability argument (called dependability specification argument in this example) is used to identify the required system dependable operation. Following identification of the goals of dependable operation, each of the attributes is supported by separate arguments focusing on arguments pertinent to that attribute (Table.3.1).

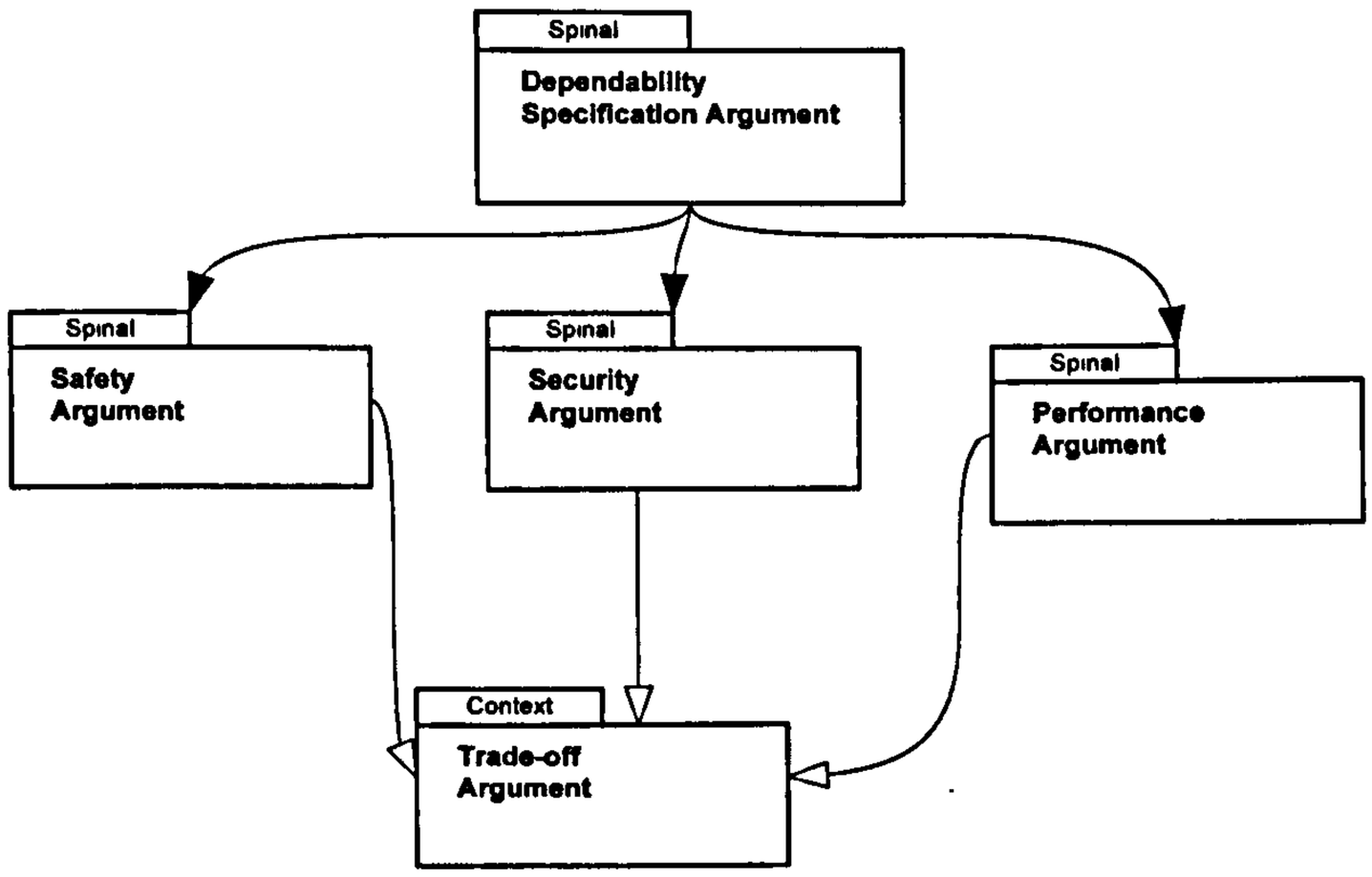

Fig.6.9 – Case Partitioning Based on Attributes

The argument modules concerning the dependability attributes are stated in the context of a module arguing about the conflicts and trade-offs between the attributes. Although this architecture for a dependability case is appealing there are certain problems that make its use difficult for communicating a clear argument about the achievement of the required dependability properties.

As demonstrated in chapter 4, there are associations between dependability attributes which during DDA are captured as 'can cause' relationships between failures. This dependency is carried through to the respective goals that were elicited during DDA. For example, consider a safety goal for the AGO scenario, claiming achievement of satisfactory safety (loss of life) where this has been identified as being a system concern during application of DDA. During DDA the consequences of failure condition 'Artillery will receive the wrong location' (FC6) were associated with the safety concern. Correspondingly, the claim about achievement of safety will depend on achievement of reliability in those system elements that participate in transmitting target data. Hence, a requirement stated from the perspective of one attribute may be supported by claims stated from the perspectives of other dependability attributes. Making many cross-references between the argument modules as shown in Fig.6.9 will result in a case that is difficult to comprehend and review. An alternative, more viable, approach to structuring the architecture of a dependability case is presented in section 6.5.3.

Establishing any high level dependability argument will entail the following aspects:

- Top level claim: The overall claim that the dependability case communicates.
- Attributes of Interest: Goals defining the required (high level) operation of the system regarding the dependability attributes of interest to the system stakeholders.
- Relation to DDA analysis: identification of how DDA can be used to identify the stakeholders' concerns and how these can be supported using the products of DDA.

The individual aspects of the high level argument (i.e. the arguments contained within the high level dependability argument *module*) are described in the following sections.

6.5.1.1 Top Claim

Dependability Deviation Analysis, as described in chapter 2, provides a framework for thinking about how each attribute can affect the operation of the system. Establishing

the dependability claims that the system will need to satisfy are determined during the first stages of the DDA.

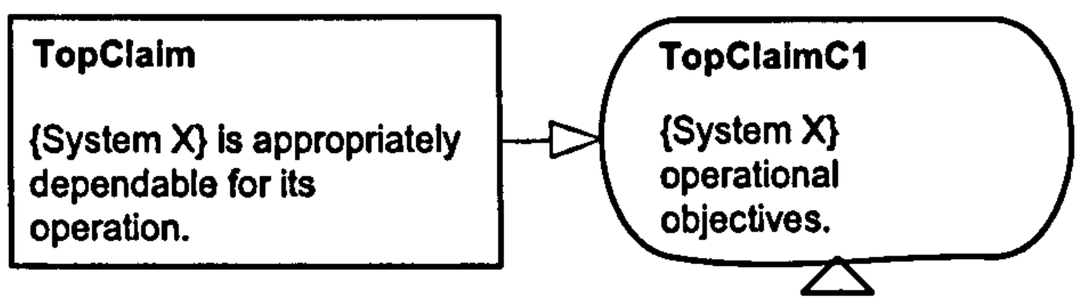

Fig.6.10 – Top Claim in Context of the System's Operation

The top claim (Fig.6.10) of the dependability case represents the overall claim regarding the envisioned operation of the system. Correspondingly, the goal is stated in the context of the objectives of the operation of the system. The top level claim is the starting point of the argument. In the initial stages of dependability case development references to dependability attributes cannot be meaningfully established. Hence at this stage, identifying the operational objectives of the system can provide the necessary background for decomposition of the top level claim, in which dependability attributes can meaningfully relate to the stakeholders' interests.

6.5.1.2 Argument from the Perspective of Dependability Attributes

Identifying the attributes of interest entails stakeholders examining the operational objectives (captured in context *TopClaimC1*) from the perspective of each dependability attribute. Definition of the system objectives is one of the initial stages of the DDA process.

During DDA typical issues pertinent to the dependability attribute under which the system objectives are examined can reveal the primary concerns for the system stakeholders. A concern is an issue that can credibly affect the system objectives and hence the overall acceptable operation of the system.

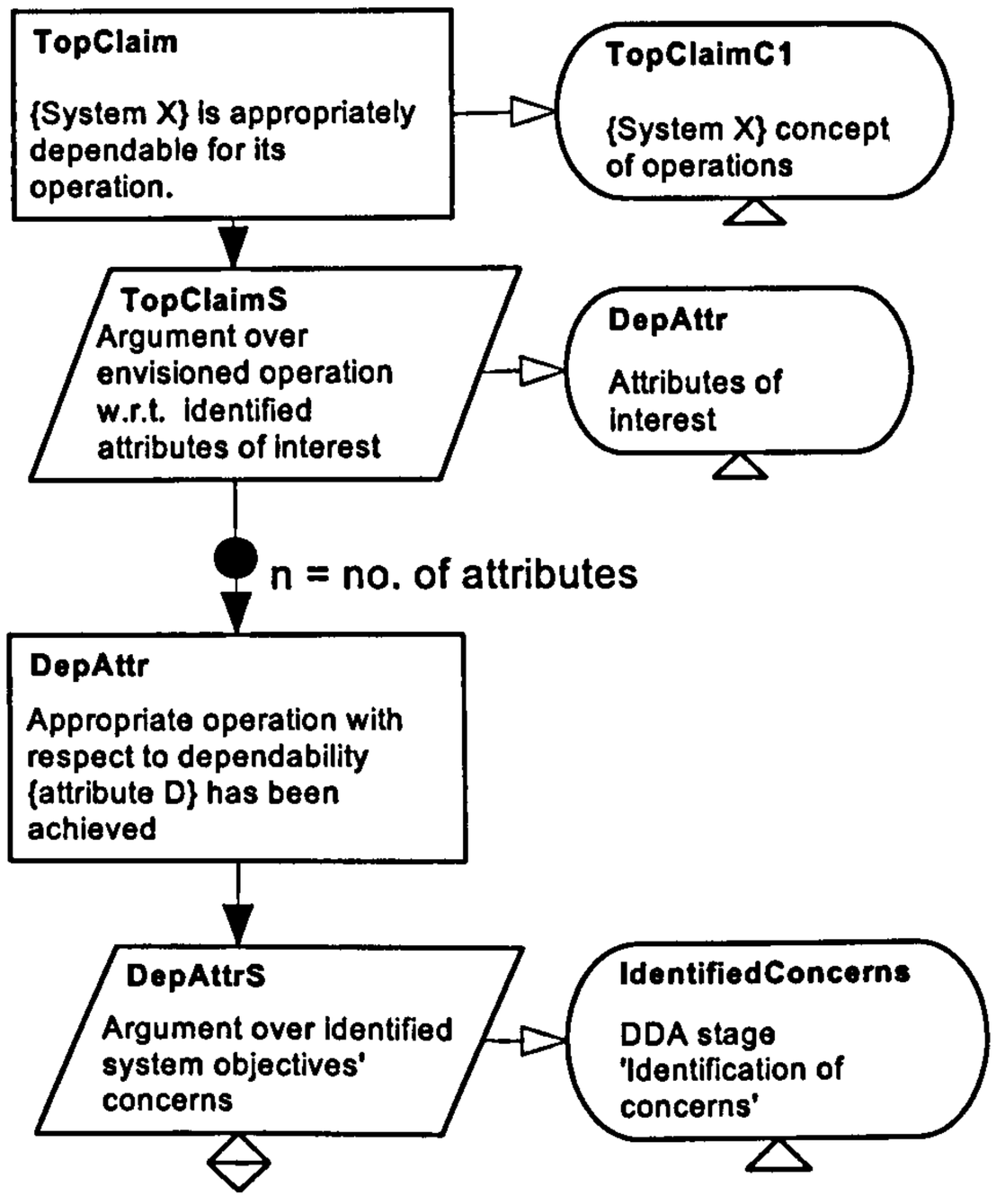

Fig.6.11 – Attributes of Interest in the High Level Module

Fig.6.11 illustrates how the top level goal can be decomposed to capture the attributes of interest to the stakeholders. *TopClaimS* is the strategy based on which the top goal is substantiated, requiring identification of the attributes of interest to the stakeholders that will contribute to achieving dependable operation. The strategy is solved by goal *DepAttr* which is instantiated for each identified dependability attribute of interest and claims acceptable achievement of that attribute. The strategy (DepAttrS) for solving goal *DepAttr* is to argue over the concerns (identified during DDA) that may compromise the attribute of interest. Use of concerns and DDA to support goal *DepAttr* is discussed in the following section.

6.5.1.3 Arguing Acceptable Operation Regarding Dependability Concerns

Development of the goals referring to the attributes of interest is achieved in substantiating goals addressing the stakeholders' concerns. Analysis of the system during the Dependability Deviation Analysis identifies how deviating from normal operation with respect to an attribute can compromise the operation of the system. Goal *DepAttr* is substantiated by arguing over the concerns identified by the stakeholders during DDA. Fig.6.12 presents the resulting argument.

Goal *ConcAttr* supports strategy *DepAttrS* by claiming that the concerns that may compromise the attributes of interest have been addressed. DDA provides the necessary information for claims *ConcEffect* and *DepConcOp*, supporting the *ConcAttr* goal which was identified earlier. *ConcEffect* claims identification and understanding of the concern's effect on the system operation. This takes place during the initial stages of DDA, in which the top level objectives are examined. *DepConcOp* argues that the operation of the system with respect to the concern is acceptable. *DepConcOp* is defined in context of the elicited target and limit, which define the bounds of acceptability. Also, *DepConcOp* is stated in the context of the claim that the bounds of acceptability have been justified. This claim is associated with the argument produced by TOM stage one, which captures the rationale behind the determination of limits.

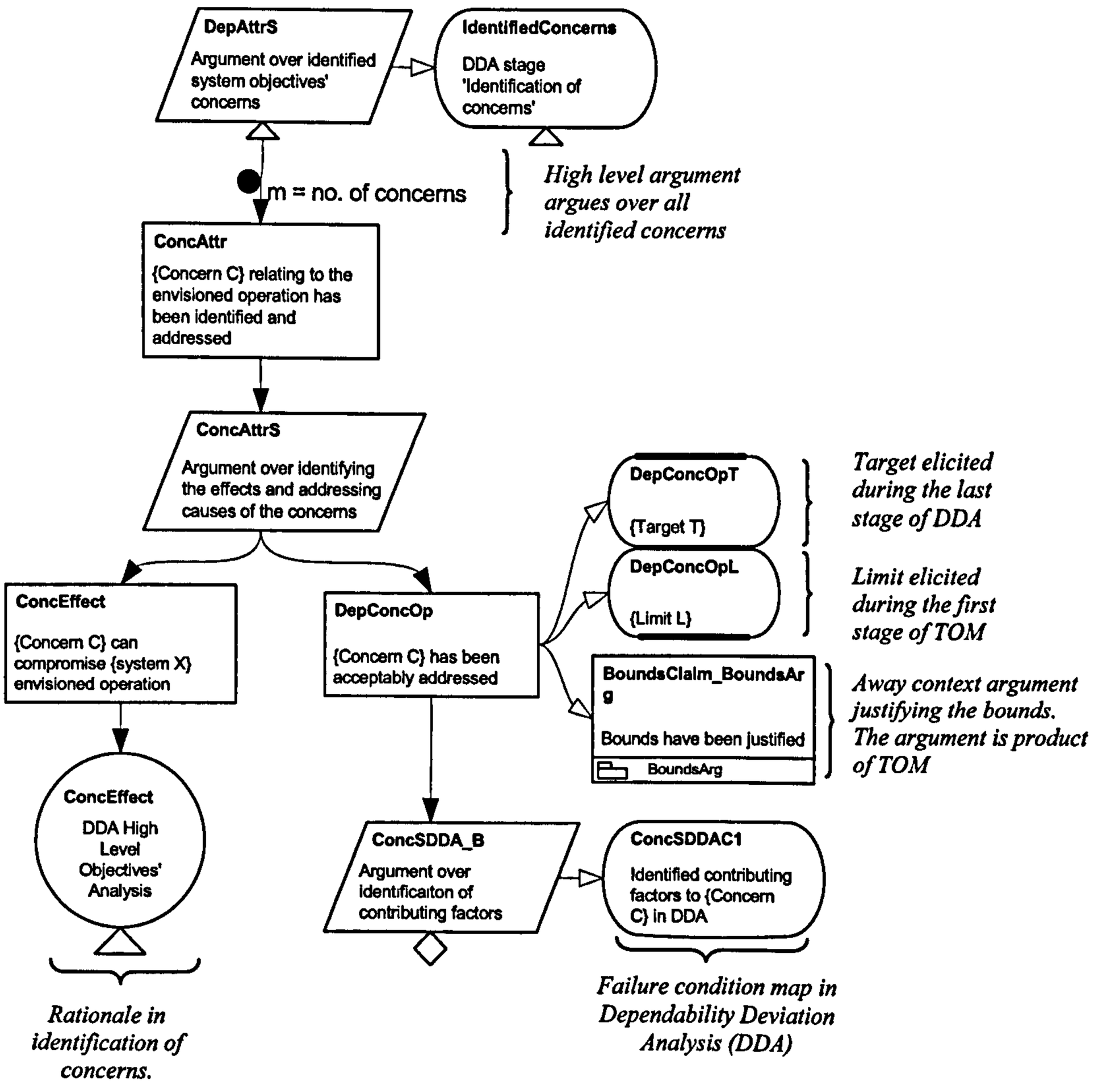

Fig.6.12 – Arguing About the Concerns

Further development of the argument requires identifying how a single failure condition or combination of failure conditions, can result in concerns that will compromise the dependable operation of the system (strategy *ConcSDDA_B*). This is stated in the context of the failure conditions map identified during DDA (*ConcSDDAC1*), which provides a 'causal picture' of which failure conditions can result in a concern. Understanding the failure conditions that can cause the concern being argued allows further decomposition of the argument under strategy *ConcSDDA_B*. As explained there are interrelations between failure conditions which were depicted using the failure maps. Use of case studies by the author indicated that it is difficult to decompose *ConcSDDA_B* with goals relating to each of the failure conditions that may cause the concern; this would result in circular references and a great number of associations between goals, which would make the dependability case difficult to manage. The use of the dependability profile is used in this research as a means of structuring the dependability case.

6.5.2 Dependability Profiles Supporting the High Level Dependability Argument

The high level dependability argument's objective is to capture and argue about the overall required operation of the system. Hence, the high level dependability argument primarily focuses on the overall system requirements stated in context of the envisioned operation. However arguments claiming satisfaction of the high level requirements needs to be developed in context of a particular system design. The dependability profile is a concept introduced to facilitate traceability between the high level dependability argument module and argument modules developed in the context of the system. The dependability profile is created during Dependability Deviation Analysis (DDA) and describes acceptable behaviour of system elements that are modelled (in the modelling framework that has been selected) with respect to various dependability attributes. Each profile is a collation of requirements elicited from the perspective of each attribute of interest, *regarding a particular system element.*

As described the strategy to argue over contributing factors *ConcSDDA_B* (Fig.6.12) introduces certain problems in the design, resulting in a complicated and difficult to manage dependability case. However, information such as the effects and consequences of the various failure conditions does not need to be part of the argument, but

constitutes the context in which the argument is created. An alternative (to *ConcSDDA_B*) strategy (recommended by this thesis) would be to argue over achievement of requirements that were elicited in the context of the identified failure conditions. This way instead of supporting a goal about a concern (goal *DepConcOp*) by arguing over the failure conditions, the argument will be made by claiming achievement of the requirements that were elicited given the identified failures. For example, in the case of the AGO (discussed in §6.5.1) claiming achievement of a safety concern would need an argument over the reliability levels of the target data provision function.

Fig.6.13 illustrates a refactored high level dependability argument, substituting strategy *ConcSDDA_B* with a strategy (*DepConcOpS*) supported by an argument based upon satisfaction of the dependability profiles of the system elements. This is the approach favoured by this thesis.

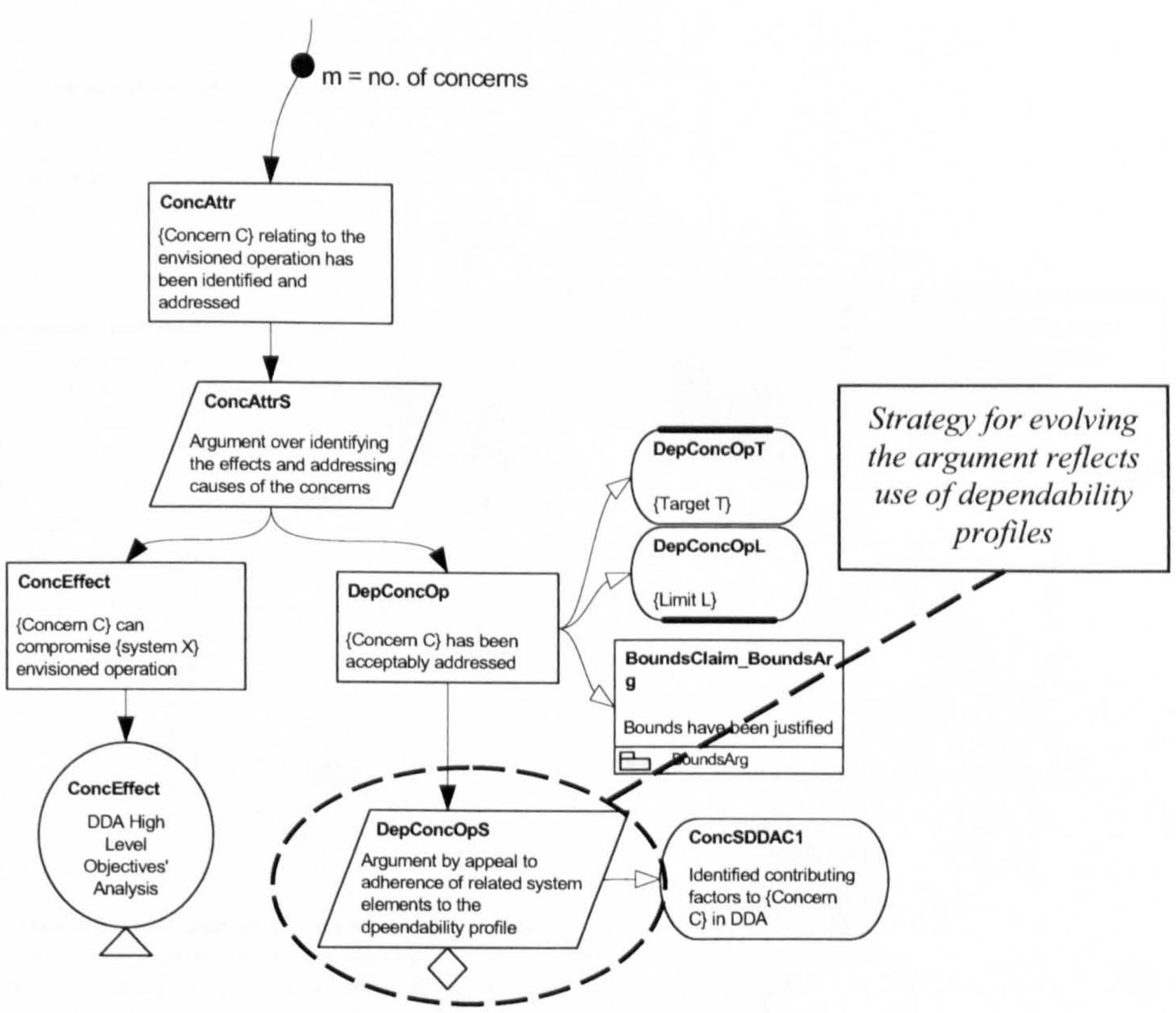

Fig.6.13 – Strategy by Appeal to the Dependability Profile

In order to create a clear and manageable case, the dependability profiles associated with the concern are not directly associated with *DepConcOpS* since this would result in separate arguments for each concern of each attribute of interest. Instead, the arguments referring to the dependability profiles are contained in separate modules, which support the high level argument. Linking argument modules requires recording the interfaces between the arguments in an understandable manner, maintaining traceability and clarity of the argument. GSN contracts are employed to compose the dependability case out the identified individual arguments.

6.5.3 Use of GSN Contracts to Structure the Dependability Case

Using GSN, a case can be partitioned into modules, each of which constitutes an individual argument. The overall case is defined by aggregating and documenting the relationships between the participating modules [100]. Fig.6.14 shows how contracts can be used to structure a case. The top level module contains the overall claims about the system.

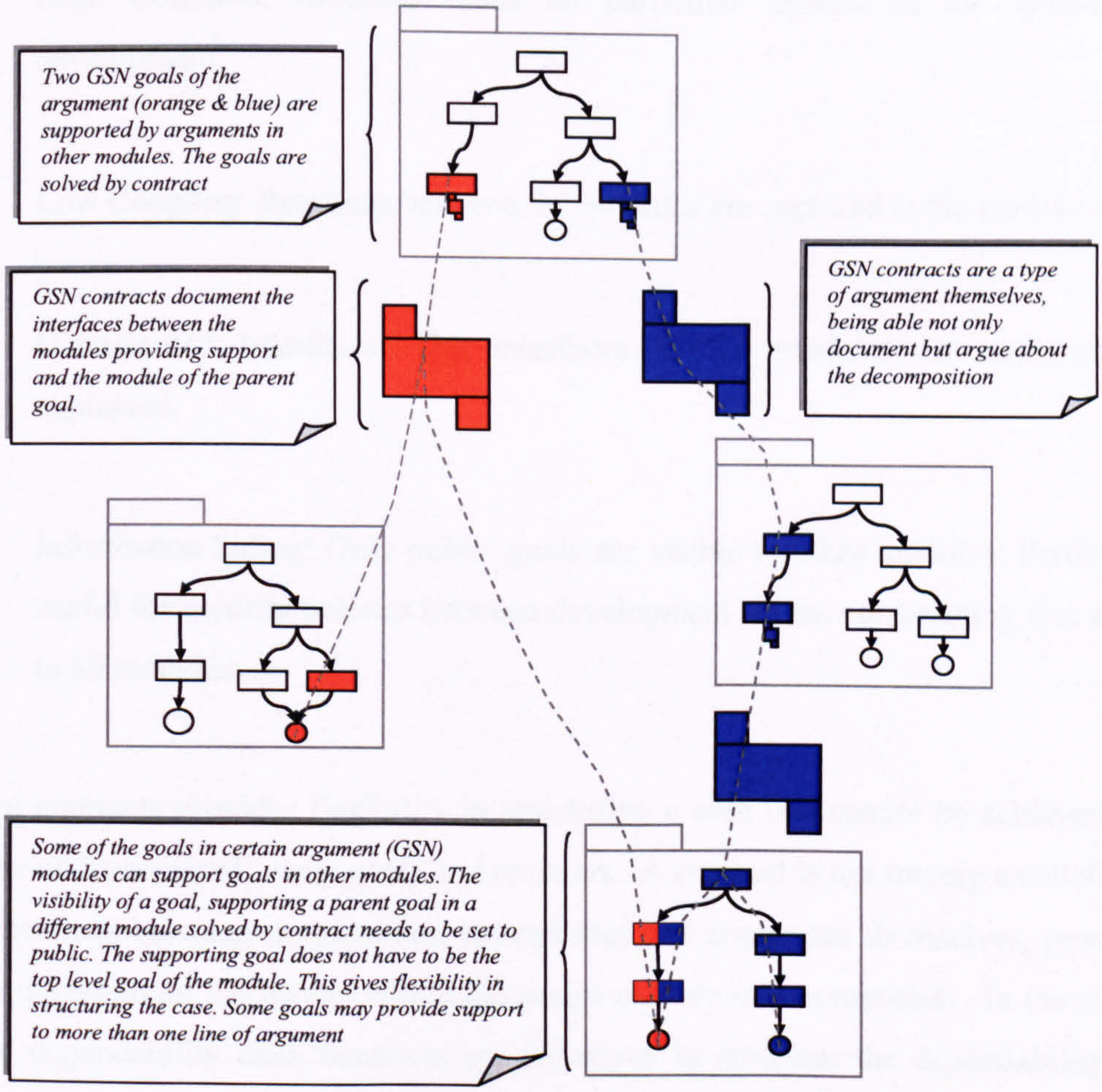

Fig.6.14 – Use of GSN Contracts in Structuring a Case

For example, overall claims about the achieved safety levels of the system (safety), or the dependability behaviour of the system (dependability case). Development of the argument contained in the top level module can result into goals that are supported by arguments in other modules (orange and blue goals).

The two goals are 'developed by contract', which captures the association of the parent goal with the modules supporting it. Contrary to 'away goals', a goal developed by contract does not necessarily have to be associated with the top goal of an argument module, but it can be supported by any *public* goal in the module (a public goal in GSN is a goal visible outside its container module). Moreover goals within an argument module can provide support to more than one contract; in Fig.6.14 the bottom module provides support to two contracts. Use of GSN contracts provides the following benefits:

- High Cohesion: Modules focus on particular aspects of the system-case development.
- Low Coupling: Relations between the modules are captured at the module level.
- Documented Interfaces: The interfaces of the modules are captured and explained.
- Information hiding: Only public goals are visible to other modules. Particularly useful for security policies between development teams, constraining free access to all modules.

Use of contracts provides flexibility in structuring a case that cannot be achieved with the use of 'hard-wired' away goals and contexts. A contract is not merely a collation of the interfaces between the modules; instead they are arguments themselves, providing an argument about the way in which the parent module is decomposed. In the context of the dependability case, contracts are employed to structure the dependability case capturing the interfaces between the modules, and arguing about how the arguments

regarding the achievement of the dependability profiles can support the dependability goals of the high level dependability argument. Fig.6.15 presents the resulting architecture of the dependability case, proposed in this thesis, using contracts in modular GSN.

In this thesis, we structure the dependability case of a system based on the models that represent it (the system). In particular we align the dependability case with MODAF products. In general, defence architectural frameworks are optimised to Systems of Systems (SoS). As discussed in chapters 2 and 4, the MoD and DoD specification do not provide a definite way for developing the products. However, the DoD deskbook provides guidelines for a data-centric approach, called the activity Based Methodology (ABM) [23] presents the sequence in which the products are constructed following the ABM.

ABM is used to maintain consistency between the DAF products, creating integrated architectures by adding detail to existing models, the elements of which can be mapped onto the new models. An important aspect of the ABM methodology is that during the evolution of the system, the developers follow a set of well specified steps incorporating traceability during development of the models. Addition of detail in each step, results to the evolution of the overall system by defining new models. DAF models using the ABM, define a hierarchy that can be used to trace dependability considerations from the high level concept of operations (CONOPS), to individual systems and functions that constitute the operation of the system.

The dependability case architecture proposed in thesis is organised around the hierarchy presented in Fig.B.2. Fig.6.15 illustrates the resulting architecture of the dependability case employing GSN contracts for its composition. The case consists of four different types of argument modules:

- *High level dependability argument*: High level dependability argument: Argument about the satisfaction of the high level dependability requirements, of primary interest to the system stakeholder.

- *Operational view product arguments*: Arguments about the satisfaction of dependability requirements of each operational level product. These requirements are derived according to how the operation of the SoS may affect the dependability attributes of interest to the stakeholders.

- *System view product arguments*: Arguments about the satisfaction of dependability requirements of each system level product. These requirements are derived according to how the systems may affect the operation of the SoS.

- *Trade-off arguments*: The trade-off arguments, product of TOM, constitute context to the dependability case, providing an argument that the selected alternative is the most suitable.

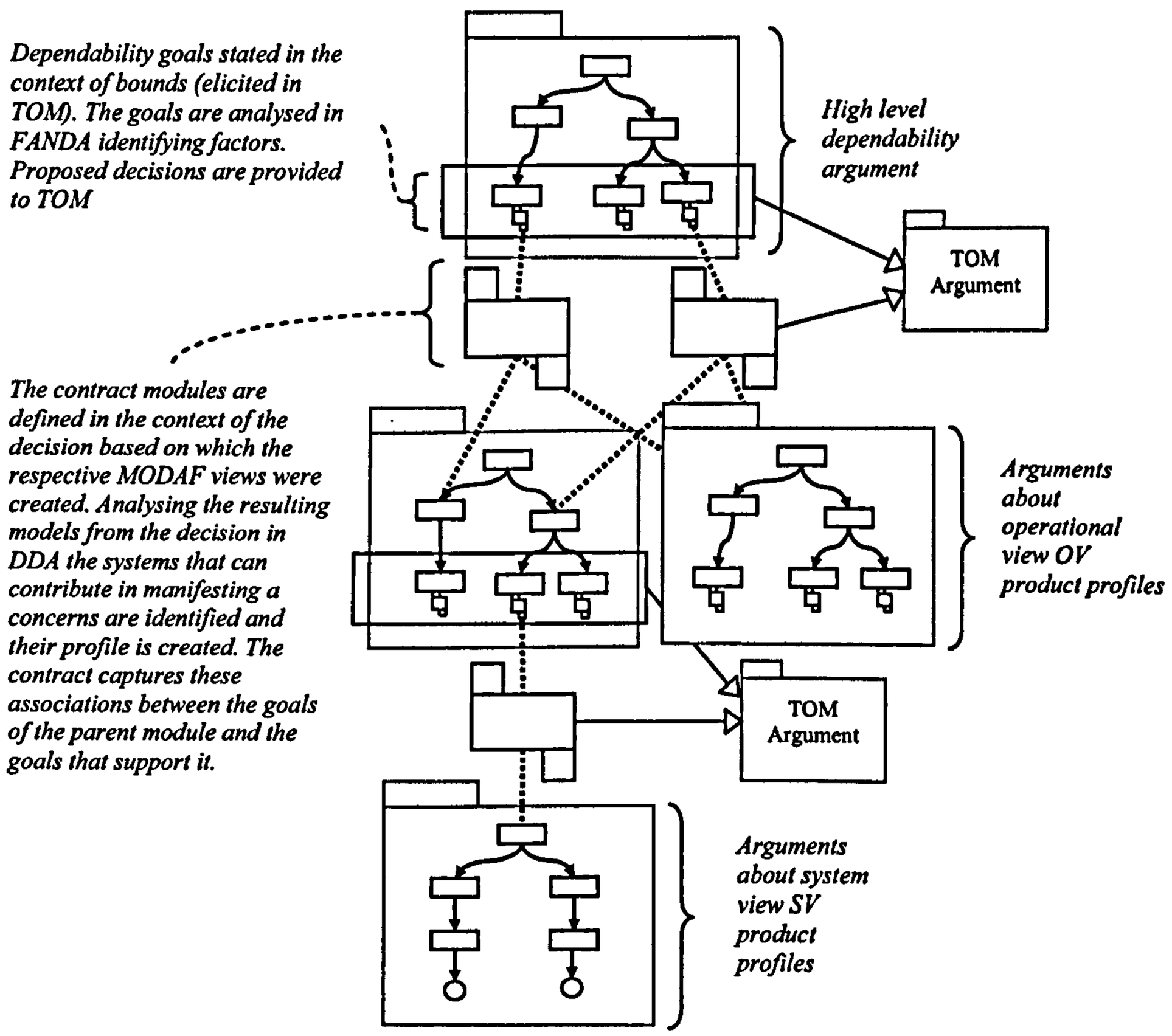

Fig.6.15 – MODAF Centric Dependability Case Architecture

The high level dependability focuses on the dependability requirements of the system stakeholders. The leaf goals of the module are stated in the context of bounds and their justification argument. Justification of the bounds is a part of the trade-off method argument. Using GSN contracts the leaf goals of the high level argument are supported by arguments regarding the dependability profiles of each of the (MODAF) operational view products, which in their turn are supported by arguments about the dependability profiles of each of the systems view products.

6.5.3.1 Refactoring the High Level Argument to Use Contracts

As illustrated in Fig.6.13 the strategy followed in the high level dependability argument was to argue over addressing the concerns that were identified (during DDA) to be able to compromise the envisioned operation of the system. Fig.6.16 shows the leaf goals of the high level argument refactored to reflect the use of contracts.

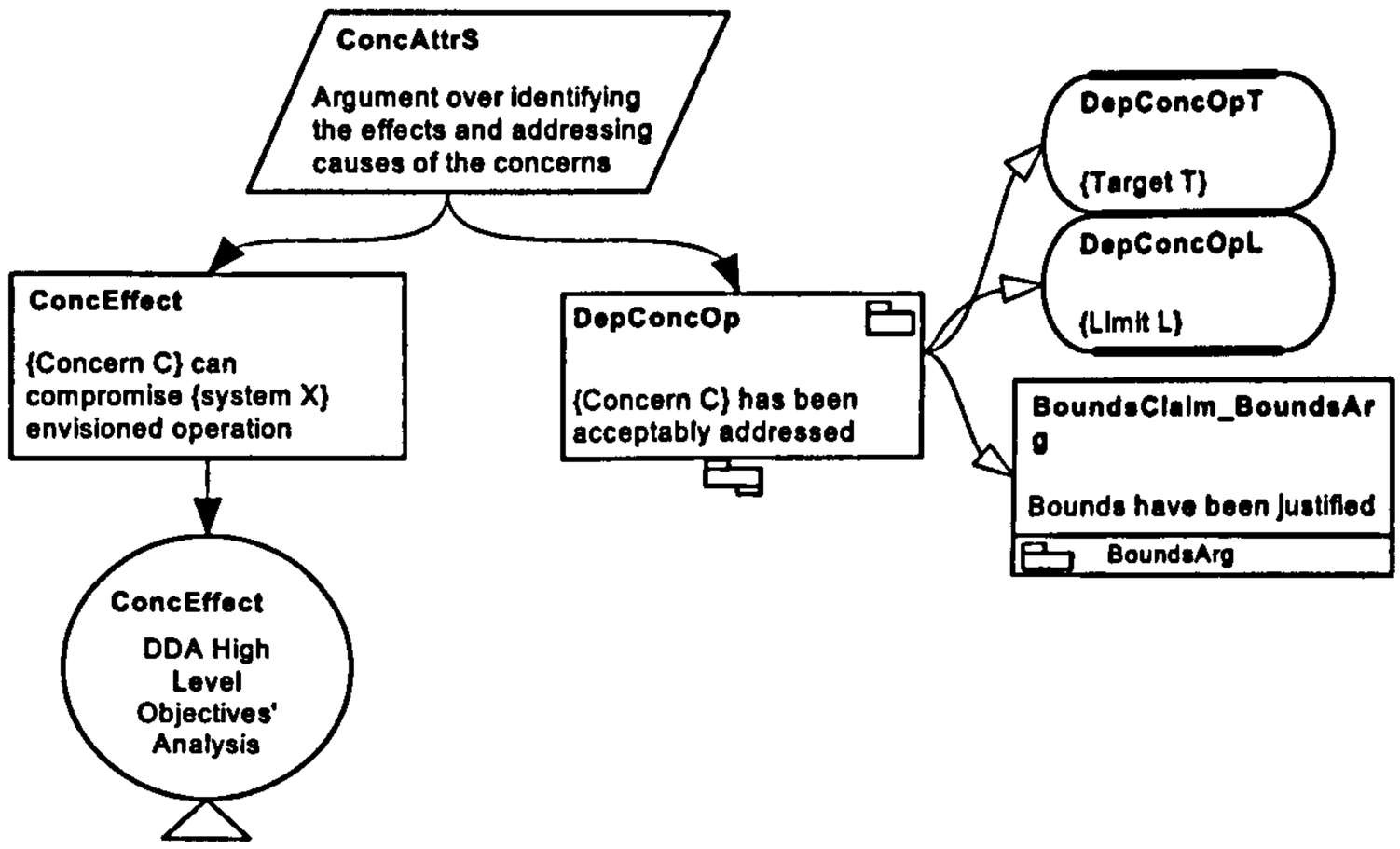

Fig.6.16 – High Level Argument Using GSN Contracts

Using the GSN contracts the high level argument focuses purely on addressing the overall dependability goals of the stakeholders, as identified by the DDA. Details of how the case is structured are now not a part of the high level argument. In Fig.6.13 the *DepConcOp* was substantiated using a strategy to argue over adherence to the appropriate dependability profiles (*DepConcOpS).* The strategy was stated in the context *ConcsDDAC1* which references the product of the DDA that shows the dependability profiles related to a particular concern (i.e. failures map).

Information regarding how the dependability case modules support the instances of goal *DepConcOp* is now part of the contract module. Fig.6.17 illustrates how the developed by contract goals (*DepConcOp)* are associated with their respective contract (*DepConcOpContract*).

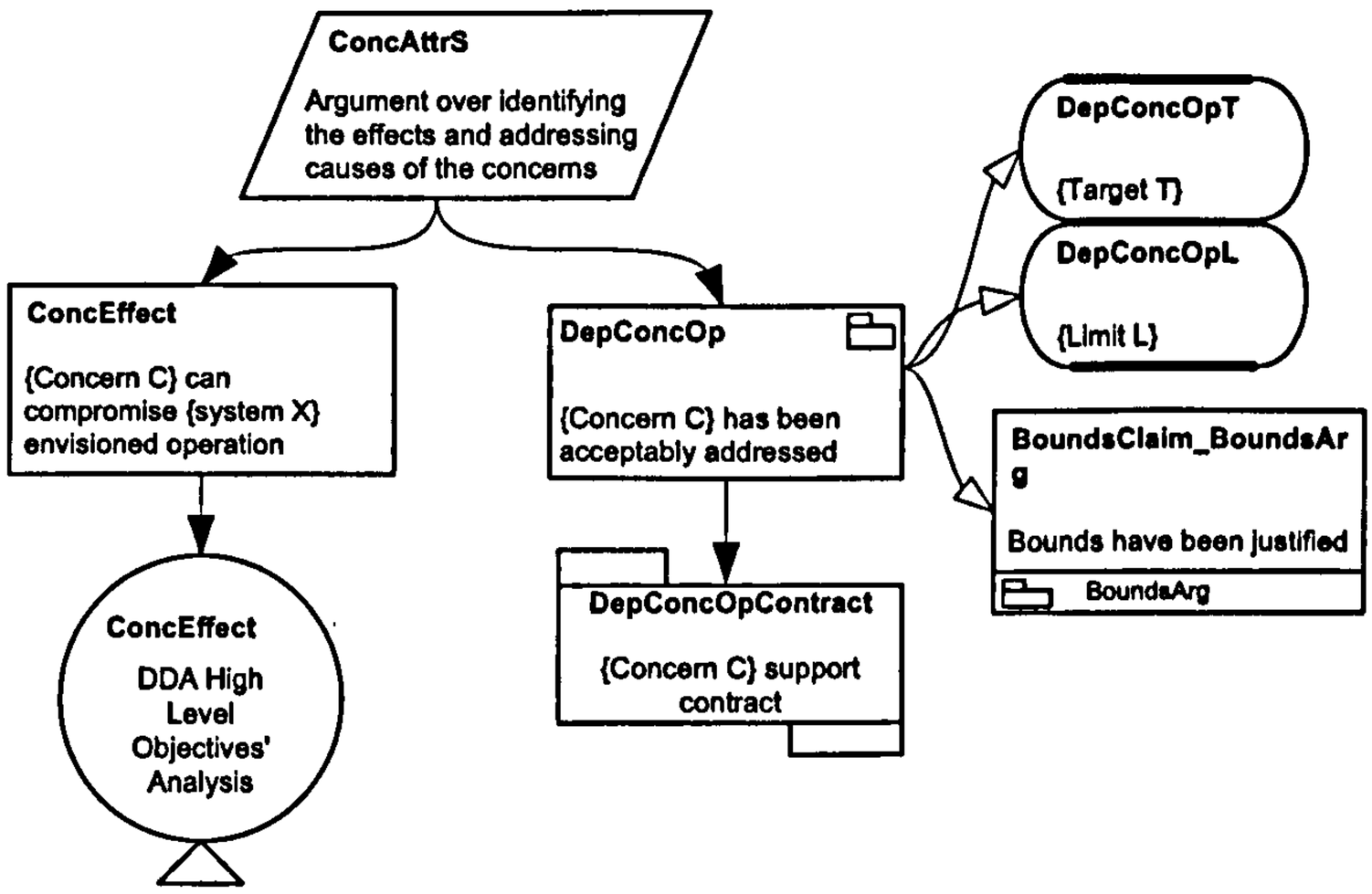

Fig.6.17 – High Level Argument Linking to GSN Contract

6.5.3.2 Contract Module Supporting the High Level Argument

Using modular GSN and in particular GSN contracts, information regarding the partitioning of the dependability case is not part of the high level dependability argument. As described a contract is an argument module itself, which can capture the rationale and justification for the particular dependability case layout.

Fig.6.18 illustrates a pattern for contract modules representing the approach adopted in this thesis. *DepConcOp_HighLevelArgument* is the goal in the contract module that corresponds to the (leaf) 'developed by contract' goal *DepConcOp* in the high level dependability argument.

The goal is substantiated by strategy *DepConcContrS.* The strategy states that the argument will be developed, by arguing over meeting the requirements defined in the profile of the system elements, which have been identified as contributing to concern C.

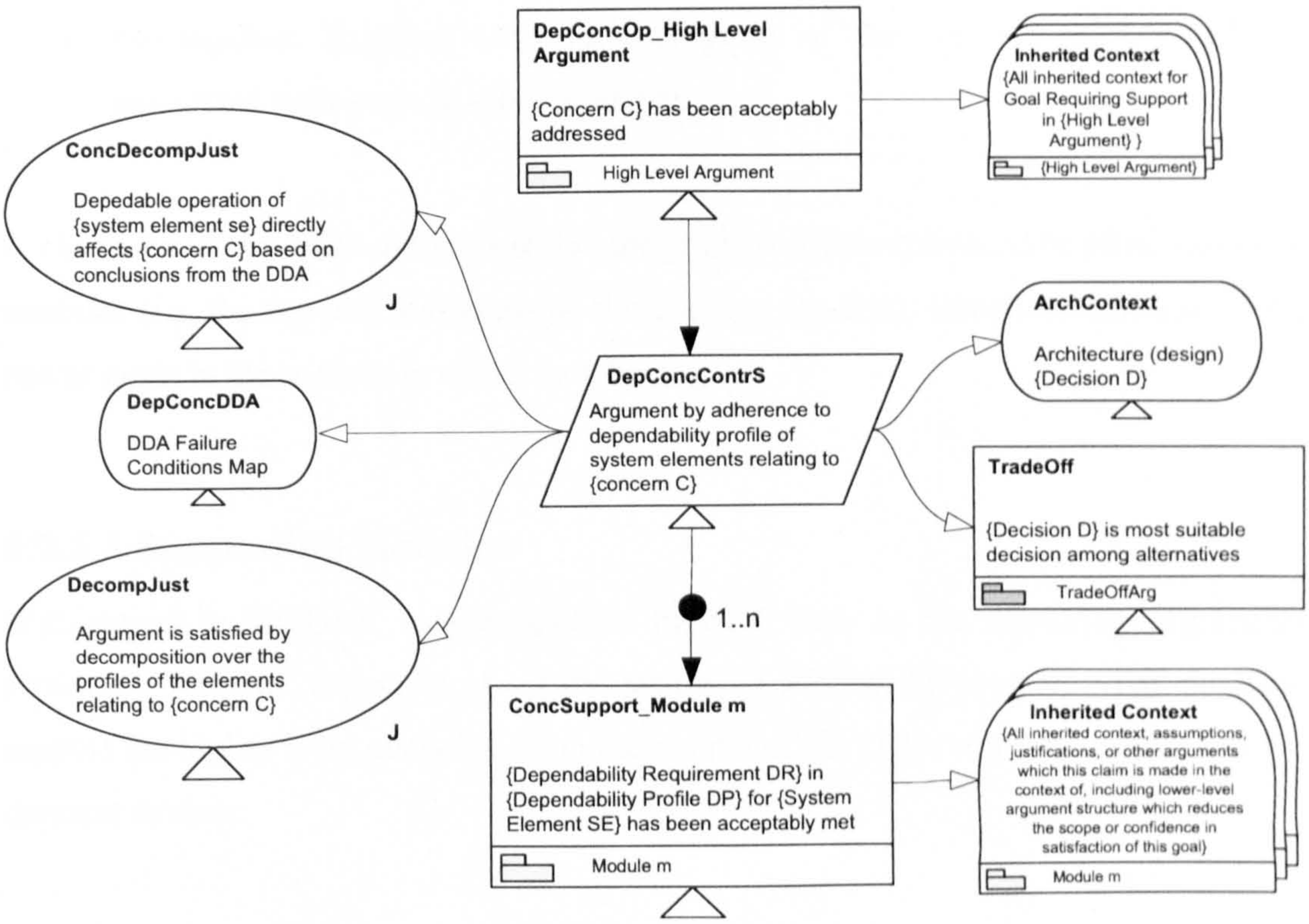

Fig.6.18 – Dependability Case Contract

The strategy *DepConcContrS* is stated in the context of the following items:

- *DepConcDDA*: Refers to the DDA failure map that shows the failure conditions of which system elements can contribute to concern C.

- *ArchContext*: Specifies the design decision which resulted in the definition of the model elements that were analysed in DDA, resulting in the failure map referenced from *DepConcDDA*.

- *TradeOff*: This is a reference to the *argument* resulted from TOM justifying that the architectural (or design) decision stated in *ArchContext* is the most suitable among the candidate alternatives.

- *ConcDecompJust*: Explains how concerns are related with the system elements based on the results of the DDA.

- *DecompJust*: Explains how the leaf goals of the contract module will be associated with goals in other modules.

A characteristic of GSN contracts is that the goals that are referenced in other argument modules (i.e. the top and leaf goals of the contract module), inherit the context of the parent goals in the module in which they belong.

6.5.3.3 Supporting Modules

With regard to MODAF, as presented in Fig.6.15 each of the supporting arguments serves as 'wrapper' argument about the profile of a MODAF product. The modules support the higher level goals based on the results of the DDA analysis, justified in the contract module.

Fig.6.19 shows the pattern of a supporting module. The top level goal of the argument *MODAFProdProf* claims that the profiles of the system elements related to the product have been met. The goal is decomposed to goals regarding each dependability requirement of each system element associated with the MODAF product. The goals resulting from this decomposition (*RQSEMet)* are the public goals referenced from the contract.

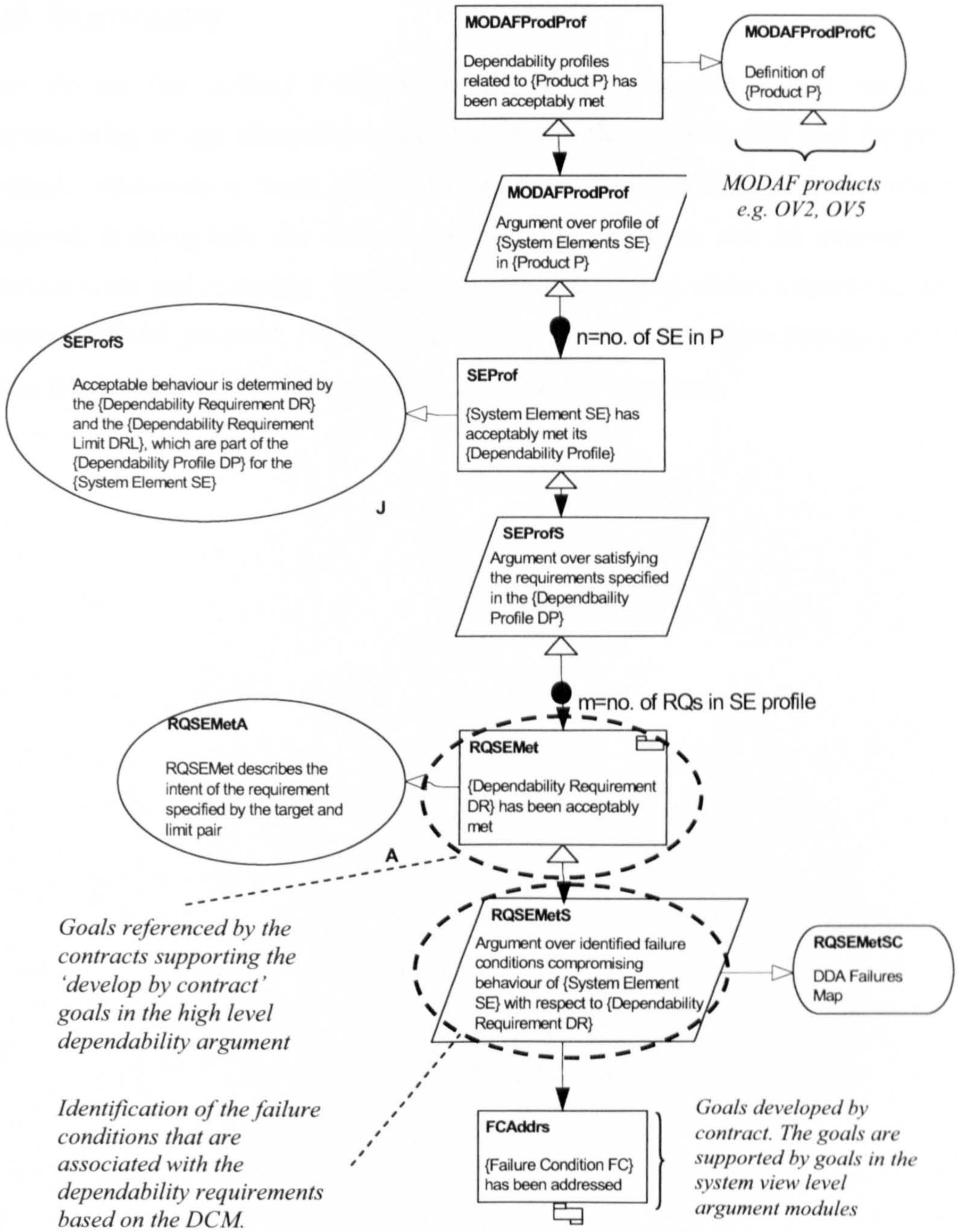

Fig.6.19 – MODAF Product Arguments

The argument is further developed in the context of the DDA analysis, according to which the failure conditions of other system elements can compromise the requirement claimed by *RQSEMet.* Addressing the identified failure conditions (goal *FCAddrs*) requires references to other modules, which are implemented using contracts.

6.6 Summary

This chapter has defined FANDA, a method for capturing design rationale and brainstorming design alternatives. In addition the chapter describes how the proposed methods collaborate to create a dependability case. A dependability case architecture is proposed defining how the overall dependability argument can be structured using modular GSN and contracts. Use of dependability profiles allows structuring the case around MODAF products. Finally, the chapter presents how the information collected using DDA and TOM are related to the dependability argument.

Intentionally Blank

Chapter 7

Evaluation

This chapter describes the means by which the work presented in thesis was evaluated against the thesis proposition, throughout the duration of the research. The thesis proposition was stated in the following terms:

"This thesis demonstrates that it is feasible to establish a structured approach to evolving and presenting a dependability case for Systems of Systems through a unified approach to eliciting flexible dependability requirements, facilitating resolution of trade-offs between competing objectives, and combining and managing these activities using structured argumentation."

As discussed in Chapter 1, the proposition implies the following challenges:

- Understanding relationships between dependability requirements
- Analysis of the design from the viewpoints of the dependability attributes
- Elicitation of acceptable dependability requirements
- Design rationale and identification of resultant trade-offs
- Management and facilitation of trade-offs
- Evolution of case in parallel to the design
- Composition and architecture of the dependability case
- Traceability between all the levels of the dependability case

The challenges are addressed by three distinct strands within the research, namely:

- Dependability requirements elicitation using deviation analysis, described in Chapter 3
- Argument-based resolution and management of trade-offs, described in Chapter 4
- Dependability case evolution and architecture, described in Chapter 5

The evaluation of the proposition was focused on examining the application of the framework with respect to the following concerns:

1. Feasibility of the proposition – examining the resulting dependability case structure after application of the methodologies
2. Benefit in applying the methodologies in order to address the challenges inherent in the proposition

7.1 Means of Evaluation

Various means of evaluation were employed during the different stages of the research. These included the following:

- Use of simple examples and anti-examples
- Peer review
- Formalisation and tool support
- Case studies

7.1.1 Examples

Examples were the simplest means of evaluating the concepts of the thesis. They were a means of quickly testing newly introduced concepts within the framework. Initial acceptance or rejection of concept was based on proof of concept (using an example) or identification of an 'anti-example' demonstrating inefficiency of the concept. Larger scale evaluation was also required for evaluation of the overall resultant framework.

7.1.2 Peer Review

Design and proposal of a framework is an inherently difficult subject to evaluate, since it is typically infeasible to obtain a statistically significant sample of case studies. Furthermore, the domain of application of the research is characteristically conservative

in adopting and evaluating new methodologies. For these reasons, the need for evaluation by means of peer review was highlighted at an early stage of the research. The author used all available opportunities for peer review from highly experienced persons who were professionals in domains related to high integrity systems and requirements engineering. Having been based extensively on the concept of safety cases, the input of experiences and practices from reviewers related to the domain had a significant impact on the evolution of the work. Reviews included comments on the feasibility of the methodologies, as well as the potential benefit gained through application of the framework. Description of the dependability case framework was communicated using examples and worked through case studies.

7.1.2.1 Peer Review through Publications

Most of the material in this thesis has been presented in international conferences and workshops. Safety and requirements engineering conferences were targeted with the aim to subject the research to opinions from professionals in as closely related domains as possible. In general there was positive feedback relating to the work. Although this is the weakest form of peer review, positive feedback was consistently received during different conferences, giving an indication of acceptance of the work. An exception in which the work was subjected to more scrutiny was the presentation of a paper in the 24th International System Safety Conference. The paper, describing FANDA and TOM and their use in the dependability case framework, received the best paper award after being evaluated by a panel of professionals and academics affiliated to the System Safety Society. Although the methodologies were not investigated by the panel in depth it was considered a vote of confidence regarding the benefit from potential application of the methodologies. Furthermore, the overall framework was presented at the 1st Object Management Group (OMG) Workshop on Software Assurance Cases – a workshop with the declared aim of standardising the structure and presentation of assurance cases. The metamodel presented in this thesis addresses some of the issues identified by the workshop panel. Finally, some of the work has been cited in a number of papers [101], [102], [103]; most notably, is the reference of all the methodologies that constitute the dependability case framework, in a report issued by the US homeland security regarding good practice in software assurance [104].

7.1.2.2 Peer Review within the HIRTS Defence and Aerospace Research Partnership

This research is carried out under the High Integrity Real Time Systems Defence and Aerospace Research Partnership (HIRTS DARP), funded by the MoD, DTI and EPSRC. The members of the HIRTS DARP are BAE SYSTEMS, Rolls-Royce plc, QinetiQ and the University of York.

As part of the partnership the results of the research was frequently presented to the industrial partners during two types of events; quarterly working group meetings and yearly DARP workshops. The former type of event consisted of persons with particular interest and experience to the strand of research regarding dependability cases and SoS. The latter involved a broader audience interested in the work, not limited to the DARP industrial partners.

Quarterly working group meetings constituted a very effective means of evaluation and provision of feedback. The meetings included review of examples and case studies, and presentation of the methodologies and their underlying theories. Discussions triggered by the presentations were held throughout the meetings, examining the qualities of the methodologies as well as their feasibility and potential benefit. Experience and close relationship of participants to the subject, as well as in depth discussions made the DARP working group meetings a strong means of evaluation. Feedback collected in the meetings significantly influenced the work.

DARP workshops provided an opportunity to present the work to a broader audience interested in the topic. Compared to the working group meetings, DARP workshops did not provide as much substantial feedback. The dependability case framework and its associated methodologies were not examined in the same detail. In general the work received positive feedback and some comments were further discussed during the working group meetings.

7.1.2.3 Peer Review during Case Studies

This type of peer review encompasses discussions taking place prior and during application of case studies, in order to familiarise persons related to the case studies

with the framework. This form of evaluation was useful in gaining the experience of those other than the thesis author in their first-hand application of the methods. In brief, the majority of the received comments reflected recognition of benefit from the application of the methods.

7.1.3 Formalisation of the Framework and Tool Support

Formalisation of the dependability case framework involved creating a metamodel that captured concepts and interrelationships required to support the proposed methods. Further support was provided to the metamodel with scripts that performed a range of functions: These included functions to perform validation of instances of the metamodel, production of graphs, as well as automatic labelling and creation of objects where applicable. Although there is not a properly developed tool encompassing the dependability case framework, the eclipse model editor and the scripts written provide some fundamental computer aided support for someone wishing to create a dependability case based on the proposed framework.

The metamodel classes and their related attributes define the essential concepts of dependability cases. Associations between the metamodel classes help understand the relationships between the methods described in the thesis. Definition of a metamodel requires a high degree of rigour. Existence of a rigorous metamodel that captures the concepts of the assurance cases domain can deliver a number of benefits. Most importantly, it offers a common vocabulary and consensus on the concepts. Furthermore, satisfactory use of the scripts offers some evidence of the correctness, coherence, sufficiency and detail in the implementation of the metamodel.

7.1.4 Case Studies

Evaluation by means of application on case studies was necessary in order to evaluate the research framework. The methodologies were applied to three case studies:

1. *The Anti-Guerrilla Operations (AGO)* scenario (presented in appendix B). This is a fictional fully developed case study created by the author for the purposes of the research. Although fictitious, the case study was considered to be realistic

enough for its use. This claim was also supported by the DARP partners as well as by feedback received during DARP workshops. It was substantially based on input from industry, as well as examples widely available from sources such as the US Department of Defence. The AGO case study was the main case study used to communicate the results of the research for peer review during working group meetings and workshops.

2. *Network Centric Warfare (NCW)* case study. The second case study was provided by BAE Systems, one of the DARP partners. Due to the sensitive nature of the case study, details of the underlying scenario, models and results cannot be revealed. The NCW case study was defined in the spirit of a typical NCW example, and has very similar qualities with the AGO scenario. Application of the methodologies on the NCW case study was particularly useful. Compared to the AGO example, this was a 'real' example which demonstrated the readiness of the research to be applied in industry. Furthermore, the particular case study evaluated the framework in terms of scalability. Being a real scenario, the case study included numerous platforms collaborating and exchanging information. Through the course of the case study revisions of the methods took place.

3. *High Altitude Platforms (HAPs)*. This is the third case study, developed in collaboration with the Department of Electronics of the University of York. Typically, a HAP is an airship or an endurance aircraft capable of flying at an altitude of around 20km. This is above any current normal aircraft but significantly below stratospheric satellites. The case study was initiated during the latter stages of the research. The main motivation was to establish an overall dependability case about the use of the HAP and the accompanying systems required for the required scenarios. The methodologies presented in the thesis have been presented and explained to the HAP stakeholders. The HAP case study has been a very effective means of supporting the evaluation, because it is developed in collaboration with individuals with no safety engineering background. This allowed the author to examine how easily the concepts can be understood and applied by someone unfamiliar to safety cases – a founding concept underlying the approach presented in this thesis. The case study offered

evidence for the clarity of the concepts as well as the sufficiency of detail of the methods. Moreover, the case study offered a platform of evaluation which demonstrated all the characteristics of a typical System of Systems, whilst being considerably different to the Network Centric Warfare examples.

7.2 Evaluation of the Contributions

In this section the contributions are examined individually. The discussion that follows is based on feedback received from the application of the means of evaluation as described in section 2.

7.2.1 Evaluation of Dependability Deviation Analysis (Chapter 4)

Dependability Deviation Analysis advocates a philosophy of examining potential deviations from the perspective of each dependability attribute. Furthermore it introduces the concept of the dependability profile, which is used as a 'stepping stone' for the specification of acceptable dependability contracts used in the assurance process (construction of the case). Application of the methodology results in identification of credible deviations and their resultant failure conditions that can compromise the stakeholders' objectives, as well as a dependability profile for each of the system model elements that were analysed. DDA can also be graphically represented in a *failures map* which is automatically generated from the metamodel instance.

An important aspect of DDA is the definition of concepts common to all dependability attributes, namely *issues, concerns, failure conditions* and *system element*. The purpose of these concepts was to provide generalisations for concepts specific to individual attributes. Furthermore their associations were defined as part of the metamodel. The representation of the concepts of *failure conditions* and *system elements* were swiftly understood when presented to other individuals as part of working group meetings or during case studies. Comparatively, the concepts of *issues* and *concerns* proved to be more challenging to explain. These two concepts are more abstract than their equivalent attribute specific concepts such as security vulnerabilities and security breaches.

However, upon further elaboration, the differences as well as the purpose of each of the two concepts was recognised. Separation of concerns and issues makes a distinction between the ultimate interests of the stakeholders regarding overall system operation (i.e. concerns), and the properties of the system design that could lead to these concerns. Comments received also suggested the specification of templates of typical issues that could be defined and reused during the analysis of similar systems. The usefulness of this was explicitly highlighted during the HAP case study, in which the set of concerns incorporated communication specific issues.

All three case studies have extensively exercised the DDA method. DDA has proved to be very efficient in identifying the relationships between failure conditions. Application of deviations that were optimised to reveal issues particular to one dependability attribute, resulted in the wider identification of failure conditions that compromised other attributes. Identified failure conditions can directly compromise the stakeholders' concerns, or they can be the cause of another failure condition already been identified. Comments received through peer review were positive, mostly concentrating on the usefulness of the failures map to capture and clearly record failures from multiple dependability viewpoints. Often the failures map revealed associations between dependability failures that otherwise would have been difficult to conceive.

Dependability profiles capture the specification of behaviour required from a system element in order for the system as a whole to operate dependably. Dependability profiles were introduced in the framework as a means of facilitating partitioning of the argument modules of the dependability case, using GSN contracts. Dependability profiles can also potentially be used to aggregate specifications that were elicited with methodologies other than DDA. Finally, the dependability profile helped specifying the bounds of the goal.

An important characteristic of the DDA method is that it is model independent. DDA has not been defined for use with a particular modelling framework. Instead, DDA is applied on, what is described in the metamodel as, system elements. This represents an entity of the system irrespectively of how it is modelled. In a few cases independence of the modelling framework was practically shown when transforming some system

models used in the case studies from one notation to another (e.g. from a MODAF specific notation to UML), without having to reapply the method.

In conclusion, it is considered that evaluation of the DDA has been successful. The means of evaluation were applied extensively throughout the research and provided sufficient evidence. The methodology can offer a valuable perspective on typical issues regarding the dependability attributes, the concerns of the stakeholders, and how deviant behaviour of the system can affect the overall dependability of a system in its operational context.

7.2.2 Evaluation of the Trade-off Methodology (Chapter 5)

The Trade-Off Method (TOM) provides a qualitative approach for managing conflicting goals and arguing about trade-offs between goals.

One of the significant contributions of TOM is the introduction of the bounds in GSN. The bounds consist of a *target* and *limit*, which constitute the acceptability criteria of the goal and signify the region in which the goal could be traded-off against another. During the initial stages of the research there was some scepticism about having to explicitly think about the bounds during the definition of a goal, i.e. before encountering an actual conflict. However, during the research the benefits of thinking about the bounds became clear. The worked examples and case studies showed that defining the bounds enables reasoning about the overall acceptable levels of dependability in system operation. Definition of bounds involves capturing and sharing the rationale of the requirements between the stakeholders. Moreover, their introduction provided a useful means for unambiguously separating the core intent of the goals from their acceptability criteria.

During the definition of TOM, the use of numerical approaches for resolving trade-offs and in particular AHP was examined for its suitability to handling the trade-offs within a dependability case. TOM addresses the problems identified in numerical approaches by means of qualitative reasoning. Participants do not need to a priori prioritise the goals without considering the operational context of the system. Instead, the goals are traded according to the impact that a possible compromise will have on the operation of

the system. Willingness to trade-off helps in sharing of viewpoints and reaching consensus between the stakeholders of the system. Willingness to trade-off was recognised in peer review as being both easy to understand and apply. Another quality of the method that received positive comments is the fact that it does not provide a definite solution but it deliberately seeks debate and argumentation when selecting an option. Creation of the GSN arguments provides a clear way of expressing the motivation for selection of one between two or more competing decisions. Furthermore it adds traceability to the dependability case, as the resultant GSN argument reflects all the steps of the method.

One concern is scalability. Although TOM has worked acceptably well in the examples and case studies there is a concern with a major increase of competing objectives and alternatives. In such situations there may be occasions in which similarities between the alternatives make conclusion of an optimal decision difficult. Although such a problem was not encountered there are several suggestions for solving the problem such as further refinement of the *willingness to trade-off* categories. Although scalability may pose a challenge to TOM this is a shared concern for all of the reviewed trade-off methods. In numerical methods, scores close to each other make it difficult to appreciate the qualitative differences with respect to the system operation. Having scalability in mind the author, has written scripts for TOM which process decision alternatives and their related goals. The script generates a report in which the best option is identified and a list is produced showing the potential compromise and benefit from choosing an option other than the one identified as the best alternative.

Overall, TOM provided an alternative means of managing and arguing about trade-offs, in the context of a dependability case.

7.2.3 Evaluation of the Dependability Case Evolution and Architecture (Chapter 6)

Chapter 6 presents contributions at three different levels. Firstly, it shows the links between the methods (DDA, FANDA, TOM and GSN) and how they collaborate in order to evolve a dependability case in parallel with system design. Secondly, it presents FANDA, a design rationale method developed to work with TOM. Finally, the chapter proposes a dependability case architecture.

FANDA has a dual purpose during the evolution of the dependability case. Firstly it is used to identify the features (factors) of the design that affect the required dependability goals. Thereby FANDA highlights the elements of the design on which the developers should focus. The second use of FANDA is to assist brainstorming, the creation of design alternatives, and recording design rationale. Identification of design factors and how they affect the required goals proved to be a useful tool during application of TOM. Conflicts between design alternatives can be traced to the underlying design factors that cause them. Furthermore, it provides an early assessment of the impact of design factors on the goals before committing to an actual decision. FANDA employs the six hats method which helps participants to manage information related to the identified factors and to brainstorm, identifying new design alternatives. Furthermore FANDA can also be used to manage the collection of evidence regarding the evaluation of the alternative with respect to the dependability goals. FANDA was applied on a number of examples and it was also applied during case studies. It helped to clearly reveal the design factors responsible for a goal conflict. This was recognised during all forms of peer review and during application of the AGO case study. FANDA helped the participants in brainstorming alternatives. However, in the other case studies the system details were already finalised making application of FANDA less effective. Although the benefit from identifying sensitivity points in the design was clearly expressed, there was not adequate evidence to suggesting that use of FANDA resulted in proposing designs alternatives which otherwise would not have been conceived.

The Goal Structuring Notation (GSN) was chosen to capture the core arguments of the dependability case. GSN proved to be a correct decision in terms of:

- Previous experience from use in safety cases
- Appeal to peers and ease of use
- Extensibility

From an early stage in the research it was clear that creating a dependability case combining individual argument modules (using the features of modular GSN) was the most appropriate route. GSN modules provide a useful means of separating the dependability case into arguments corresponding to elements within the system

structure. GSN contracts help record how individual modules can be be composed to satisfy the overall objectives of the dependability case. Furthermore, modules make it possible to organise the dependability case in terms of the MODAF products. At the highest levels an argument was stated in context of the envisioned operation of the system, abstracting contextual information about system design. The proposed architecture for dependability cases was received positively both during case studies and peer review; especially the suggestion of a high level dependability argument focusing on the dependability qualities of interest to the stakeholders. This has been adopted as the starting point in other projects such as [101].

Additionally, the chapter provided a useful insight to the synergies between the proposed methodologies. Failures identified during dependability deviation analysis helped eliciting goals arguing about the acceptable behaviour of the system. The dependability profile was introduced as a means of capturing the requirements elicited during DDA without explicitly referring to DDA as this would result in weaker and more complicated arguments. The use of GSN contracts in conjunction with the dependability profile was accepted as being a viable solution constructing a dependability case. The argument modules that support the contracts as well as the profile of a SoS element, can potentially make clearer the decomposition of the dependability argument.

The suggestions of this chapter have been subject to lengthy discussions during the research. Feasibility of the suggestions was demonstrated with examples and creation of arguments during case studies. Regarding the benefit from the contributions, some of the suggestions have been used by peers, however the ultimate evaluation in terms of benefit will only come with wider industrial application.

7.2.4 Evaluation of the Metamodel

The metamodel is an important contribution of the research. The metamodel provides formalisation as well as a concrete representation of the dependability case concepts and their associations. In terms of feasibility the metamodel was created using tools widely established in the modelling research community. The correctness of the metamodel was achieved through numerous examples and application of case studies. For any

problems, ambiguities, or inefficiencies that were discovered the metamodel was changed appropriately.

The benefit from the implementation of the metamodel was immediately apparent and significant. The metamodel introduces traceability in the dependability case by unambiguously defining the associations between the concepts. Furthermore, traceability at the instance of the metamodel, can clearly show the related elements that were produced during application of the methods. This allows the use of automated verification, processing and transformation of the model. For example, scripts were written the functions of which included verification that there are no errors in the model, and automatically producing a list of applicable deviations from a template including the dependability attributes of interest and a basic traceability model between the system elements on which the deviation analysis was applied. Scripts accompanying the metamodel are a feature particularly useful regarding scalability. The industrial case study provided by BAE Systems produced a dependability case with approximately 1,500 elements. Scripts provided an invaluable means of managing the elements of the case within reasonable time limits. Moreover the metamodel makes a solid starting point for creation of a tool that will incorporate the methods. A proof of concept exercise was attempted by the author to create a tool using the eclipse GMF framework. The results of the exercise were very encouraging and a very simple application was created, able to create and graphically represent simple GSN arguments.

The metamodel proved a very successful means of communicating the proposed concepts, and positive comments as well as further interest regarding dependability cases echoed among all research peers.

7.3 Evaluation of the Thesis Proposition

The contributions were also assessed in terms of the distinct characteristics of the thesis proposal, as highlighted in Chapter 1.

- ***Structured:*** The Dependability Case Metamodel extends GSN to incorporate all of the concepts and associations necessary to structure a dependability case. In

addition, step-by-step guidance has been established for each of the methods proposed.

- ***Evolving:*** Iterative collaboration between the methods allows the evolutionary, analysis, design rationale and development of the design and the dependability case.

- ***Systems of Systems:*** This is the type of systems predominantly used in the examples and case studies throughout the research. DDA and FANDA in particular have been shown to work with Systems of Systems descriptions provided using the MODAF framework. The research has also shown how it is possible to use the modelling structure provided by MODAF products to organise and present a modular Systems of Systems dependability case.

- ***Unified:*** Definition of the metamodel was particularly successful in bringing together the concepts and associations underlying the proposed methods. As a result, when working on any element of the dependability case (e.g. a goal or a dependability concern) it is possible to clearly identify its association with other elements.

- ***Flexible:*** Flexibility was introduced by stating the dependability goals in the context of bounds of acceptability. Encouraging developers to think about flexibility from the early stages of the design is a necessary activity effective in supporting trade-offs between goals.

- ***Facilitating resolution of trade-offs:*** TOM systematically identifies potential trade-offs between dependability goals, enabling their resolution through a process of debate and argumentation.

- ***Argumentation:*** Using GSN as the underlying argumentation approach, the framework presented encourages the development of 'primary' dependability arguments, together with arguments concerning the rationale behind the

dependability goals as stated (e.g. concerning the bounds of acceptability) and justifying the optimality of the trade-offs made.

Intentionally Blank

Chapter 8

Conclusions and Future Work

8.1 Overall Conclusions

This thesis has presented an integrated approach for managing and evolving dependability cases. The contributions of this thesis can be summarised as follows:

- Definition of a dependability requirements elicitation technique. Dependability Deviation Analysis (DDA) is the proposed method, documented in Chapter 4.

- Systematic management, analysis and justification of trade-offs resulting from competing dependability objectives. The Trade-Off Methodology (TOM) is the proposed method presented in Chapter 5.

- Support for the evolutionary development and architecting of dependability cases in parallel with system design processes and structures. The Factors Analysis and Decision Alternatives (FANDA) method, together with a proposed architecture for the dependability case, are documented in Chapter 6.

- Definition of the Dependability Case Metamodel (DCM) that rigorously captures the associations between the concepts inherent in dependability cases. The technical approach taken to define the metamodel is presented in Chapter 3. The complete metamodel can be found in Appendix C.

This chapter discusses how the contributions presented in this thesis support the proposition stated in Chapter 1, and discusses areas of future work.

8.1.1 Conclusions on Dependability Deviation Analysis

Chapter 4 discusses how safety analyses are used to analyse the system and elicit requirements during each stage of the system lifecycle. Dependability Deviation Analysis (DDA) is a method for the analysis and elicitation of dependability requirements, optimised for the SoS paradigm. DDA extends existing well established deviation-based safety analysis techniques. It includes a number of novel concepts. DDA allows participants to explore how deviations – established from the perspective of one dependability attribute – can impact the achievement of other attributes. DDA introduces the concept of failure maps to explicitly document and visualise these associations. Creation of failure maps relies upon the underlying traceability between system models. Although this can be found in some safety analysis techniques (e.g. Hip-Hops [105]), DDA has been optimised for models representing SoS behaviour (specifically MODAF). Finally, DDA includes checks for distinguishing the 'end' requirements – requirements of primary interest to the stakeholders – from the 'means' requirements – requirements that are contributing factors in achieving the former.

8.1.2 Conclusions on the Trade-Off Method

The Trade-Off Method (TOM), documented in Chapter 5 is a methodology for systematic identification of trade-offs, facilitating the production of arguments justifying the associated decisions. The arguments produced provide essential context for the development of the dependability case. A distinct characteristic of TOM is the adoption of qualitative, rather than quantitative, reasoning. Moreover, TOM introduces the concept of flexible requirements, considered necessary to enable trade-offs to be made. Whilst inspired by elements of the ALARP principle, TOM allows examination of trade-offs between multiple dependability attributes (in contrast to the safety – cost trade-offs made in the ALARP framework).

8.1.3 Conclusions on the Dependability Case Evolution

Chapter 6 describes how a GSN based dependability case can be developed in parallel to the system design. In the six-step method for constructing arguments using GSN developers are encouraged to identify and document the strategies they have used in argument decomposition. However, it provides little guidance on *how* strategies are

identified. FANDA focuses on brainstorming and gradual elicitation of design rationale in order to identify possible decision alternatives. Although FANDA is presented in the context of SoS configuration decisions, experience from its application shows potential for its use in other domains. Moreover, the chapter discussed the collaboration of the proposed methods in order to establish a dependability case. Finally, a dependability case architecture is presented that integrates the outputs of the proposed methods.

8.1.4 Conclusions on the Dependability Case Metamodel

At present there is no established metamodel for assurance cases. Existence of a rigorous metamodel that captures the concepts of the assurance cases domain can deliver a number of benefits. Most importantly, it offers a common vocabulary and consensus on the concepts involved in the task of assurance case development and their semantics. Recently, there has been increasing interest in this subject. In particular, the Object Management Group will shortly be issuing a Request for Proposals for an assurance case metamodel.

8.2 Revisiting the Dependability Case Roadmap

Chapter 1 described a number of the challenges that exist in establishing a dependability case for Systems of Systems. The following challenges were introduced:

1. Multiple dependability attributes
2. Allocation and apportionment of requirements
3. Conflicting requirements
4. Changing requirements
5. Traceability
6. Interaction of case and design
7. Ownership of the dependability case

As stated in Chapter 1, this research specifically targeted a number of these challenges, namely 1, 2, 3, 5 and 6. The work presented in this thesis has made the following contributions with respect to the targeted challenges:

8.2.1 Multiple Dependability Attributes

Interaction of dependability attributes is identified during DDA, in which failure conditions are associated, thereby identifying how the operation of the SoS from the perspective of one attribute can affect the operation of the SoS from the perspective of other dependability attributes. Requirements regarding all of the attributes of interest for a specific SoS element are collated using the *dependability profile*. Dependability profiles are used to structure the architecture of the dependability case. The proposed dependability case architecture does not simply merge heterogeneous attributes (an approach identified to be problematic). Using the traceability mechanism introduced by the rigorous definition of all associations between the dependability case concepts, reviewers of the case can trace an overall dependability case (GSN) goal to the requirement within the dependability profile for a given SoS element, and the attribute of interest with which the goal is associated.

8.2.2 Allocation and Apportionment of Requirements

Allocation and apportionment of requirements is an activity that takes place during DDA. Initially the system stakeholders identify their overall concerns with regard to the system's operation. Following that, using guidewords, the design of the system is prompted to identify how deviating from the intended operation (with respect to a dependability attribute) can affect the overall operation of the system. Accordingly, appropriate requirements are derived for each system element. However, the methods proposed in this thesis do not explicitly handle negative emergent behaviour of SoS. DDA allows engineers to hypothesise deviations and map the causal relationships between failure conditions. However, negative emergent behaviour can arise even when no deviations have occurred (e.g. through the composition of individual systems' *normal* behaviour).

8.2.3 Conflicting Requirements

The Trade-Off Method allows stakeholders to examine their requirements – identifying and justifying the extent to which compromise of their requirements can be tolerated. TOM allows stakeholders to exchange views concerning the preference of one design alternative over another, as well as helping establish the rationale for the chosen option.

After applying the method stakeholders will have established a justification for the trade-off made (communicated through a structured argument) which can then be used to support the dependability case.

8.2.4 Traceability

The Dependability Case Metamodel was defined as a means of documenting the associations between the contributions of this thesis. It provides traceability between the concepts used in this thesis that contribute to establishing a dependability case. In particular, methods that support the trade-offs and evolution of the dependability case. By instantiating the metamodel (instances of) fully traceable dependability cases are created. Having an underlying metamodel allows the (instantiated) models to be fully navigable. Moreover, with the help of modelling management tools, certain aspects of dependability case evolution can be automated.

8.2.5 Interaction between System and Case Development

This thesis has defined how the proposed methods can be used in combination to support the evolution of a dependability case, alongside the decision-making processes of system development. FANDA facilitates the evolution of the case and system, by capturing how features of proposed decision alternatives will affect the goals of the system. Documenting these observations FANDA facilitates further elicitation of design alternatives and identification potential improvements to existing alternatives. Moreover, the various interfaces between the stages of the (proposed) methods are captured describing and documenting their collaboration during evolution of the dependability case. Finally the products of the proposed methods are related with the dependability case product (describing how they are used in the final dependability case), resulting in an architecture for a dependability case.

8.3 Areas of Further Work

During the course of the research reported in this thesis, some possible directions for future work, improving the proposed concepts have been identified:

- Extending the library of issues and deviations

- Determining assurance levels in dependability cases
- Dependability cases in the presence of change
- Socio-technical issues concerning flexible requirements and trade-offs

These areas are discussed further in the following sub-sections.

8.3.1 Extending the Library of Issues and Deviations

During the development of DDA, a number of typical *issues* were identified for 'key' dependability attributes that were applicable across a wide range of domains. However case studies, in particular the HAP case study, identified a number of domain specific dependability issues that needed to be considered. Moreover, deviations have been optimised alongside models that feature particular characteristics capable of exploring dependability concerns. Further study of domains and cataloguing of related issues, as well as incorporation of other model frameworks such as SysML, could expand the applicability of DDA.

8.3.2 Determining Assurance Levels in Dependability Cases

Dependability case arguments are, and will be, inherently subjective – relying upon inductive reasoning. This thesis has demonstrated how we can capture claims regarding the acceptability of a system. An important facet of argumentation is the degree of assurance with which these claims are made. Work done by Weaver et al. [106] demonstrates how assurance levels can be incorporated in argumentation. Applying this to the dependability case is a potentially useful way forward enhancing the framework proposed in this thesis. In particular, it would be interesting to explore the relationship between the newly introduced concepts of GSN 'bounds' and the determination and achievement of assurance levels. As defined currently, bounds identify the limits of an acceptable solution in terms of achievement claims, without considering the degree of *assurance* of those claims. There will be a relationship between these two concerns. For example, a *stronger* claim, closer to the parent goal's target, may be more *weakly* assured that a claim closer to the parent goal's limit.

8.3.3 Dependability Cases in the Presence of Change

One of the characteristics identified for System of Systems is that they often include re-configuration. This means that the assumptions and context regarding the SoS may change. A dependability case based on these assumptions and context will be called into question following re-configuration. It will be unrealistic to expect complete re-evaluation of a dependability case following every change to a SoS configuration. In this thesis a modular dependability case architecture has been proposed that will *potentially* allow changes to the dependability case to be bounded and limited to only affected parts. In addition, the concept in this thesis of establishing *dependability profiles* that capture the required characteristics of system elements could help support change by enabling substitutions of system elements with matching characteristics. However, further work is required to explore how a dependability case architected in accordance with the principles in this thesis will cope with realistic change scenarios (and potentially timescales).

8.3.4 Socio-technical Issues Concerning Flexible Requirements and Trade-offs

Whilst it is accepted that conflicts and trade-offs are inevitable, it is not current subcontracting practice to contract elements of the system using flexible requirements. There is a general belief that adopting such an approach would require a framework for ensuring that suppliers have made 'genuine' trade-offs, not simply aiming for the minimum acceptable system. Moreover, the significant of trade-off decisions needs to be related to authority to make such decisions. An example where this is demonstrated is shown in the U.S. MIL-STD-882C [107]. In this standard the higher the residual safety risk the higher authority is required for ultimate sign-off and approval.

8.4 Final Remarks

The concept of a dependability case may initially be perceived as a straightforward extension of the well established concept of a safety case. However, this thesis has shown that the concept brings with it a number of new challenges, primarily concerned with the management of the interrelationships that exist between dependability attributes. These challenges are particularly apparent in the operation of Systems of Systems, where the problem becomes one of configuring a network of interoperating

systems in such a way as to address the dependability objectives associated with the overall concept of operation. Through the methods presented in this thesis we have begun to address these challenges. The methods defined support the systematic development of a dependability case – from the initial identification of dependability objectives, the management of trade-offs, and the evolution of the case in step with the configuration of Systems of Systems. However, a number of challenges remain in establishing the dependability case concept for System of Systems. Questions still exist, for example, concerning the overall ownership of the Systems of Systems dependability case, and the sustainability of the concept in the presence of rapidly changing requirements and system configurations. The work presented in this thesis provides a framework within which these, and other, issues can continue to be explored.

Intentionally Blank

Appendix A

Overview of the ARP 4761

Safety is a system property, the achievement of which has a crucial contribution to the system's final acceptance. Due to its importance, safety should be methodically analysed along the system lifecycle. A number of standards and recommended practices define the processes and the objectives of the safety lifecycle.

The civil aerospace guidance document ARP 4761 provides a comprehensive guide of safety analysis for airborne systems [70]. ARP 4761 suggests the following activities during a system's lifecycle: Preliminary Hazard Analysis (PHI), Functional Hazard Assessment (FHA), Preliminary System Safety Analysis (PSSA) and System Safety Analysis (SSA). The safety lifecycle takes place in parallel with the system lifecycle providing appropriate feedback, according to the design information available, during the evolution of the system. The FHA is a process that takes place at the beginning of the system development cycle. The purpose of the FHA is to identify and classify the failure conditions that are associated with the system functions or with combinations of system functions. The rationale for the classification of the failure conditions is specified by taking into account the risk of each condition, justifying the classification based on the severity of the failure. Furthermore FHA examines failure conditions involving multiple functions as well as multiple systems, which is conducted fault tree analysis (FTA) at function and system level respectively.

PSSA is conducted interactively with the design of a system and it is used to complete the failure conditions list and elicit detailed safety requirements for each of the participating systems and subsystems. PSSA extends FHA and uses the conclusions of the FHA to further explore how the individual systems can contribute to the identified failure conditions. PSSA uses fault trees to identify how possible single or combination of system of function failures can affect the assessed failure conditions during the FHA. The PSSA results to detailed safety requirements for the system which are examined and design strategies are proposed for their achievement.

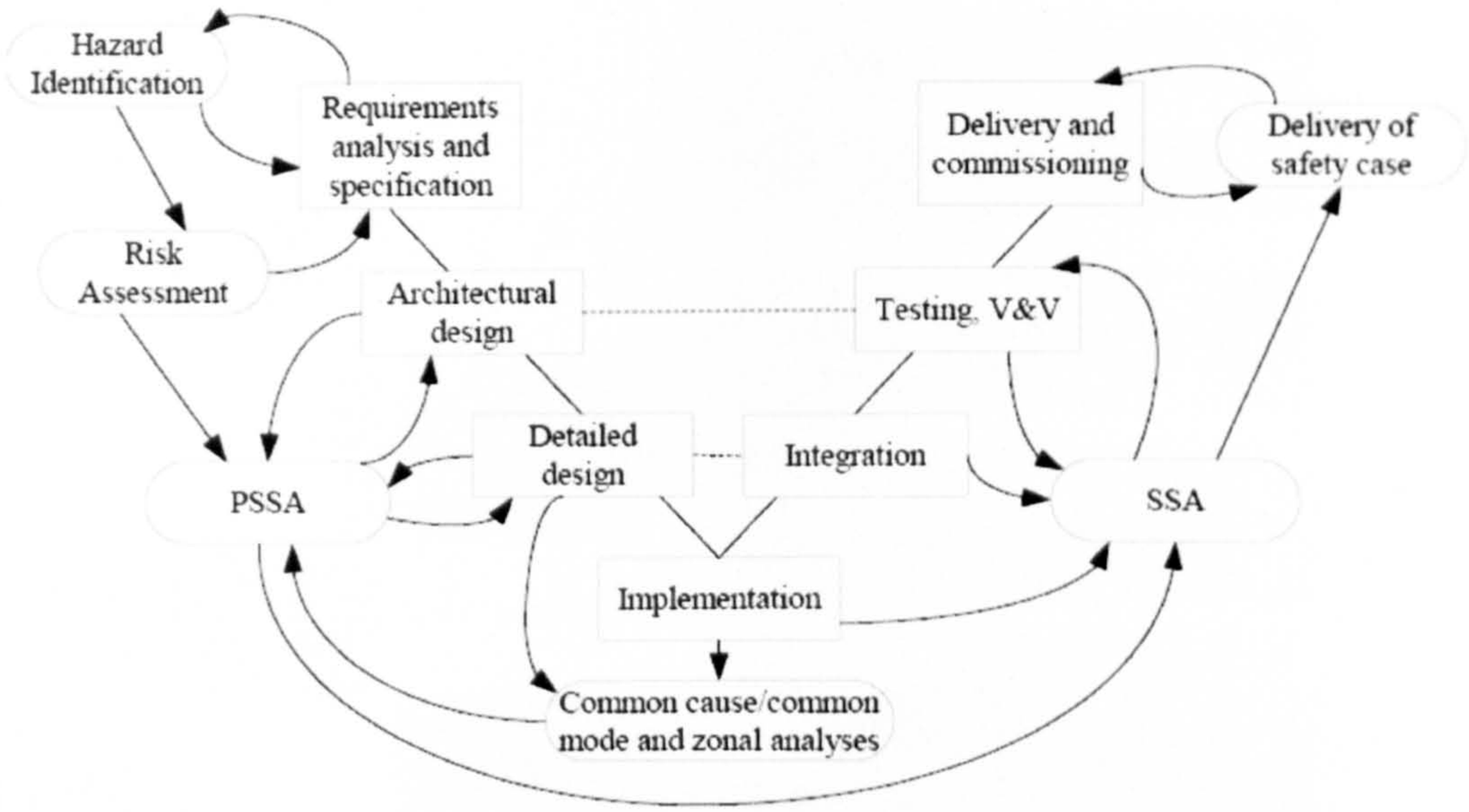

Fig.A.1 – ARP 4761 Steps within the V-Lifecycle

Fig.A.1 presents Pumfrey's adaptation of the V-lifecycle model [76] including the main safety activities during the development of a system, such PSSA and SSA which are also described in ARP 4761. A prominent feature of the diagram is the clear association of the safety activities and the system design activities. The proposed safety lifecycle model is nearly identical to what is described in ARP 4761. Fig.A.2 shows an excerpt from ARP 4761 that has been abstracted to highlight the relation between function analysis and system specification with extraction of general and system specific requirements during the development of a system. Initially identification of the overall safety objectives of the system with respect to its overall functionality and the functionality of the identified systems (FHA & System FHA) takes place. Understanding of how the individual systems contribute to the overall functionality of the platform (i.e. aircraft) is essential in order to apportion specific requirements to each of the systems. FHA and system level FHA are followed by PSSA, which involves identification of the proposed detailed design of the system.

During PSSA the identified failure conditions in FHA and system FHA are evaluated against the proposed design, which may result in altering the design in order to improve the overall safety levels. SSA takes place during the evaluation of the system and the main purpose to complete the safety assessment with verification that the requirements set during FHA and PSSA have been achieved.

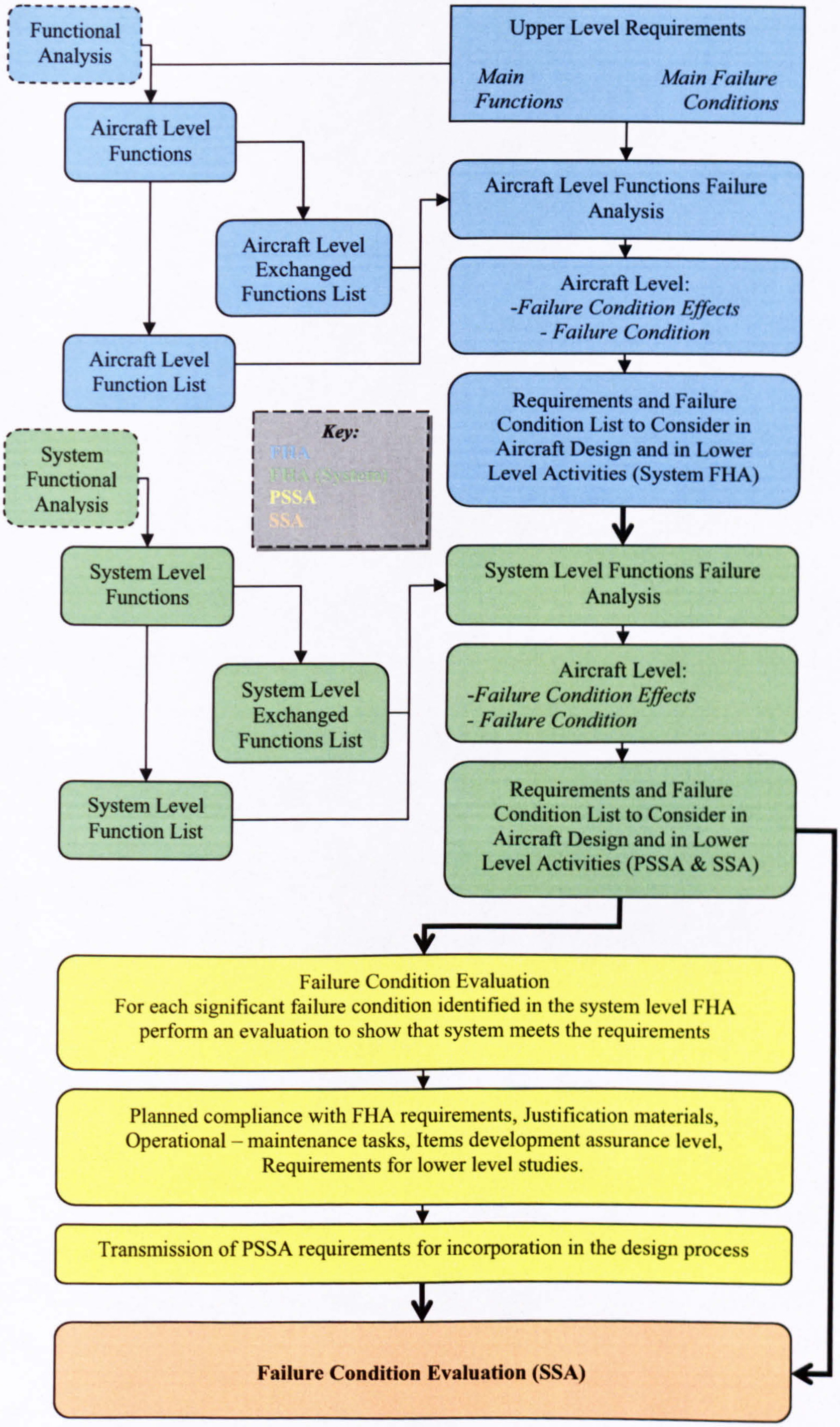

Fig.A.2 – The ARP 4761 Processes

The civil aerospace guidance document ARP 4761 provides a comprehensive guide of safety assessment analysis for airborne systems [70]. ARP 4761 suggests the following activities during a system's lifecycle: Functional Hazard Assessment (FHA), Preliminary System Safety Analysis (PSSA) and System Safety Analysis (SSA). Fig.A.3 presents a schematic of how the safety analysis stages are associated with the system lifecycle.

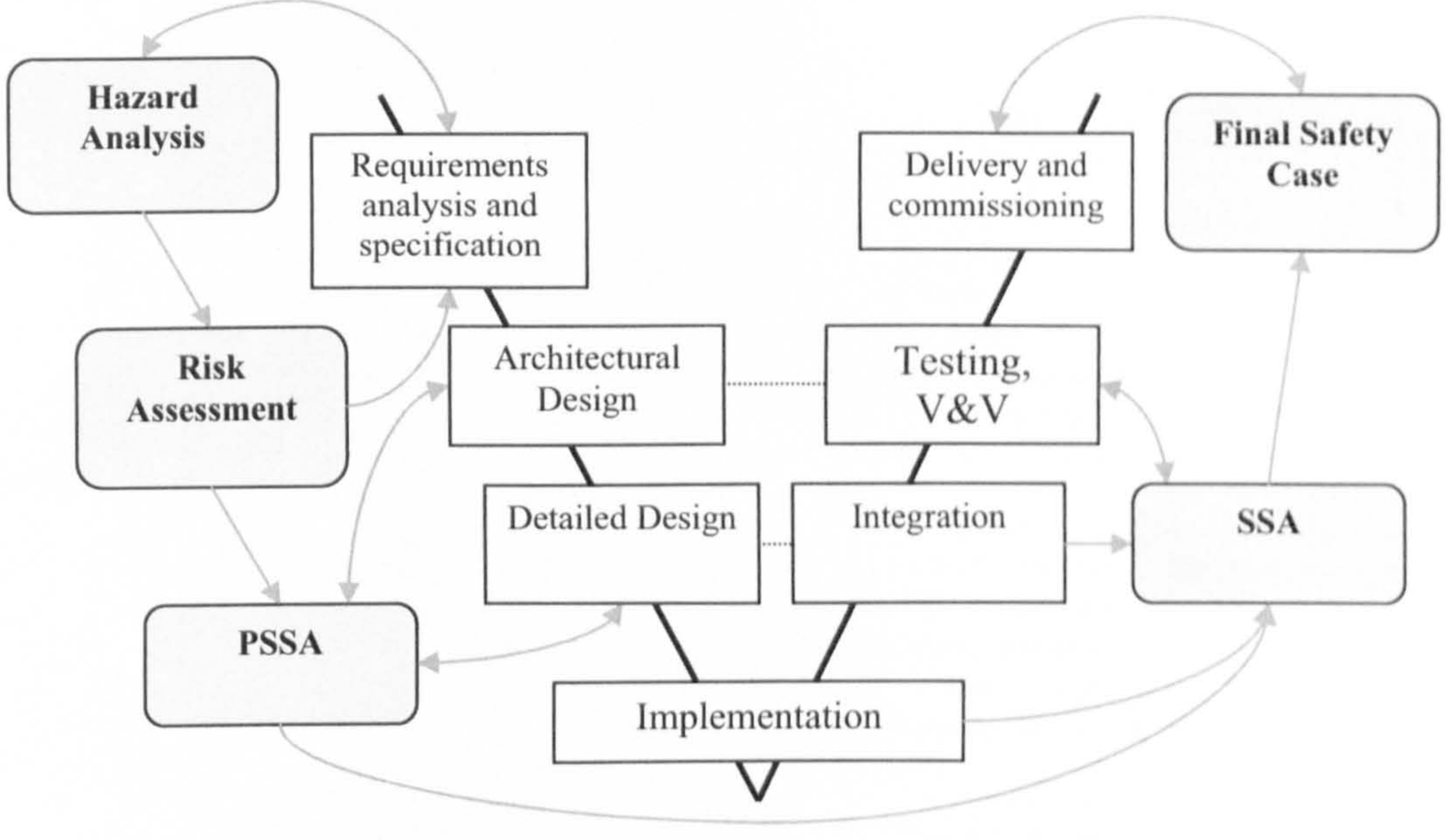

Fig.A.3 – Safety Analyses During System Lifecycle [76]

FHA considers the aircraft functions and identifies failure conditions related to them. The participants during FHA are required to evaluate the consequences of each failure condition and assign probability target according to the severity of each failure condition. Taking into account the contribution of the aircraft systems to the overall aircraft functions, analysts specify safety objectives for the constituent systems of each function. Table.A.1 shows an excerpt from ARP 4761 FHA, in which the breaking function (whilst on ground) of a passenger aircraft is evaluated with respect to a set of possible failure conditions.

FHA is followed by PSSA which is described by ARP 4761 as "*a systematic examination of the proposed system architecture to determine how failures can lead to the functional hazards identified by the Functional Hazard Assessment (FHA), and how the FHA requirements can be met*". Finally the SSA is a confirmatory analysis, which

takes place at the end of the system lifecycle, verifying that the safety targets that were finalised during PSSA have been met.

Table.A.1 – FHA for an Aircraft Wheel Braking System

Function	**Failure Condition (Hazard Description)**	**Phase**	**Effect of Failure Condition on Aircraft / Crew**	**Classification**	**Verification**
Decelerate Aircraft on the ground	Loss of Deceleration Capability	Landing /RTO /Taxi	See Below		
	a. Unannunciated loss of Deceleration Capability	Landing /RTO	Crew is unable to decelerate the aircraft, resulting in a high speed overrun	Catastrophic	S18 Aircraft Fault Tree
	b. Annunciated loss of Deceleration Capability	Landing	Crew selects a more suitable airport, notifies emergency ground support, and prepares occupants for landing overrun	Hazardous	S18 Aircraft Fault Tree
	c. Unannunciated loss of Deceleration Capability	Taxi	Crew is unable to stop the aircraft on the taxi way or gate, resulting in low speed contact with terminal, aircraft or vehicles.	Major	
	d. Annunciated loss of Deceleration Capability	Taxi	Crew steers the aircraft clear of any obstacles and calls for a tug or portable stairs	No safety effect	

Pumfrey [76] classifies the analysis techniques according to the amount of information available during the system's lifecycle, about the identified failure with respect to causes and effects. Hence the causes and effects of a failure can be unknown, projected and known. Fig.A.4 shows how each of the analysis stages described in ARP 4761 can contribute into increasing the information about the design. At the beginning of the system's lifecycle exploratory analyses are employed to identify system failures that constitute possible hazards. During this stage of the system's lifecycle, the causes as well as the effects are unknown. At this stage analysts extrapolate possible effects based on the preliminary studies of the design and early identification of hazards. As the analysis methods are applied on the gradually evolving design, the confidence about the causes and effects increases.

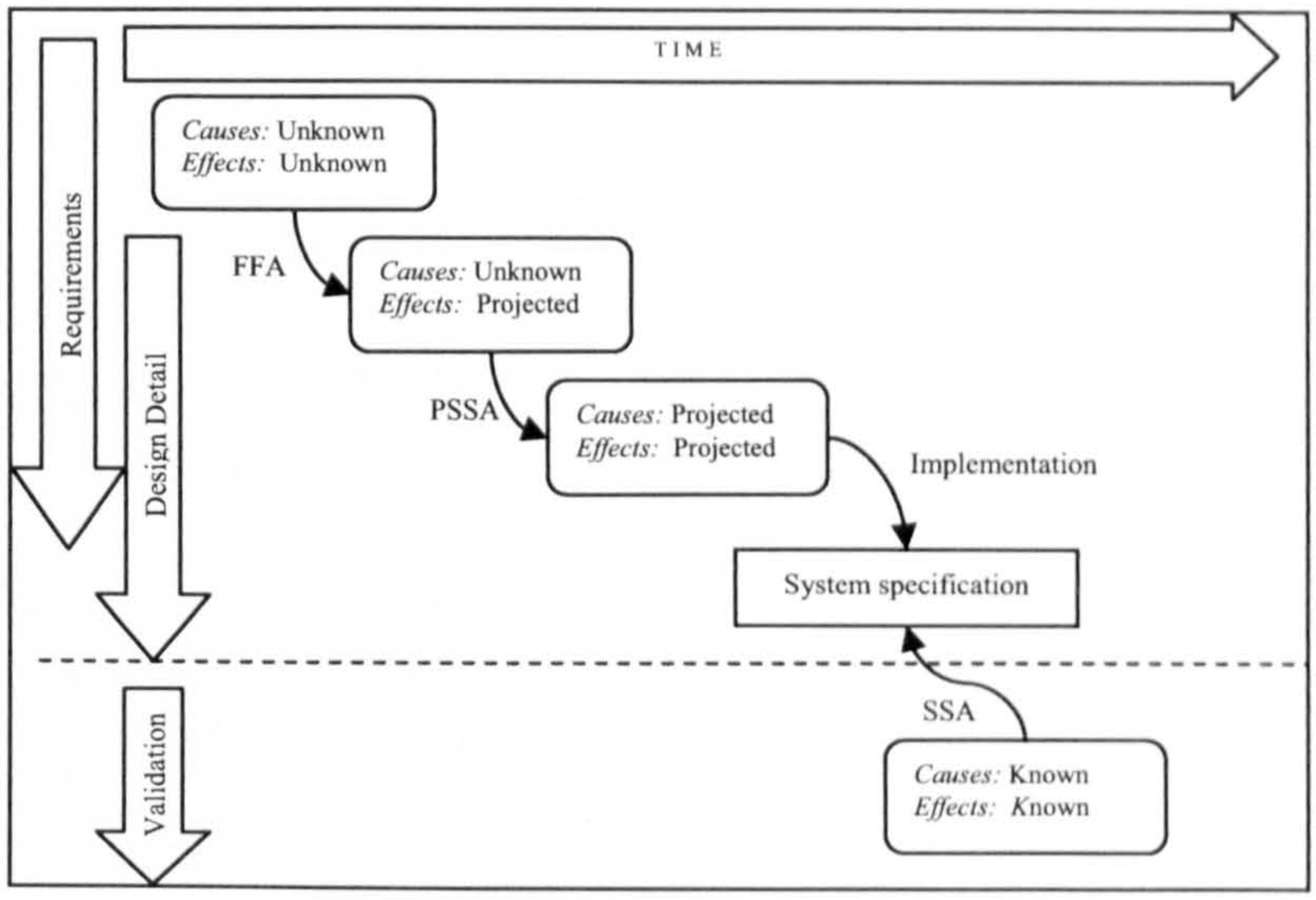

Fig.A.4 – Failure Information During System Lifecycle

Exploratory analyses assist in considering possible failures and eliciting requirements about the system. In specific, after the completion of PSSA the requirements should be clear and complete and the involved stakeholders should be able to understand the rationale for their elicitation. For example in ARP 4761 the PSSA identifies how each individual system contributes in achieving the functions analysed in FHA (Table.3.1). The achievement of the design to satisfy the requirements is shown by employing confirmatory analysis. SSA is the phase of the lifecycle, during which the design has reached a mature and enough detailed stage, can be used to identify with certainty weather the safety requirements have been met. Starting from known causes and effects confirmatory analysis takes place verifies the requirements and the projected safety levels specified in during the earlier stages of the safety analysis. Ideally, both the causes and effects of a deviation in the intended system operation should be known at the end of the safety lifecycle.

Intentionally Blank

Appendix B

Overview of MODAF and the AGO Scenario

The emergence of concepts such as the Network Centric Warfare (DoD) and the Network Enabled Capability (MoD) has resulted in highlighting many aspects, necessary for their intended operation of a system. Analysis and design evolution of such systems required a concrete framework which could be used to communicate architectures as well as their intended operation. The US Department of Defence proposed an architecture framework (DODAF) to be used by all the different stakeholders to represent their concerns, during the system's entire lifecycle. In this document we present how the methodologies and descriptions provided by the DoD and MoD can be used to model a system, by illustrating the Anti Guerrilla Operations (AGO) scenario, as presented in the strand 2 examples.

1. Overview

DODAF is used to *"define a common approach for DoD architecture description development, presentation, and integration for both war fighting operations and business operations and processes"*. DODAF has evolved over the years from frameworks that were used by individual bureaus within the US DoD, initially resulting in the C4ISR Architecture Framework [21]. The framework consists of several products organised in views, which when used will ultimately describe the complete operation of the system. DODAF products are organised in three main views: Operational, Systems and Technical, accompanied by a fourth view used as reference and to maintain consistency between the other views, called the All-Views (Table.B.1 summarises the DODAF products).

MODAF is the equivalent architecture framework defined by the UK ministry of defence. MoD has tailored DODAF to its needs with the addition of the strategic and the acquisition views. Table.B.2 presents the products of the additional MODAF views.

Table.B.1 – Overview of the DODAF Products and Views

ID	OV (Operational)	SV (Systems)	TV (Technical)	AV (All-views)
1	High-Level Operational Concept Graphic	Systems Interface Description	Technical Standards Profile	Overview and Summary Information
2	Operational Node Connectivity Description	Systems Communications Description	Technical Standards Forecast	Integrated Dictionary
3	Operational Information Exchange Matrix	Systems-Systems Matrix	-	-
4	Organisational Relationships Chart	Systems Functionality Description	-	-
5	Operational Activity Model	Operational Activity to Systems Function Traceability Matrix	-	-
6	*a)* Operational Rules Model	Systems Data Exchange Matrix	-	-
	b) Operational State Transition Description	-	-	-
	c) Operational Event-Trace description	-	-	-
7	Logical Data Model	Systems Performance Parameters Matrix	-	-
8	-	Systems Evolution Description	-	-
9	-	Systems Technology Forecast	-	-
10	-	*a)* Systems Rules Model	-	-
	-	*b)* Systems State Transition Description	-	-
	-	*c)* Systems Event-Trace Description	-	-
11	-	Physical Schema	-	-

Table.B.2 – MODAF Additional Views

ID	StV (Strategic View)	AcV (Acquisition View)
1	Strategic Capability Vision	SoS Acquisition Clusters
2	Capability Functions	SoS Acquisition Programmes
3	Capability Phasing	-
4	SoS Clusters	-
5	Capability to Systems Deployment Mapping	-

2. DODAF Modelling notations and Frameworks

DODAF is intended to provide the common grounds for description of architectures. DODAF does not include any specific modelling methodologies/notations. Instead, it provides descriptions of what information the products at each stage of the system's lifecycle should capture.

This provides DODAF with the flexibility to use any model that can satisfactorily address all the issues that are required for each of the products. UML is the modelling language most commonly used to model DODAF products, even though there are problems to overcome, mainly due to the fact that there is no definite way to model the DODAF products in UML.

The DODAF and MODAF as well as other studies (e.g. [90]), suggest UML models that can be employed to represent the DODAF/MODAF products. Table.B.3 presents the products and the UML models that can be employed to construct them.

Table.B.3 – Representation of DODAF/MODAF Products in UML

[D]: DODAF Technical Specification, [M] MODAF Technical Specification,
[T]: Telelogic's white paper on modelling NCW and NEC

ID	Product	Recommended Modelling in UML
StV1	Capability Vision	n/a (free format)
StV2	Capability Functions	«Capability» Stereotyped Class
StV3	Capability Phasing	n/a
StV4	SoS Clusters	«Capability» Stereotyped Class with Dependencies
StV5	Capability to Systems Deployment Mapping	Stereotyped classes representing «Systems» can be overlaid on a 2-D (X,Y) table representing capabilities (X) and organisations (Y).
StV6	Capability Function to Operational Mapping	n/a (tabular form)
OV1	High Level Operational Concept	n/a (free form) [D], [M] Class or use case diagram & graphics [T]
OV2	Operational Need Connectivity Description	Collaboration diagrams [D] Composite (Stereotyped) Structured Diagram [M], [T]
OV3	Operational Information Exchange Matrix	n/a [D], [M] UML report queries [T]
OV4	Organisational Relationships Chart	Class diagram using actor icon [D] Class diagram [M], [T]
OV5	Operational Activity Model	Operational activity diagram with object flows & actors representing nodes [D], [M], [T]
OV6a	Operational Rules Model	n/a (free text), post and pre conditions and OCL can be applied on use cases [D], [M], [T]
OV6b	Operational State Transition Description	Statecharts [D], [M], [T]

OV6c	Operational Event Trace Description	Sequence diagram [D], [M], [T]
OV7	Logical Data Model	Class diagram [D], [M], [T]
SV1	Systems Interface Description	Deployment diagrams [D] Class diagram using stereotypes [M] Composite structure diagram [T]
SV2	Systems Communications Description	n/a however could be modelled using deployment & component diagrams denoting the communications links [D] SysML (UML 2) [M] Composite Structure Diagram [T]
SV3	Systems-Systems Matrix	n/a, tabular form [D], [M] DOORS traceability view [T]
SV4	Systems Functionality Description	Use cases, classes and class operations [D], [M] Activity diagram with object flows [T]
SV5	Operational Activity to Systems Functionality Traceability Matrix	n/a however can be gathered from OV-5 and SV-4. «Include» relationships. [D], [M] DOORS traceability view [T]
SV6	Systems Data Exchange Matrix	n/a [M], however the matrix product expands on the information associated with SV-1 systems, SV-4 use cases, and system data flows [D]. UML model queries [T]
SV7	Systems Performance Parameters Matrix	n/a [D], [M] UML model queries [T]
SV8	Systems Evolution Description	n/a [all]
SV9	Systems Technology Forecasts	n/a [all]
SV10a	Systems Rules Model	Pre, post conditions on classes of SV-4 [D], OCL [M] Text [T]
SV10b	Systems State Transition Description	Statechart [D], [M], [T]
SV10c	Systems Event Trace Description	Sequence diagram [D], [M], [T]
SV11	Physical Schema	Class diagram [D], [M], [T]

3. Evolution and Model Dependencies

The MODAF and DODAF technical specification do not provide definite guidelines regarding the stage of the lifecycle that each of the products should be constructed. Each of the products can be used at any stage of the system lifecycle if it's useful for the concerned stakeholders. However there are examples of suggested use of the models. The department of navy use the Architecture Definition Process Management which defines 432 tasks and their interdependencies. MODAF acquisition community of interest handbook suggests a set of products for requirements management during the CADMID [109] lifecycle, shown in Fig.B.1. Furthermore the DODAF deskbook gives a data-centric perspective in which the data is developed and organised to continually add layers of complexity to the description of the enterprise, creating integrated architectures (shown in Fig.B.2).

User Requirements Document

- **StV6, Capability Functions to Operational Mapping (Concept)**
- **OV1, High Level Operational Concept (Concept)**
- **OV2, Operational Node Connectivity Description (Assessment)**
- **OV3, Operational Information Exchange Matrix (Assessment)**
- **OV5, Operational Activity Model (Assessment)**
- **TV1, Technical Standards Profile (Assessment)**

System Requirements Document

- **OV7, Logical Data Model (Assessment)**
- **SV1, Systems Interface Description (Assessment)**
- **SV3, Systems – Systems Matrix (Assessment)**
- **SV4, Systems Functionality Description (Assessment)**
- **SV2, Systems Communication Description (Assessment)**
- **SV6, Systems Data Exchange Matrix (Assessment)**
- **SV7, Systems Performance Parameters Matrix (Assessment)**
- **TV1, Technical Standards Profile (Assessment)**

Fig.B.1 – MODAF Requirements Documents

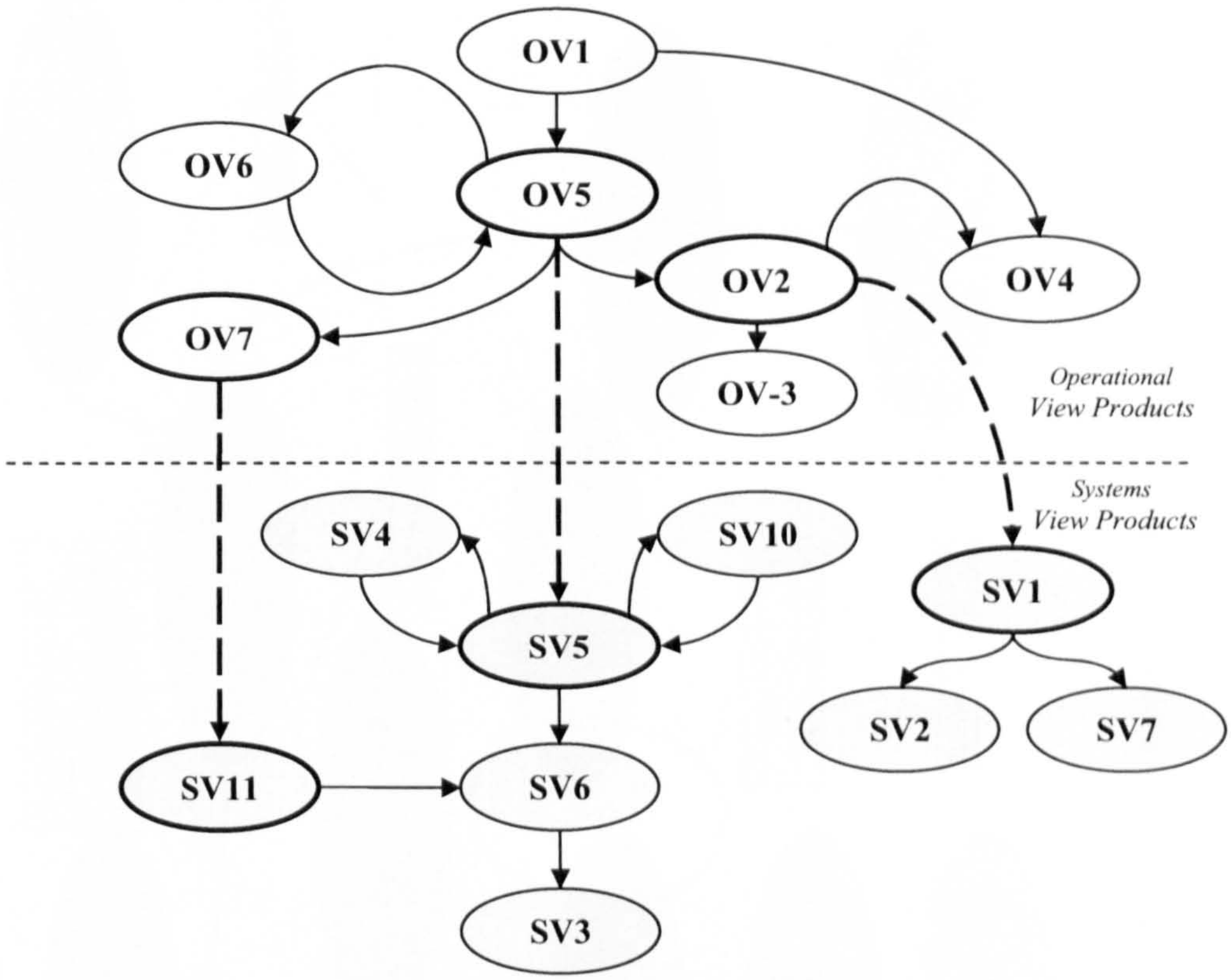

Fig.B.2 – DODAF Product Dependencies (Data Centric Perspective)

According to Fig.B.2 the development of the products is oriented around the definition of the operational activity diagram (OV5) and the mapping of those activities onto individual systems (SV5). The two views have three main 'design areas' with which they are linked. The first represents the functions of the system and their activities and how these are mapped onto individual systems. The second refers to the data and entities that need to be used and how the systems represents the defined entities and use the identified data. Finally the third area involves communications and in general links that need to be established in order for the system to perform the specified functions. Using the MODAF metamodel Fig.B.3 presents the common classes between the products according to how their relations as indicated in Fig.B.2. Identifying the common classes between two products can help the analysts understand the system element shared by the two products.

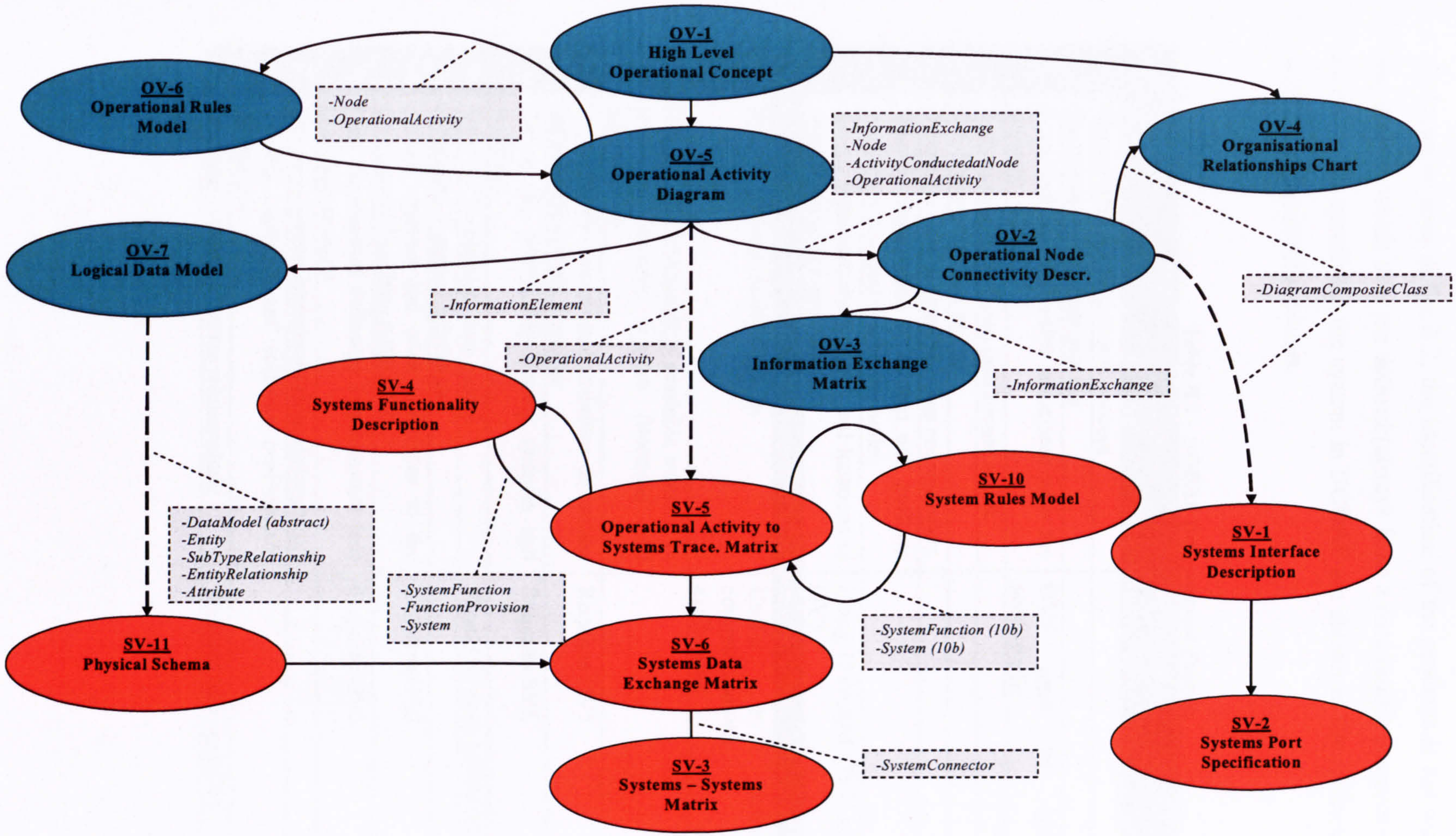

Fig.B.3 – Common Classes between MODAF Products (Data Centric)

As it can be seen in Fig.B.2, the identification of the products is not representative of the order in which they are defined (always from a data centric perspective). The main activities for specifying the system in DODAF are shown in the following table along with their respective products.

Table.B.4 – DODAF Development Process

	Activity	Comment	Product
	Development of Operational View		
1	Obtain/ build an operational concept		OV1
2	Document the business process		OV5
3	Document business rules associated with the business process	OV5 must be updated accordingly	OV6
4	Aggregate activities into operational nodes		OV2
5	Develop logical data model		OV7
6	Determine information exchange requirements		OV3
7	Identify organisation types that will perform the activity associated with the nodes		OV4
8	Assign organisations and physical locations to operational nodes and activities	Using OV4 and OV2, update OV3	OV3
	Development of Systems View		
1	Identify physical node locations	Useful to determine communications asset availability	
2	Identify and characterise available systems in terms of owners, system functions and performance		
3	Based on the operational activities determine the requires system functions	Requires OV5	SV5
4	Define the relationship among systems and functions	Requires SV5	SV4
5	Develop the physical data model	Should be done before SV6	SV11
6	Determine systems' behaviour		SV10
7	Assign systems and other interfaces to the Operational Facilities (OPFACS)	Requires OV2	SV1
8	Map information exchange requirements into candidate systems	Requires OV3	SV6
9	Develop systems communications description		SV2
10	Identify hardware and software performance parameters		SV7
11	Describe system to system relationships	Requires SV6 and SV1	SV3

4. The Anti Guerrilla Operations (AGO) Scenario

The Anti Guerrilla Operations (AGO) is a hypothetical scenario, in which the army forces (in this case air force, artillery and Special Forces) collaborate, based on paradigms described by the concepts of NCW and NEC, to suppress guerrilla operations in a hostile territory. The case study includes example products for the operational view and two of the system view products (SV4 & SV5). The systems view was not developed in full due to the detailed technical knowledge that it requires. The operational view and products 4 and 5 from the systems view are adequate to demonstrate the logical links and traceability between the DODAF products when developing an integrated architecture.

OV1 High Level Operational Concept

The following figure shows a schematic of the overall concept of operations during the AGO scenario as described in [110].

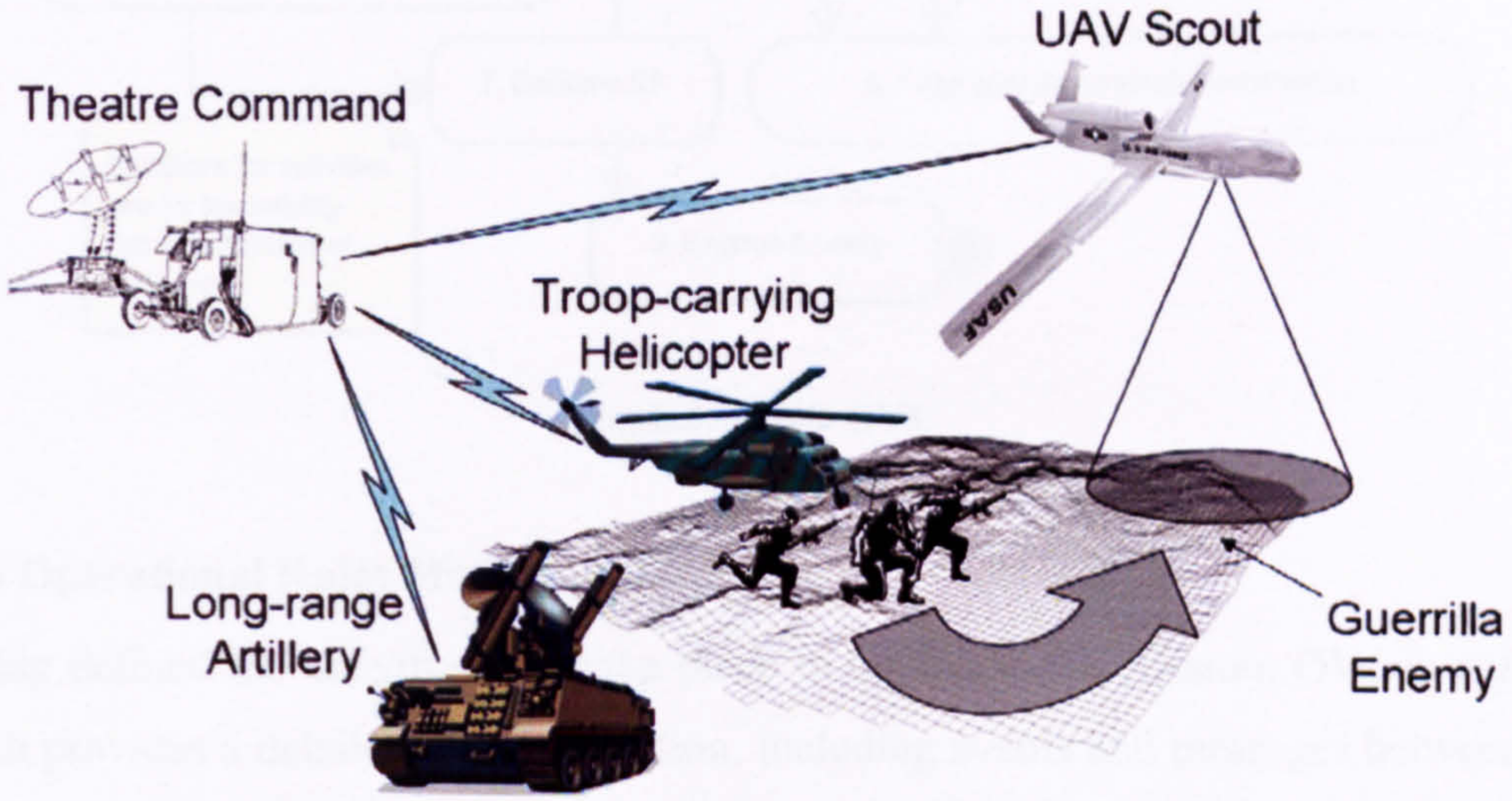

Fig.B.4 – AGO OV1

According to the scenario guerrilla activity is identified by unmanned vehicles patrolling the hostile area. Artillery is used to weaken the enemy defences allowing transport and attack helicopters to transport and support Special Forces in order to neutralise the guerrilla teams.

OV5 Operational Activity Diagram

The suggested way to model OV5 is by using activity diagrams. Activity diagrams model the flow of events. This way the system behaviour captured in OV-1 can be further analysed, without showing any detail about the implementation and structure of the system.

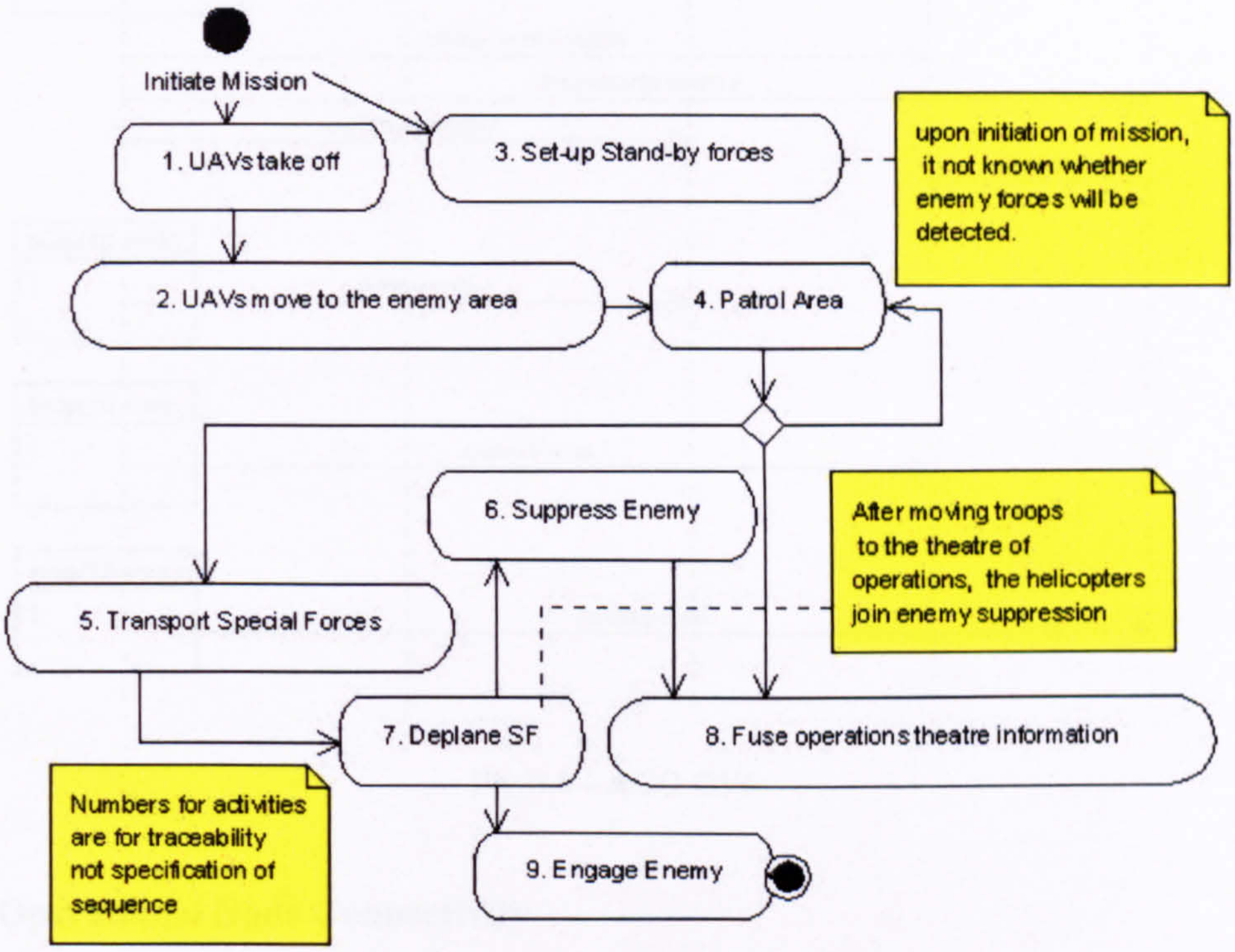

Fig.B.5 – AGO OV5

OV6 Operational Rules Model

Having defined the activities that take place to complete the mission, OV6 is defined which provides a detailed view of mission, including events and messages between the mission entities. As part of OV6 the collaboration diagram for the AGO scenario is provided.

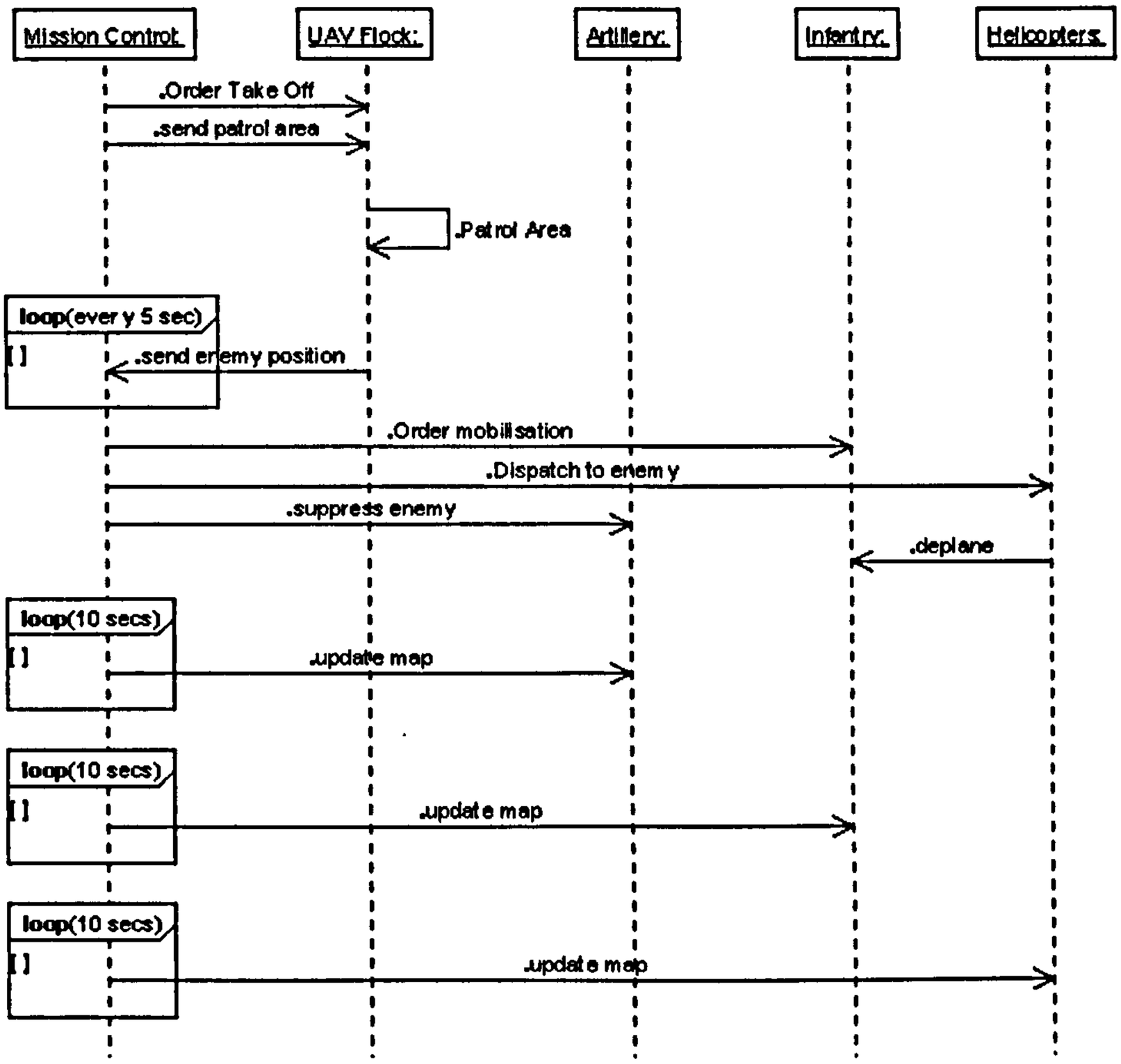

Fig.B.6 – AGO OV6

OV2 Operational Node Connectivity

The suggested representation of this product is made with collaboration diagrams, identifying the entities within the system and the needlines, which show the information exchanges that exist between the participating entities. In the example presented instead of using a collaboration diagram strictly using UML, we have employed a use case to show the information dependencies between the different actors. Merely identifying the "needline links" between the system actors is not adequate to specify this product. We also need to specify what information will be 'carried' by the needline as well as the consumer and provider of the information.

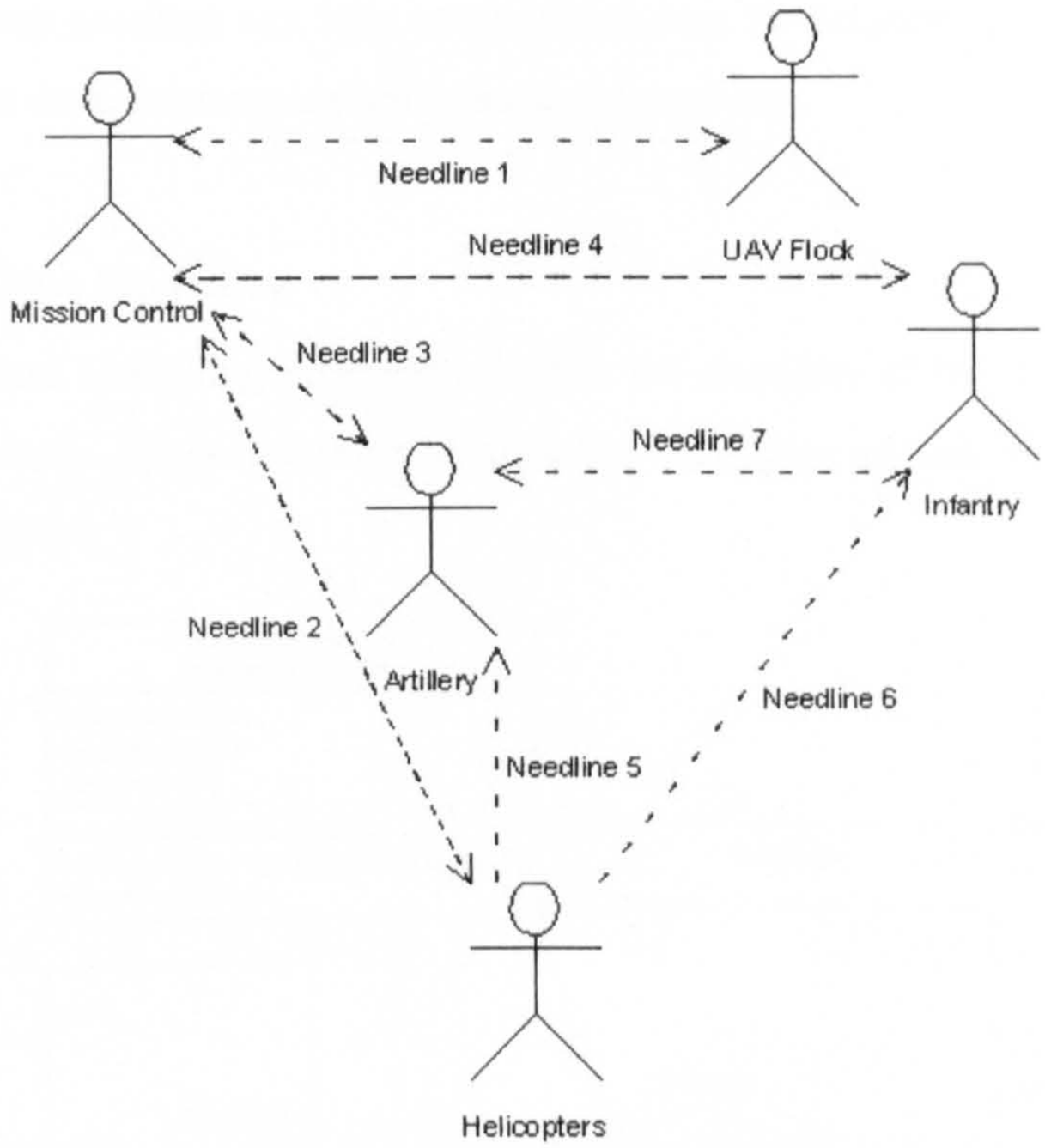

Fig.B.7 – AGO OV2

Table.B.5 – AGO OV-2 (Needline Description)

Needline ID	Information Exchange	Producer	Consumer
1	Flight path	Mission Control	UAV Flock
	(Fuse) sensor data	UAV Flock	Mission control
2	Target Area	Mission Control	Helicopters
	(Fuse) sensor data	Helicopters	Mission control
3	Target area	Mission control	Artillery
4	Target area	Mission Control	Infantry
	Map	Mission Control	Infantry
	(Fuse) feedback	Infantry	Mission Control
5	Map	Mission Control	Helicopters
	Target area	Mission Control	Helicopters
	(Fuse) feedback	Helicopters	Mission Control
6	Theatre Support	Helicopters	Infantry
7	Target location	Infantry	Artillery

OV2 is a view of particular interest as it specifies at the highest level the interconnections and sharing of information that is required by each of the SoS elements. This product can be very useful when conducting analysis associated with the safety or dependability of the system. Each of the needlines serves a purpose regarding the successful completion of the scenario. Identification of the criticality of the services

provided by each needline can help in understanding the required integrity for system features such as data, communication links, and interfaces.

OV7 Logical Data Modelling

The Logical Data Modelling product describes the structure of the data types that are used by the system. Class diagrams identify the data types as well as the relationships between the entities.

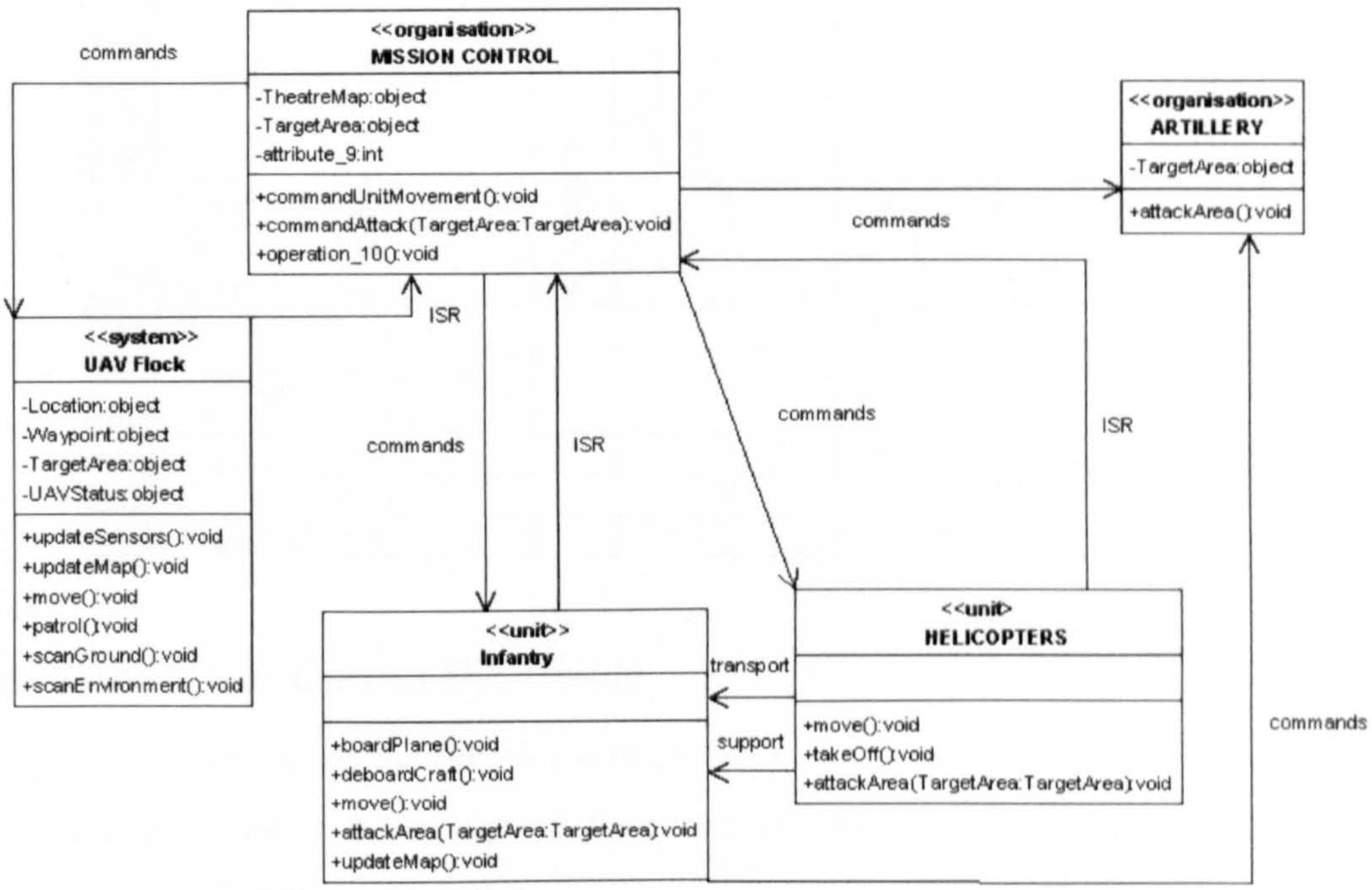

Fig.B.8 – AGO OV7

SV5 Operational Activity to Systems Function Traceability Matrix

SV5 is one of the products that provide a link between the operational and systems view. This is achieved by analysing the defined activity for the scenario (as described by OV5), identifying which system (or organisation) participate during each of the activities. This allows determining how the individual entities contribute in achieving the overall functionality (or capability). SV5 cannot be specified only from the activity diagram, but it also requires knowledge of what functions each of the system can provide. Hence, based on the functions that a system provides the developers examine which systems contribute to an activity and how. An example Activity to Systems Function Matrix for the AGO scenario is presented below. The table also show a

possible decomposition of the Mission Control entity into two subsystems, one for decision making and the other responsible for communications and fusing the received data. As more detail is added the class diagrams should be updated accordingly.

Table.B.6 – AGO SV5

Activity ID		Mission Control		UAV	Infantry	Helicopters	Artillery
		C4	Comms & Fusion				
1	UAV Take-off	X	X	X			
2	Guidance of UAVs	X	X	X			
3	Set up of forces	X			X	X	X
4	Patrol Enemy Area			X			
5	Transport Special Forces				X	X	
6	Deplane Special Forces				X	X	
7	Suppress Enemy			X		X	X
8	Fuse information		X	X	X	X	
9	Engage Enemy				X		X

SV4 Systems Functionality Description

This product documents the functionality of the systems participating in the scenario. A use case provides a hierarchy of functionality showing the relationship between functionality using the <<include>> association.

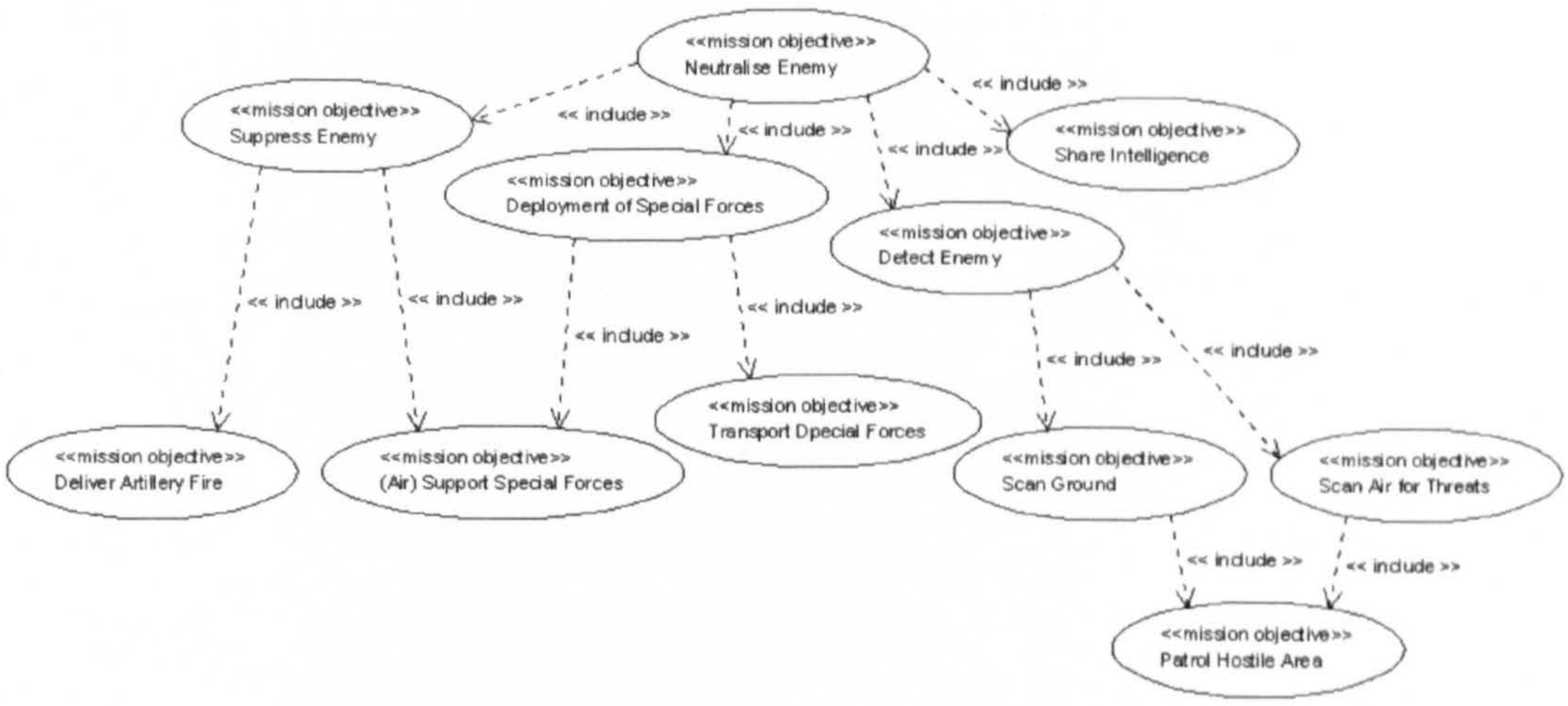

Fig.B.9 – AGO SV4

Furthermore a class diagram can be employed identifying the functions (methods) of each class, as well as the data flows in between them. Fig.B.9 shows the use case diagram for this product. As class diagram the OV7 product can be used to depict the methods of each of the entities and their data interconnections. The class diagram abstracts detail of the exact data flow between the systems.

5. Summary

This section presents a brief exercise, the objective of which was to develop the AGO example according to DODAF, using UML as modelling basis. The main focus was on the operational view since it is the one in which the overall dependability operation of the system can be analysed, showing all the characteristics distinct to SoS such as geographic distribution, collaboration and data sharing. The operational view is the main view for requirements elicitation as it can be used to associate and analyse the system behaviour with the overall scenario objectives. Furthermore of particular interest is the decomposition of the SoS level to the individual systems (elements) level, which is achieved using the products SV4 and SV5 maintaining the traceability of requirements (functional and non-functional) to systems level.

Intentionally Blank

Appendix C

DCM & EOL Code

The appendix presents the Dependability Case Metamodel as defined using KM3. Moreover some of the scripts are presented whish were used for the creation and analysis of the dependability case. Due to the fact that EOL is still an experimental language there cannot be an integrated way in which the scripts are applied on the model. Instead the user needs to know which script needs to be run at each stage of the dependability case. At the time this thesis is written the author is migrating the code to a newer version of EOL which supports these features. The scripts in this appendix are presented as code samples.

1. The Dependability Case Metamodel

```
1 package DC {
2
3    class Case extends Package {
4    }
5
6    class ArgumentModule extends PackageableElement, Package {
7            reference goal : Goal;
8    }
9
10   class SolvedBy {
11               reference parent : SpinalElement oppositeOf solvedBy;
12               reference child container : SpinalElement;
13               attribute cardinality : String;
14               attribute optional : Boolean;
15   }
16
```

```
17    class InContextOf {
18                reference parent : SpinalElement oppositeOf inContextOf;
19                reference child container : ContextualElement;
20                attribute cardinality : Integer;
21     }
22
23    abstract class SpinalElement extends ModelElement {
24          reference solvedBy [*] container : SolvedBy oppositeOf parent;
25          reference inContextOf [*] container : InContextOf oppositeOf parent;
26    }
27
28    abstract class LeafSpinalElement extends SpinalElement {
29    }
30
31    class ReferenceSpinal extends LeafSpinalElement {
32          reference pointsTo : SpinalElement;
33    }
34
35    abstract class ContextualElement extends ModelElement {
36          attribute contextStatus : ContextStatus;
37    }
38
39     class ReferenceContextual extends ContextualElement {
40          reference pointsTo : ContextualElement;
41    }
42
43    class Goal extends SpinalElement, PackageableElement {
44          attribute goalStatus : GoalStatus;
45          attribute isGoalPublic : IsGoalPublic;
46          }
47
48    class AwayGoal extends LeafSpinalElement, ContextualElement {
49          reference argumentModule : ArgumentModule;
50     }
```

```
51
52      class Solution extends LeafSpinalElement {
53              attribute solutionStatus : SolutionStatus;
54      }
55
56      class Strategy extends SpinalElement{
57              attribute strategyStatus : StrategyStatus;
58      }
59
60      class Context extends ContextualElement {
61      }
62
63      class AwayContext extends ContextualElement {
64          reference argumentModule : ArgumentModule;
65      }
66
67      class Bounds extends ContextualElement {
68          attribute target : String;
69          attribute limit : String;
70      }
71
72      class Justification extends ContextualElement {
73      }
74
75          class ModuleSolution extends LeafSpinalElement {
76              reference argumentModule : ArgumentModule;
77          }
78
79          class ModuleContract extends ArgumentModule {
80          }
81
82          class ContractSolution extends LeafSpinalElement {
83              reference moduleContract : ModuleContract;
84          }
```

```
85
86      class InheritedContext extends ContextualElement {
87          reference inherits [*] container : ReferenceContextual;
88      }
89
90      class Factor extends ModelElement {
91              attribute approachOnFactor : String;
92              reference affects [*] : Goal;
93 reference inFactorInstance [*] container : FactorInstance oppositeOf
94      fromFactorInstance;
95          }
96
97          class FactorInstance extends ModelElement {
98              reference impactsOnGoal [*] container : ImpactOnGoal;
99              reference fromFactorInstance : Factor oppositeOf inFactorInstance;
100         }
101
102         class Decision extends PackageableElement {
103             reference influencedBy [*] : FactorInstance;
104             reference decisionImpactOnGoal [*] container : DecisionImpactOnGoal;
105     }
106
107     class DesignDecision extends Decision {
108             reference systemElements [*] : SystemElement;
109     }
110
111     class ImpactOnGoal extends ModelElement {
112             reference goal : Goal;
113             attribute typeOfImpact : TypeOfImpact;
114             attribute magnitudeOfImpact : MagnitudeOfImpact;
115     }
116
117     class DecisionImpactOnGoal extends ModelElement {
118             attribute tolerabilityClassification : TolerabilityClassificaiton;
```

```
119             reference goal : Goal;
120             reference statementOfImpact : ArgumentModule;
121     }
122
123
124     class DependabilityAttribute extends PackageableElement {
125             reference concern [*] container : Concern oppositeOf concernOfAttribute;
126             reference typicalIssue [*] container : Issue oppositeOf issueOfAttribute;
127     }
128
129     class Concern extends PackageableElement {
130             reference addressedInGoal : Goal;
131             reference concernOfAttribute : DependabilityAttribute oppositeOf concern;
132             reference causedByFailureCondition [*] : FailureCondition oppositeOf
133                     compromisesConcern;
134     }
135
136     class Issue extends ModelElement {
137             reference issueOfAttribute : DependabilityAttribute oppositeOf typicalIssue;
138             reference revealedBySuitableDeviation [*] : SuitableDeviation oppositeOf
139     revealsIssue;
140     }
141
142     class Guideword extends PackageableElement {
143     }
144
145     class SystemModel extends PackageableElement {
146             reference containsSystemElement [*] : SystemElement oppositeOf
147                     elementAppearsInModel;
148     }
149
150     class SystemElementType extends PackageableElement {
151             reference systemElements [*] : SystemElement oppositeOf isOfSystemElementType;
152             reference suitableDeviation [*] : SuitableDeviation oppositeOf systemElementType;
```

```
153         }
154
155         class SystemElement extends PackageableElement {
156                 reference appearsInDeviation [*] : ApplicableDeviation oppositeOf
157                         deviationOnSystemElement;
158                 reference elementAppearsInModel [*] : SystemModel oppositeOf
159                         containsSystemElement;
160                 reference hasDependabilityProfile container : DependabilityProfile oppositeOf
161                         systemElement;
162                 reference isOfSystemElementType : SystemElementType oppositeOf
163                         systemElements;
164         }
165
166         class SystemTask extends PackageableElement {
167         }
168
169         class TaskIssue extends PackageableElement {
170                 attribute taskIssueIsAConcern : TaskIssueIsAConcern;
171                 attribute aggregated : Aggregated;
172                 reference systemTask : SystemTask;
173                 reference issue : Issue;
174         }
175
176         abstract class Deviation extends PackageableElement {
177                 reference guideword : Guideword;
178         }
179
180         class SuitableDeviation  extends Deviation {
181                 reference revealsIssue [*] : Issue oppositeOf revealedBySuitableDeviation;
182                 reference systemElementType : SystemElementType oppositeOf suitableDeviation;
183                 attribute deviationIsApplicable : DeviationIsApplicable;
184         }
185
186         class ApplicableDeviation extends Deviation {
```

```
187             reference manifestsAsFailureCondition  : FailureCondition oppositeOf
188                     causedByDeviation;
189             reference deviationOnSystemElement : SystemElement oppositeOf
190                     appearsInDeviation;
191     }
192
193     class FailureCondition extends PackageableElement {
194     attribute effect : String;
195     reference causedByDeviation : ApplicableDeviation oppositeOf
196             manifestsAsFailureCondition;
197     reference effectTrace [*] container : TraceabilityOfEffect oppositeOf
198             traceBelongsToFailureCondition;
199     reference causedByFailureConditionEffectTrace [*] : TraceabilityOfEffect
200             oppositeOf resultsToFailureCondition;
201     reference compromisesConcern  [*] : Concern oppositeOf
202             causedByFailureCondition;
203     reference dependabilityRequirement : DependabilityRequirement oppositeOf
204             failureCondition;
205     }
206
207     abstract class TraceabilityOfEffect extends ModelElement {
208             reference traceBelongsToFailureCondition : FailureCondition oppositeOf
209                     effectTrace;
210             reference resultsToFailureCondition [*] : FailureCondition oppositeOf
211                     causedByFailureConditionEffectTrace ;
212     }
213
214     class TextualTraceabilityOfEffect extends TraceabilityOfEffect {
215     }
216
217     class DependabilityProfile extends ModelElement {
218             reference systemElement : SystemElement oppositeOf hasDependabilityProfile;
219             reference dependabilityRequirement [*] container : DependabilityRequirement;
220     }
```

```
221
222     class DependabilityRequirement extends ModelElement {
223             reference addressedByGoal : Goal;
224             reference failureCondition : FailureCondition oppositeOf dependabilityRequirement;
225     }
226     abstract class ModelElement {
227             attribute description : String;
228             attribute name : String;
229             reference taggedValue [*] container : TaggedValue;
230         }
231
232 class TaggedValue {
233         attribute key : String;
234         attribute value : String;
235 }
236
237 class Package extends PackageableElement {
238         reference content [*] container : PackageableElement;
239 }
240
241 abstract class PackageableElement extends ModelElement {
242 }
243
244         datatype String;
245         datatype Boolean;
246         datatype Integer;
247
248         enumeration TypeOfImpact {
249             literal NotDefined;
250             literal Negative;
251             literal Positive;
252     }
253
254     enumeration MagnitudeOfImpact {
```

```
255             literal NotDefined;
256             literal High;
257             literal Medium;
258             literal Low;
259     }
260
261     enumeration Severity {
262             literal Catastrophic;
263             literal Critical;
264             literal Marginal;
265             literal Negligible;
266     }
267
268     enumeration GoalStatus {
269             literal Normal;
270             literal ToBeDeveloped;
271             literal ToBeInstantiated;
272             literal SolvedByContract;
273             literal ToBeDevelopedAndInstantiated;
274     }
275
276
277     enumeration ContextStatus {
278             literal Normal;
279             literal ToBeInstantiated;
280     }
281
282     enumeration SolutionStatus {
283             literal Normal;
284             literal ToBeInstantiated;
285     }
286
287     enumeration StrategyStatus {
288             literal Normal;
```

```
289                 literal ToBeDeveloped;
290                 literal ToBeInstantiated;
291         }
292
293         enumeration IsGoalPublic {
294                 literal False;
295                 literal True;
296         }
297
298         enumeration DeviationIsApplicable {
299                 literal NotDefined;
300                 literal False;
301                 literal True;
302         }
303
304         enumeration DeviationSeverity {
305                 literal NotDefined;
306                 literal Minor;
307                 literal Substantial;
308                 literal Major;
309         }
310
311         enumeration TolerabilityClassificaiton {
312                 literal Unconstrained;
313                 literal Probable;
314                 literal Potential;
315                 literal Hesitant;
316                 literal Ineligible;
317         }
318
319         enumeration TaskIssueIsAConcern {
320                 literal NotDefined;
321                 literal ItIsConcernExistingInItsOwnRight;
322                 literal ItIsAFailureCondition;
```

```
323         }
324
325         enumeration Aggregated {
326                 literal NotDefined;
327                 literal True;
328                 literal False;
329         }
330 }
```

2. Samples of Scripts Used in the Dependability Case

Creation of Task Issues from Typical Issues and Tasks

```
1   ---------------------------------
2   ---------VARIABLES----------------
3   ---------------------------------
4   def systemElementsInDC : Sequence;
5   def issues : Sequence;
6   def tasks : Sequence;
7   def packages : Sequence;
8   def systemTasks : Sequence;
9
10
11  ---------------------------------
12  -----INITIALISATION OF VARIABLES --------
13  ---------------------------------
14  systemElementsInDC := SystemElement.allInstances();
15  issues := Issue.allInstances();
16  systemTasks := SystemTask.allInstances();
17  packages := Package.allInstances();
18  ---------------------------------
19
20
21  ---------------------------------
```

```
22  ----------MAIN PROGRAM----------
23  --COUNT SYSTEM ELEMENTS
24  (systemElementsInDC.size() + ' Total System Elements')-> println();
25  (issues.size() + ' Specified Typical Issues')-> println();
26  (systemTasks.size() + ' Overall System Tasks')-> println();
27
28  def tipackage : new Package;
29  tipackage.name := 'Task Issues';
30  packages.select(p|p.name='DDA').first().content.add(tipackage);
31  def taskIssueCounter : Integer;
32  --tipackage.println();
33  for (i in issues) {
34          for (st in systemTasks) {
35                  def ti : new TaskIssue;
36                  taskIssueCounter := taskIssueCounter + 1;
37                  ti.name := i.name + ':' + st.description;
38                  ti.issue := i;
39                  ti.systemTask := st;
40                  ti.taskIssueIsAConcern := TaskIssueIsAConcern#NotDefined;
41                  tipackage.content.add(ti);
42          }
43  }
44  ('Created ' + taskIssueCounter + ' Issue Instances')->println();
45  DC.store(DC.modelFile.replace('ecore', 'copy.ecore'));
46  ------------------------------
47
```

Creation of Deviations

```
1  ------------------------------
2  ---------VARIABLES-----------
3  ------------------------------
4  def systemElementsInDC : Sequence;
5  def packages : Sequence;
6  def issues : Sequence;
```

```
7   def suitableDeviations : Sequence;
8   def guidewords : Sequence;
9   def errorCount : Integer;
10  def warningCount : Integer;
11  def suppressWarnings : Boolean;
12  def createApplicableDeviations : Boolean;
13  def loopCounter : Integer;
14
15  ----------------------------------
16  -----INITIALISATION OF VARIABLES --------
17  ----------------------------------
18  systemElementsInDC := SystemElement.allInstances();
19  issues := Issue.allInstances();
20  packages := Package.allInstances();
21  suitableDeviations := SuitableDeviation.allInstances();
22  guidewords := Guideword.allInstances();
23  ----------------------------------
24  -----CUSTOMISATION VARIABLES-----------
25  ----------------------------------
26  suppressWarnings := false;
27  createApplicableDeviations := true;
28
29  ----------------------------------
30  ----------MAIN PROGRAM-------------
31  'Starting deviations checking.....'->println();
32  'Opening File...'->println();
33  (DC.modelFile)->println();
34  'Done.'->println();
35  ''->println();
36
37  warningCount := 0;
38  'Examining deviations....'->println();
39  'Checking issues for suitable deviations...'->println();
40  for (i in issues ){
```

```
41          if ( i.revealedBySuitableDeviation.size()=0) {
42                  if (not suppressWarnings) {
43                          ('   #WARNING-Issue "' + i.name + '", has not been associated with any
44  suitable deviation')->println();
45                  }
46                  warningCount := warningCount + 1;
47          }
48  }
49  'Issues checked !!'->println();
50  "->println();
51
52  'Checking suitable deviations...'->println();
53  def errorFound : Boolean;
54  --errorFound is a flag indicating that there is an error in the definition of a suitable deviation
55  --if an error is found the script will not attempt to create name for the sd
56
57  for (sd in suitableDeviations) {
58          def errorFound : Boolean;
59          errorFound :=false;
60          if (sd.systemElementType.size() = 0) {
61                  ('   #Error-Suitable deviation "' + sd.name + 'has not been associated with any System
62  Elements.')->println();
63                  errorCount:= errorCount + 1;
64                  errorFound := true;
65          }
66          if (sd.guideword.size() = 0) {
67                  ('   #Error-Suitable deviation "' + sd.name + 'has not been associated with a
68  Guideword.')->println();
69                  errorCount:=errorCount +1;
70                  errorFound := true;
71          }
72
73          if (errorFound = false) {
74          --if the suitable deviation is defined correctly create name
75          --and check wether applicability has been defined
```

```
76                      sd.name := sd.guideword.name + '::' + sd.systemElementType.name;
77                              if (sd.deviationIsApplicable.name = 'NotDefined') {
78                                      if (not suppressWarnings) {
79                                              ('   #WARNING-Define if suitable deviation  "' + sd.name +
80  '" is applicable')->println();
81                                      }
82                                      warningCount := warningCount + 1;
83                              }
84              }
85              --sd.println();
86  }
87  'Suitable Deviations checked !!'->println();
88  'Storing Model...'->print();
89  DC.store();
90  'Done!'->println();
91
92  if (createApplicableDeviations) {
93              ''->println();
94              'Creating Applicable Deviations...'->println();
95              ('Found ' + suitableDeviations.size() + ' suitable deviations for the attribute issues specified.')-
96  >println();
97              loopCounter := 0;
98              for (sd in suitableDeviations) {
99                      if (sd.deviationIsApplicable.name = 'True') {
100                             loopCounter := loopCounter + 1;
101                     }
102             }
103             ('   of which ' + loopCounter + 'have been defined as applicable.')->println();
104             def adPackage : new Package;
105             packages.select(p|p.name='Deviations').first().content.add(adPackage);
106             adPackage.name := 'Applicable Deviations';
107             --create a new package below the package deviations to add the applicable deviations
108             loopCounter :=0;
109             for (sd in suitableDeviations) {
110                     if (sd.deviationIsApplicable.name = 'True') {
```

```
111                 for (se in sd.systemElementType.systemElements) {
112                         --for every type instance associated with a suitable deviation
113                         def ad : new ApplicableDeviation;
114                         ad.name := sd.guideword.name + '::' +se.name;
115                         --ad.name->println();
116                         ad.deviationOnSystemElement := se;
117                         ad.guideword := sd.guideword;
118                                 loopCounter := loopCounter + 1;
119                                 adPackage.content.add(ad);
120                             }
121                     }
122             }
123             ('Created ' + loopCounter + ' applicable deviations')->println();
124     }
125
126
127     ".println();
128     DC.store(DC.modelFile.replace('ecore', '2.ecore'));
129     ('Storing Model as: ')->println();
130     (' ' + DC.modelFile.replace('ecore', '2.ecore')) ->println();
131     ".println();
132
133     (warningCount + ' Warnings. ')->println();
134     (errorCount + ' Errors.')->println();
135     if (suppressWarnings) {'To see the warnings change the "suppressWarnings" variable in the script.'-
136     >println();}
137
138     '** FINISHED **'->println();
139     ------------------------------
140
```

Creation of Failure Conditions

```
1   ------------------------------
2   ---------VARIABLES--------------
3   ------------------------------
4   def applicableDeviations : Sequence;
5   def packages : Sequence;
6   def failureConditions : Sequence;
7
8   def errorCount : Integer;
9   def warningCount : Integer;
10  def loopCounter : Integer;
11
12
13  ----------------------------
14  applicableDeviations := ApplicableDeviation.allInstances();
15  packages := Package.allInstances();
16  failureConditions := FailureCondition.allInstances();
17  ----------------------------
18
19
20  --create a package to add failure conditions
21  def fcPackage : new Package;
22          packages.select(p|p.name='DDA').first().content.add(fcPackage);
23          fcPackage.name := 'Failure Conditions';
24  'Created Failure Conditions Package under DDA'->println();
25
26  --create a failure condition ofr every deviation
27  loopCounter :=1; --counter will be used to ID failure conditions
28  for (ad in applicableDeviations) {
29      --check if there already is one
30      if (ad.manifestsAsFailureCondition.isDefined()){
31          fcPackage.content.add(ad.manifestsAsFailureCondition);
32          loopCounter := loopCounter + 1;
33      }
34      if (not ad.manifestsAsFailureCondition.isDefined()){
```

```
35                  def fc : new FailureCondition;
36                  fc.causedByDeviation := ad;
37                  fc.name := 'FC' + loopCounter;
38                  fcPackage.content.add(fc);
39                  loopCounter := loopCounter + 1;
40          }
41  }
42
43  ('Created ' + (loopCounter-1) + ' failure conditions')->println();
44
45  "->println();
46  'Combining failures...'->println();
47  for (fc in failureConditions) {
48          fc.println();
49          fc.causedByDeviation.guideword.println();
50          if (fc.causedByDeviation.guideword = 'omission') {
51          }
52  }
53  '***  FINISHED  ***'->println();
54
```

Creation of GraphViz File for Failure Map

```
1   def failureConditions : Sequence;
2   def deviations : Sequence;
3   def systemElements : Sequence;
4   def traces : Sequence;
5   def applicableDeviations : Sequence;
6
7   def out : String;
8   def shapeCase : String;
9   def count : Integer;
10
11  failureConditions := FailureCondition.allInstances();
```

```
12  applicableDeviations := ApplicableDeviation.allInstances();
13  systemElements := SystemElement.allInstances();
14  traces := TraceabilityOfEffect.allInstances();
15
16
17
18  ----------
19  ----------MAIN----------
20
21          'digraph G {'.println();
22          --begin digraph
23          'Dependability Deviation Analysis Automated Graph Extraction'->println();
24          'Process started!'->println();
25          'Digraph created!'->println();
26          ''->println();
27          'ranksep = 2'.println();
28          'nodesep = 0.7'.println();
29
30          ('Identified ' + failureConditions.size() + ' failure conditions.')->println();
31          ('Identified ' + applicableDeviations.size() + ' deviations.')->println();
32          ('Identified ' + systemElements.size() + ' system elements.')->println();
33          ('Identified ' + traces.size() + ' associations.')->println();
34          ''->println();
35
36
37          'Drawing failure conditions'->println();
38
39          for (fc in failureConditions) {
40                  shapeCase := 'rectangle';
41                  ('fc' + failureConditions.indexOf(fc) + '[ ' + 'shape = ' + shapeCase + ', label= '' +
42  fc.name +'\\n' +  (fc.description).formatText() + ' '' + ' ,color = ' + ((fc.effectTrace).size()).colourFC() + '
43  , style = filled ' + ' ]; ').println();
44          }--endfor
45
46          ' '->println();
```

```
47
48              'Drawing deviations'->println();
49              for (d in applicableDeviations.select(d|d.manifestsAsFailureCondition.isDefined())) {
50                      ("" + d.name + "" + '[color = ' +
51  (d.deviationOnSystemElement.isOfSystemElementType.name).colourSystemElement() + ', fontcolor = '
52  + (d.guideword.name).colourDeviation() + ']' ).print();
53                      '.'->print();
54                      --d.guideword.name->println();
55              } --endfor
56
57              ' '->println();
58              'Drawing failure condition associations'->println();
59              for (fc in failureConditions) {
60              --fc.name->println();
61                      for (et in fc.effectTrace.select(et|et.resultsToFailureCondition.isDefined())) {
62                      --(' '+et.name)->println();
63                              for (fc2 in et.resultsToFailureCondition) {
64                              '.'->print();
65                                      ('fc' + failureConditions.indexOf(fc) + ' -> ' + 'fc' +
66  failureConditions.indexOf(fc2)).println();
67                                      --(' '+fc2.name)->println();
68                                      }
69                      }--end for
70              }--end for
71              "->println();
72
73
74              'Drawing deviation associations'->println();
75              for (d in applicableDeviations.select(d|d.manifestsAsFailureCondition.isDefined())) {
76                      ("" + d.name + "" -> fc' +
77  failureConditions.indexOf(d.manifestsAsFailureCondition)).println();
78                      '.'->print();
79                      --d.guideword.name->println();
80              } --endfor
81
```

```
82              '}'.println();
83              "->println();
84              'Digraph closed!'->println();
85
86      out.store(DC.getModelFile().replace('.ecore','.DDA.col.graph'));
87      'DDA.graph file created!'->println();
88      'Process complete'->println();
89      "->println();
90      ----------------------------
91      ----------------------------
92      operation Any println(){
93              out := out + self + '\n';
94      }
95
96      operation Any print(){
97              out := out + self;
98      }
99
100     operation String colourSystemElement () : String {
101             def inString : String;
102             def outString : String;
103
104             --inString := self.toCharSequence();
105
106             inString :=self;
107
108             if ( inString = 'Information Exchange') {
109                     outString := 'forestgreen';
110             }
111             if (not (inString = 'Information Exchange')) {
112                     outString := 'black';
113             }
114             return outString;
115     }
```

```
116
117
118  operation String colourDeviation () : String {
119          def inString : String;
120          def outString : String;
121
122          --inString := self.toCharSequence();
123              inString :=self;
124
125      if (inString = 'Omission') {
126              outString := 'deepskyblue2';
127      }
128      if (inString = 'Early') {
129              outString := 'darkorange4';
130      }
131      if (inString = 'Less') {
132              outString := 'firebrick';
133      }
134      if (inString = 'Late') {
135              outString := 'goldenrod';
136      }
137      if (inString = 'Fake') {
138              outString := 'dodgerblue1';
139      }
140      if (inString = 'Value') {
141              outString := 'darkorchid2';
142      }
143      if (inString = 'Public') {
144              outString := 'gray';
145      }
146      if (inString = 'Damage') {
147              outString := 'cornsilk3';
148      }
149      return outString;
```

```
150     }
151
152     operation Integer colourFC () : String {
153             def outString : String;
154
155             if (self = 0) {
156                     outString := 'green3';
157             }
158             if (self = 1) {
159                     outString := 'olivedrab3';
160             }
161             if (self = 2) {
162                     outString := 'yellow1';
163             }
164             if (self = 3) {
165                     outString := 'orange1';
166             }
167             if (self >= 4) {
168                     outString := 'orangered1';
169             }
170             return outString;
171     }
172
173
174     operation String formatText () : String {
175             --formats the text according to the number of words and the length of the sentence
176
177             def length : Integer;
178             def charCounter : Integer;
179             def inString : Sequence;
180             def wordCount : Integer;
181             def outString : String;
182             def wordPointer : Integer;
183
```

```
184             '.'->print();
185             --TEST-- self->println();
186             inString := self.toCharSequence();
187             length := inString.size();
188             --TEST-- length->println();
189             wordCount :=1; --Start from the 1st word not 0th
190             --count the words
191             --TEST-- 'Counting Words...'->println();
192             for (c in Sequence{0..length} )  {
193                     if (inString.at (c+1) = ' ') {
194                             wordCount := wordCount + 1;
195                             --TEST-- 'Space found'->println();
196                     }
197             }
198             --TEST-- ('# of words' + wordCount)->println();
199
200             --Formatting
201             if (wordCount < 10 and length > 25 ) {
202                     for (b in Sequence{0..length} )  {
203                             outString := outString + inString.at(charCounter);
204                             if (inString.at(charCounter+1) = ' ') {
205                             wordPointer := wordPointer + 1;
206                             --TEST-- ('Worpointer' + wordPointer)->println();
207                                     if (wordPointer = 4) {
208                                             wordPointer := 0;
209                                             outString := outString + '\\n';
210                                             --if a space is found jump to the next char by increasing the
211     counter
212                                             charCounter := charCounter + 1;
213                                     }
214                             }
215                     charCounter := charCounter + 1;
216                     }
217             }
218
```

```
219
220             if (wordCount >= 10 ) {
221             for (b in Sequence{0..length} )  {
222                     outString := outString + inString.at(charCounter);
223                     if (inString.at(charCounter+1) = ' ') {
224                     wordPointer := wordPointer + 1;
225                     --TEST-- ('Worpointer' + wordPointer)->println();
226                             if (wordPointer = 5) {
227                                     wordPointer := 0;
228                                     outString := outString + ' \\n';
229                                     --if a space is found jump to the next char by increasing the
230     counter
231                                     charCounter := charCounter + 1;
232                             }
233                     }
234             charCounter := charCounter + 1;
235             }
236     }
237
238             if (wordCount < 10 and length < 25 ) {
239             --this is for shapes with small words; we don't want to break them
240             for (b in Sequence{0..length} )  {
241                     outString := outString + inString.at(charCounter);
242                     if (inString.at(charCounter+1) = ' ') {
243                     wordPointer := wordPointer + 1;
244                     --TEST-- ('Worpointer' + wordPointer)->println();
245                             if (wordPointer = 5) {
246                                     wordPointer := 0;
247                                     outString := outString + ' \\n';
248                                     --if a space is found jump to the next char by increasing the
249     counter
250                                     charCounter := charCounter + 1;
251                             }
252                     }
253             charCounter := charCounter + 1;
```

```
254                     }
255             }
256             --TEST-- self->println();
257             return outString;
258     }
259
```

Creation of GSN GraphViz File

```
1       def out : String;
2       def shapeCase : String;
3       def count : Integer;
4       def spinalElements : Sequence;
5       def contextElements : Sequence;
6
7
8       spinalElements := DC!SpinalElement.allInstances().select(se|not se.isKindOf(ReferenceSpinal));
9       contextElements :=DC!ContextualElement.allInstances();
10
11
12      'digraph G { '.println();
13      '               node [shape=record, fontname=Tahoma, fontsize=6];'.println();
14      '               edge [fontname=Tahoma, fontsize=8];'.println();
15      'rankdir = TB;'.println();
16      '               center=true;'.println();
17
18
19      'Formating'->print();
20      for (se in spinalElements) {
21
22              if (se.inContextOf.isDefined()) {
23                      ('Subgraph sameLevelAt' + spinalElements.indexOf(se) + ' {').println();
24                      'rank = same;'.println();
25
26                      --create nodes and shapes for all spinal elements
```

```
27                      if (se.isTypeOf(Goal)) {
28                              shapeCase := 'rectangle';
29                              ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', label= " '
30  +se.description +'\\n' + (se.name).formatText() + ' " ]; ').println();
31                      }
32
33                      if (se.isTypeOf(Strategy)) {
34                              shapeCase := 'parallelogram';
35                                      ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', ' + '
36  label="' +se.description +'\\n' + (se.name).formatText() + '"];').println();
37                      }
38
39                      if (se.isTypeOf(Solution)) {
40                              shapeCase := 'circle';
41                              ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', ' + '
42  label="' +se.description +'\\n' + (se.name).formatText() + '"];').println();
43                      }
44                      --(se.inContextOf)->println();
45                      for ( ce in se.inContextOf) {
46                              ('ce' + contextElements.indexOf(ce.child) + '[ ' + 'shape = octagon , label= " ' +
47  ce.child.description +'\\n' + (ce.child.name).formatText() + ' " ]; ').println();
48                              ('se' + spinalElements.indexOf(se)+ ' -> ' + 'ce' +
49  contextElements.indexOf(ce.child) + ' ;').println();
50                      }
51                      '}'.println();
52              }
53
54              if (not se.inContextOf.isDefined()) {
55
56                      --create nodes and shapes for all spinal elements
57                      if (se.isTypeOf(Goal)) {
58                              shapeCase := 'rectangle';
59                              ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', label= " '
60  +se.description +'\\n' + (se.name).formatText() + ' " ]; ').println();
61                      }
```

```
62
63                      if (se.isTypeOf(Strategy)) {
64                              shapeCase := 'parallelogram';
65                              ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', ' + '
66  label="' +se.description +'\\n' + (se.name).formatText() + '"];').println();
67                      }
68
69                      if (se.isTypeOf(Solution)) {
70                              shapeCase := 'circle';
71                              ('se' + spinalElements.indexOf(se) + '[ ' + 'shape = ' + shapeCase + ', ' + '
72  label="' +se.description +'\\n' + (se.name).formatText() + '"];').println();
73                      }
74
75              }
76  }
77  "->println();
78  'Added Spinal Element Nodes -- OK'->println();
79  --'Formating'->print();
80  --ADDING CONTEXTUAL ELEMENTS
81  --for (ce in contextElements) {
82  --      if (ce.isTypeOf(Context)) {
83  --              shapeCase := 'octagon';
84  --              ('ce' + contextElements.indexOf(ce) + '[ ' + 'shape = ' + shapeCase + ' , label= " '
85  +ce.description +'\\n' + (ce.name).formatText() + ' " ]; ').println();
86  --      }
87
88  --}
89  --"->println();
90  --'Added Context Element Nodes -- OK'->println();
91
92
93
94  ''.println();       ''.println();       ''.println();       ''.println();
95
96  --CONNECTING SPINAL ELEMENTS
```

```
97  for (se in spinalElements) {
98          for (sb in se.solvedBy) {
99                  --sb.child->println();
100                 --print the parent spinal element
101                 ('se' + spinalElements.indexOf(se)).print();
102
103                 --if child is reference point to target
104                 if (sb.child.isTypeOf(ReferenceSpinal)){
105                 (' -> ' + 'se' + spinalElements.indexOf(sb.child.pointsTo) + ' ; ').println();
106                 }
107                 --and if it is not reference print the child index in the spinal elements container
108                 if (not sb.child.isTypeOf(ReferenceSpinal)){
109                         (' -> ' + 'se' + spinalElements.indexOf(sb.child) + ' ; ').println();
110                 }
111         }
112 }
113 'Added spinal connectors --OK'->println();
114
115 '}'.println(); --Close the digraph G {
116
117 -- write file
118 out.store(DC.getModelFile().replace('.ecore','.GSN.graph'));
119
120 operation Any println(){
121         out := out + self + '\n';
122 }
123
124 operation Any print(){
125         out := out + self;
126 }
127
128 operation String formatText () : String {
129         --formats the text according to the number of words
130         -- the point is that the greater the number of words is the more
```

```
131         --words each line should have in order to maintain the ratio
132         --of the shapes
133
134         def length : Integer;
135         def charCounter : Integer;
136         def inString : Sequence;
137         def wordCount : Integer;
138         def outString : String;
139         def wordPointer : Integer;
140
141         '.'->print();
142         --TEST-- self->println();
143         inString := self.toCharSequence();
144         length := inString.size();
145         --TEST-- length->println();
146         wordCount :=1; --Start from the 1st word not 0th
147         --count the words
148         --TEST-- 'Counting Words...'->println();
149         for (c in Sequence{0..length} )  {
150                 if (inString.at (c+1) = ' ') {
151                         wordCount := wordCount + 1;
152                         --TEST-- 'Space found'->println();
153                 }
154         }
155         --TEST-- ('# of words' + wordCount)->println();
156
157         --Formatting
158         if (wordCount < 8 and length > 25 ) {
159                 for (b in Sequence{0..length} )  {
160                         outString := outString + inString.at(charCounter);
161                         if (inString.at(charCounter+1) = ' ') {
162                         wordPointer := wordPointer + 1;
163                         --TEST-- ('Worpointer' + wordPointer)->println();
164                                 if (wordPointer = 3) {
```

```
165                                                     wordPointer := 0;
166                                                     outString := outString + ' \\n';
167                                                     --if a space is found jump to the next char by increasing the
168     counter
169                                                     charCounter := charCounter + 1;
170                                             }
171                                     }
172                                     charCounter := charCounter + 1;
173                                     }
174             }
175
176
177                                     if (wordCount >= 8 ) {
178                                     for (b in Sequence{0..length} )  {
179                                             outString := outString + inString.at(charCounter);
180                                             if (inString.at(charCounter+1) = ' ') {
181                                             wordPointer := wordPointer + 1;
182                                             --TEST-- ('Worpointer' + wordPointer)->println();
183                                                     if (wordPointer = 5) {
184                                                             wordPointer := 0;
185                                                             outString := outString + ' \\n';
186                                                             --if a space is found jump to the next char by increasing the
187     counter
188                                                             charCounter := charCounter + 1;
189                                                     }
190                                             }
191                                     charCounter := charCounter + 1;
192                                     }
193             }
194
195                                     if (wordCount < 8 and length < 25 ) {
196                                     --this is for shapes with small words; we don't want to break them
197                                     for (b in Sequence{0..length} )  {
198                                             outString := outString + inString.at(charCounter);
199                                             if (inString.at(charCounter+1) = ' ') {
```

```
200                     wordPointer := wordPointer + 1;
201                     --TEST-- ('Worpointer' + wordPointer)->println();
202                             if (wordPointer = 5) {
203                                     wordPointer := 0;
204                                     outString := outString + ' \\n';
205                                     --if a space is found jump to the next char by increasing the
206     counter
207                                             charCounter := charCounter + 1;
208                                     }
209                             }
210                     charCounter := charCounter + 1;
211                     }
212             }
213             --TEST-- self->println();
214             return outString;
215     }
```

GSN Constraints Using EVL

```
1     pre {
2             def numberOfFANIN : Integer;
3             def numberOfSolutionFANIN : Integer;
4     }
5
6     post {
7     }
8
9     context AwayGoal {
10
11            constraint HasArgumentModule :
12                    self.argumentModule.isDefined()
13            fail :
14                    'Away goal ' + self.name + ' has no associated argument module'
15
```

```
16          constraint NamesMatch :
17                  self.name = self.argumentModule.name
18  }
19
20  context SolvedBy {
21
22          constraint HasChild :
23                  self.child.isDefined()
24          fail :
25                  'Element ' + self.eContainer().description + ' does not have a solution'
26  }
27
28  context ReferenceSpinal {
29
30          constraint PointsToNonReference :
31                  not self.pointsTo.isTypeOf(ReferenceSpinal)
32          fail :
33                  'Reference Spinal points to another Reference Spinal under: '
34                  + self.eContainer().eContainer().name
35
36      constraint ShouldnotPointToGoal :
37              not self.pointsTo.isTypeOf (Goal)
38      fail {
39              numberOfGoalFANIN := numberOfFANIN + 1;
40              return    'Goal ' + self.eContainer().eContainer().description +
41                                          ' Fans-in to goal ' + self.pointsTo.description;
42      }
43
44      constraint ShouldNotPointToSolution :
45              not self.pointsTo.isTypeOf(Solution)
46      fail {
47              numberOfSolutionFANIN := numberOfSolutionFANIN +1;
48              return 'Fan-in at solution ' + self.pointsTo.description +
49                      ' under goal ' + self.eContainer().eContainer().description;
```

```
50          }
51  }--END OF ReferenceSpinal Constraints
52
53  context Goal {
54
55          constraint HasUniqueName :
56                  Goal.allInstances.forAll(g|g.name = self.name implies g = self)
57          fail :
58                  'Duplicate Goal name: ' + self.name + ' at ' + self.description
59
60          constraint HasUniqueDescription :
61                  Goal.allInstances.forAll(g|g.description = self.description implies g = self)
62          fail:
63                  'Goal ' + self.description + ' has not unique ID'
64
65
66  }-- END GOAL CONSTRAINTS
67
68  context LeafSpinalElement {
69
70          constraint IsNotDecomposed :
71                  self.solvedBy.size() = 0
72          fail :
73                  'Goal solution ' + self.description + ' is illegally decomposed'
74
75  }
```

GraphViz Failures Map for the AGO

```
digraph G {
fc0[ shape = rectangle, label= " FC1\nDisclosure of aircraft position "  ,color = green3 , style = filled  ];
fc1[ shape = rectangle, label= " FC2\nSlow transmission of target \ndata "  ,color = yellow1 , style = filled  ];
fc2[ shape = rectangle, label= " FC3\nArtillery receive fake target \ndata with malicious intent "  ,color = orange1 , style = filled  ];
fc3[ shape = rectangle, label= " FC4\nEnemy receives firing intent \nand target information "  ,color = green3 , style = filled  ];
fc4[ shape = rectangle, label= " FC5\nArtillery will not receive \nany target data "  ,color = orangered1 , style = filled  ];
fc5[ shape = rectangle, label= " FC6\nArtillery will receive the \nwrong location/order "  ,color = yellow1 , style = filled  ];
fc6[ shape = rectangle, label= " FC7\nDelay or possibly loss of \nrequest of target order and \ntarget information "  ,color = olivedrab3 , style = filled  ];
fc7[ shape = rectangle, label= " FC8\nThere is no patrolling function \navailable, cannot notify of enemy \nforces "  ,color = olivedrab3 , style = filled  ];
fc8[ shape = rectangle, label= " FC9\nUsers mistakenly identify enemy "  ,color = orange1 , style = filled  ];
fc9[ shape = rectangle, label= " FC10\nA location is not supressed \nwhilst it is being expected \nthat it does "  ,color = olivedrab3 , style = filled  ];
fc10[ shape = rectangle, label= " FC11\nA location is suppressed \nwhen it should not "  ,color = olivedrab3 , style = filled  ];
fc11[ shape = rectangle, label= " FC12\nDelays in supressing enemy "  ,color = olivedrab3 , style = filled  ];
"Public::Needline 1"[color = blue, fontcolor = deeppink4]"Overload::Needline 7"[color = blue, fontcolor = deeppink4]"Fake::Needline 7"[color = blue, fontcolor = deeppink4]"Public::Needline 7"[color = blue, fontcolor = deeppink4]"Omission::Needline 7"[color = blue, fontcolor = deeppink4]"Value::Needline 7"[color = blue, fontcolor = deeppink4]"Late::Needline 7"[color = blue, fontcolor = deeppink4]"Omission::Activity 4"[color = blue, fontcolor = forestgreen]"Mistake::Activity 4"[color = blue, fontcolor = forestgreen]"Mode::Activity 6"[color = blue, fontcolor = forestgreen]"Late::Activity 6"[color = blue, fontcolor = forestgreen]"Public::Needline 1" -> fc0
"Overload::Needline 7" -> fc1
"Fake::Needline 7" -> fc2
"Public::Needline 7" -> fc3
"Omission::Needline 7" -> fc4
```

"Value::Needline 7" -> fc5

"Late::Needline 7" -> fc6

"Omission::Activity 4" -> fc7

"Mistake::Activity 4" -> fc8

"Mode::Activity 6" -> fc9

"Mode::Activity 6" -> fc10

"Late::Activity 6" -> fc11

fc1 -> fc11

fc2 -> fc8

fc2 -> fc10

fc4 -> fc6

fc4 -> fc9

fc4 -> fc11

fc5 -> fc10

fc6 -> fc8

fc6 -> fc9

fc6 -> fc11

fc7 -> fc4

fc8 -> fc4

fc8 -> fc5

}

Intentionally Blank

References

[1] Federal Aviation Authority, *Aeronautical Information Manual : Official Guide to Basic Flight Information and ATC Procedures,* February 2006.

[2] S. Kinnersly, "Whole Airspace ATM System Safety Case – Preliminary Study", *AEAT LD76008/2 Issue 1*, AEA Technology for Eurocontrol. http://dependability.cs.virginia.edu/research/safetycases/EUR_WholeAirspace.pdf Last accessed September 2006.

[3] DoD. Network Centric Warfare. Department of Defence Report to US Congress, 2003. http://www.c3i.osd.mil/NCW Last accessed November 2005.

[4] V. Smirnova, *Multi Agent System for Distributed Data Fusion in Peer-To-Peer Environment*, 11 2002, M.S. Thesis, University of Jyvaskyla.

[5] N. Leveson, P. Allen, and M.A. Storey, "The analysis of a friendly fire accident using a systems model of accidents". In Proceedings of the *20th International System Safety Society Conference* (ISSC 2003), pages 345–357. System Safety Society, Unionville, Virginia, 2002.

[6] R. Alexander, M. Hall-May, *Characterisation of Systems of Systems Failures*. In proceedings of the *22nd International System Safety Conference,* System Safety Society, 2004.

[7] German Federal Bureau of Aircraft Accidents Investigation, Investigation Report AX001-1-2/02 May 2004.

[8] Ministry of Defence, "Safety Management requirements for defence systems", *Defence Standard 00-56 issue 4,* Ministry of Defence, 2005.

[9] J. Laprie, *Dependability: Basic Concepts and Terminology*, Springer-Verlag, ISBN 3-211-82296-8, 1992.

[10] D. Prasad, "Dependable Systems Integration using Measurement Theory and Decision Analysis", PhD Thesis, Department of Computer Science, University of York, UK, 1998.

[11] G. Despotou, R. Alexander, M. Hall-May. *Key Concepts and Characteristics of Systems of Systems (SoS)*. February 2003. Defence and Aerospace Research Partnership (DARP-HIRTS) Public Document

[12] Ministry of Defence, "JSP430 - Ship Safety Management System Handbook," Ministry of Defence January 1996.

[13] R. Seymour, G. D. Sands, A. Grisogono, M. Unewisse, J. Vaughan, R. Baumgart, "Application of Network Centric Warfare Concepts to a Land Air System – experimentation approach", *Land Operations Division*, Defence Science and Technology Organisation, Australian Department of Defence.

[14] V. Kotov, *Systems of Systems as Communicating Structures*, Hewlett Packard Computer Systems Laboratories, 1997. (ww.hpl.hp.com/techreports/97) Last accessed September 2006.

[15] M. W. Maier, "Architecting Principles for Systems of Systems" (http://www.infoed.com/Open/PAPERS/systems.htm) Last accessed October 2006

[16] P. Periorellis, J. E. Dobson, "Organisational Failures in Dependable Collaborative Enterprise Systems", *Journal of Object Technology*, Vol.1, No.3, Special issue: TOOLS USA 2002 proceedings, pages 107 – 117.

[17] J. W. Hollenbach, W. L. Alexander, "Executing the Modelling and Simulation Strategy Making Simulation Systems of Systems a Reality", Proceedings of the *1997 Winter Simulation Conference* (pp. 948 – 954).

[18] T. A. Clare, "The Engineering and Acquisition of Systems of Systems. US Department of Defense", Research *Development and Acquisition Office*, 2000.

[19] M. Kaanich, K. Kanoun, M. Rabah, *Preliminary framework for SoS Dependability Modelling and Evaluation*, LAAS – CNRS, 2001.

[20] D. S. Caffal Michael, "Systems of Systems Design from an Object Oriented Paradigm", Proceedings of *Monterey Workshop: Radical Innovations of Software and Systems Engineering in the Future*, US Army Research Office, 2002.

[21] C4ISR *Architecture Framework*, Version 2.0, Architectures Working Group, US Department of Defence.

[22] S. Jameson, "Architectures for Distributed Information Fusion to Support Situation Awareness on the Digital Battlefield", *Fourth International Conference on Data Fusion, Montreal, Canada, August 7-10, 2001.*

[23] US Department of Defense. Architecture Framework Working Group "DODAF version 1 – Deskbook", Department of Defence 2004.

[24] J. Laprie, A. Avizienis. *Dependability: Basic concepts and terminology,* Springer-Verlag, ISBN 3-211-82296-8, 1992.

[25] B. Littlewood, L. Stringini, "Software Reliability and Dependability: A Roadmap", *Centro for Software Reliability,* City University. (http://www.csr.city.ac.uk) Last accessed July 2006.

[26] J. McDermid, "On Dependability, its measurement and its management" High *Integrity Systems Journal,* 1994, 1(1): 17-26.

[27] A. Villemeur. Reliability, Availability, Maintainability and Safety Assessment. Volume 1. Methods and Techniques. 1992, Wiley.

[28] International Electrotechnical Commission, "IEC 60050-191 International Electrotechnical Vocabulary", Chapter 191: Dependability and Quality of Service".www.iec.ch.

[29] T. Saridakis, V. Issarny, "Developing Dependable Systems Using Software Architecture". In Proceedings of the *1st Working IFIP Conference on Software, Architecture*, pages 83 – 104, February 1999.

[30] H. Thane, "Safe and Reliable Computer Control Systems Concepts and Methods" Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, KTH, Stockholm. ISSN 1400-11791996.

[31] J. D. Lawrence, "Software Reliability and Safety in Nuclear Reactor Protection Systems" Report prepared for the *US Nuclear Regulatory Commission. Lawrence Livermore National Laboratory.*

[32] N. Leveson., Safeware *System Safety and Computers,* Addison Wesley, 1995.

[33] G. J. Pai, "A Survey of Software Reliability Models", Report CS 651: *Dependable Computing.* Department of ECE, University of Virginia, 2002.

[34] E. Jonsson., L. Stromberg, S. Lindskog, On the Functional Relation between Security and Dependability Impairments. New Security Paradigm Workshop 9/99 http://www.nspw.org. Last accessed in October 2006.

[35] SafSec: Integration of Safety and Security Certification. Phase 1 Final Summary Report. S.P1199.43.5 Issue 2. January 2003, Praxis. (http://www.safsec.com).

[36] J. Moffett, D. Eames. The integration of Safety and Security Requirements. Proceedings of Safecomp 1999.

[37] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, "The Architecture Tradeoff Analysis Method" Appeared in the 4th *International Conference on Engineering of Complex Computer Systems* (ICECCS98), August 1998.

[38] M. R. Barbacci, M. H. Klein, C. B. Weinstock, "Principles for Evaluating the Quality Attributes of a Software Architecture", *Technical Report CMU/SEI-96-TR-036, Software Engineering Institute,* Carnegie Mellon University, 1997.

[39] M. Barbacci, S. J. Carriere, H. P. Feiler, R. Kazman, M. Klein, F. H. Lipson, A. T. Longstaff, B. C. Weinstock, "Steps in an Architecture

Tradeoff analysis Method: Quality Attribute Models and Analysis", SEI, Carnegie mellon University, *Technical Report CMU/SEI-97-TR-029*, 1998.

[40] S. Shum, N. Hammond, "Argumentation-Based Design Rationale: From Conceptual Roots to Current Use", *International Journal of Human Computer Studies, Volume 40 (4)*, 1993, pp 603 – 652.

[41] S. Shum S., A. MacLean, V. Bellotti, N. Hammond, "Graphical Argumentation and Design Cognition", *Knowledge Media Institute*, The Open University, 1997, Report: KMI-TR-25.

[42] L. J. SIBYL, "A Tool for Managing Group Decision Rationale", Proceedings *of Conference on Computer Supported Cooperative Work*, 1990 pp. 79-92.

[43] J. Conklin, L. Begeman, *gIBIS: A hypertext tool for exploratory policy discussion*, 1998, ACM 0-89791-282-9/88/0140.

[44] The Health and Safety Executive (HSE), "Reducing Risks Protecting People", HSE Books, Norwich, 2001.

[45] T. Walker, "Tolerability of Risk. Its Use in Nuclear Regulation in the UK", International Committee on Nuclear Technology (ILT) Symposium on Opportunities and Risks of Nuclear Power, April 2001.

[46] UK Railtrack. Engineering Safety Management Issue 3, Yellow Book 3.Railtrack PLC 2000.

[47] T. Kelly, "Arguing Safety, a Systematic Approach to Managing Safety Cases", PhD Thesis, Department of Computer Science, University of York, 1998.

[48] J. McDermid, "Software Safety: Where's the Evidence?" In proceedings *6th Australian Workshop on Industrial Experience with Safety Critical Systems and Software (SCS'01)*.

[49] P. Froome, C. Jones, *Developing Advisory Software to Comply with IEC-61508.* Prepared by Adelard for the Health and Safety Executive. ISBN 0717623041, 2002.

[50] R. Maxion, R. Olszewski. "Improving Software Robustness with Dependability Cases", Digest of Papers 28th Annual International Symposium on *Fault-Tolerant Computing, Munich* 1998, pp. 346 – 355.

[51] R. Maxion, "Measuring Intrusion-Detection Systems", Presented to *The First International Workshop on Recent Advances in Intrusion Detection,* 1998, Louvain-la-Neuve, Belgium.

[52] ASCAD, Adelard Safety Case Development Manual, 1998, Adelard, 3 Coborn Rd., London.

[53] Department for Regional Development, Safety Case Administration Manager, Railway Safety Case Regulations, Transport Division, Belfast.

[54] J. McDermid, Support *for Safety Cases and Safety Arguments using SAM,"* Reliability Engineering and System Safety, 43:111-127, 1994.

[55] T. P. Kelly, *Concepts and Principles of Compositional Safety Cases,* COMSA/2001/1/1 – Research Commissioned by QinetiQ, Dept. of Computer Science, University of York.

[56] Ministry of Defence, Directorate of Standardization, Defence Standard, 00-40 (Part 1)/Issue 4, Reliability and Maintainability (R&M), 1999.

[57] Ministry of Defence, Directorate of Standardization, Defence Standard, 00-42 (Part 2)/Issue 3, Reliability and Maintainability (R&M), 1999.

[58] M. D. Kienzle, "Practical Computer Security Analysis". PhD Thesis. *School of Engineering and Applied Science,* University of Virginia, 1998.

[59] P. Fenelon, A. J. McDermid., "An Integrated Toolset for Software Safety Analysis", Journal of Systems and Software, July 1993.

[60] A. Moore, B. Strohmayer, Visual NRM User's Manual, *Centre for High Assurance Computing*, Naval Research Laboratory, Washington, DC 20375-5320. Report: NRL/FR/5540—00-99502000

[61] Common Criteria Project Organisations. "Common criteria of information technology security evaluation", http://commoncriteriaportal.org. Last accessed on 10 October 2005.

[62] S. Robertson., J. Robertson. *Mastering the Requirements Process,* Addison-Wesley, 1999.

[63] L. Bass, P. Clements, R. Kazman,. *Software Architecture in Practice 1st Edition*, Addison-Wesley, 1998.

[64] E. Kavakli, P. Loucopoulos, "Goal Driven Requirements Engineering: Analysis and Critique of Current Methods", *Information Modeling Methods and Methodologies (Adv.topics of Database Research),* John Krogstie, Terry Halpin and Keng Siau (eds), IDEA Group,pp 102 - 124.

[65] Adelard, ASCE Tool Overview, http://www.adelard.com/web/hnav/ASCE/ Last accessed May 2006.

[66] Eclipse foundation, Getting Started with Eclipse. http://www.eclipse.org/resources/ Last accessed October 2006.

[67] D. Kolovos, R. Paige, and F Polack, "The Epsilon Object Language (EOL)", in Proc. European Conference in *Model Driven Architecture (EC-MDA)* 2006, Bilbao, Spain, July 2006.

[68] E. Gansner, E. Koutsofios, S. North, *Drawing graphs with dot.* http://www.graphviz.org/Documentation.php Last accessed August 2006.

[69] F.Jouault, J. Bézivin, KM3: a DSL for Metamodel Specification: Proceedings of *8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, LNCS 4037, 2006, Bologna, Italy, pages 171-185.

[70] SAE, "Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment ARP 4761", Society for Automotive Engineers, 1996.

[71] Ministry of Defence. "Defence Standard 00-58: HAZOP Studies on Systems Containing Programmable Electronics". 1996.

[72] EUROCONTROL. "Review Of Techniques To Support The EATMP Safety Assessment Methodology", Volume I, EEC Note No. 01/04, Project SRD-3-E1, January 2003.

[73] T. Srivatanakul, "Security Analysis with Deviational Tehniques", PhD Thesis, Department of Computer Science, University of York, 2005.

[74] R. Stephans and T. Warner "System Safety Analysis Handbook", 2nd Edition. System Safety Society, 1997.

[75] T. Kletz, "HAZOP and HAZAN. Identifying and Assessing Process Industry Hazards", Institution of Chemical Engineers, 1992.

[76] D. Pumfrey, "The Principled Design of Computer System Safety Analyses", PhD Thesis, Department of Computer Science, University of York, 1998.

[77] J.P. Rankin., "Sneak Circuit Analysis", *Nuclear Safety*, vol. 14 no. 5, 1973.

[78] S. P. Wilson and J. A. McDermid. "Integrated Analysis of Complex Safety Critical Systems", The Computer Journal 1995 38(10):765-776, ISSN 1460-2067

[79] G. Mauri "Integrating Safety Analysis Techniques, Supporting Identification of Common Cause Failures", PhD Thesis, University of York, YCST-2001-02.

[80] E. G. Amoroso, *Fundamentals of Computer Security Technology, Prentice-Hall*, 1994.

[81] D. Firesmith. "Analyzing the Security Significance of Systems Requirements", SEI, Symposium on Requirements Engineering for Information Security (SREIS 2005), August 29-30, 2005.

[82] C. P. Pfleeger, *Security in Computing,* 3rd Edition, Prentice-Hall, 2002.

[83] Department of Trade and Industry, Development of a Safety Case for the Use of Current Limiting Devices to Manage Short Circuit Currents on Electrical Distribution Networks, Report Number: URN 04/1066.

[84] Ministry of Defence, Directorate of Standardization, Defence Standard, 00-55, Requirements for Safety Related Software in Defence Equipment. August 1997. (Part 1)/Issue 2: Requirements, (Part 2)/Issue 2: Guidance

[85] Ministry of Defence, "Defence Standard 00-25, Human Factors for Designers of Systems", Principles and Process, Ministry of Defence, 2004.

[86] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice 2nd edition*, Addison-Wesley.

[87] M. T. DeGarmo, "Issues Concerning Integration of Unmanned Aerial Vehicles in Civil Airspace", report MP 04W0000323, November 2004.

[88] Institute of Electrical and Electronics Engineers, Recommended Practice for Architectural Description IEEE STD 1471-2000.

[89] MoD, MODAF Partners, "MODAF Handbook, technical specification for MODAF" Ministry of Defence, 2005.

[90] C. Kobbrin, C Sybbald, "Modelling DODAF Compliant Architectures", *Telelogic white paper* http://www.telelogic.com/standards/modaf.cfm. Last accessed June 2006.

[91] B. Nuscibeh, *Weaving Together Requirements and Architectures*, March 2001 (Vol. 34, No. 3), pp. 115-117.

[92] Oxford University Press, Oxford English Dictionary, 2000.

[93] P. Grünbacher, B. Boehm, "EasyWinWin: a groupware-supported methodology for requirements negotiation", Proceedings of the 8th *European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, Pages: 320 - 321 , 2000 ISBN:1-58113-390-1.

[94] R. Karsten, T. Garvin, "The Use of the Analytic Hierarchy Process in the Selection of Participants for a Telecommuting Pilot Project", SIGCPR/ SIGMIS '96, Denver Colorado USA 1996 ACM.

[95] Airbus, A400M, *New Airlift Requirements*, http://www.airbusmilitary.com/requirements.html Last accessed September 2006.

[96] S. French, T. Bedford, E. Atherton, "Supporting ALARP decision-making by Cost Benefit Analysis and Multi-Attribute Utility", *Journal of Risk Research (2005) Vol 8 No 3* April pp. 207 – 223.

[97] G. Bernat, A. Burns and A. Llamosi, "Weakly Hard Real-Time Systems", *IEEE Transactions on Computers vol.50 no. 4 pp 308-321.* April 2001.

[98] E. De Bono, *Six Thinking Hats,* Penguin Books, 1999.

[99] G. Booch, Software Architecture www.booch.com/architecture/blog/artifacts/SoftwareArchitecture.pdf Last accessed July 2006.

[100] J. Fenn, R Hawkins, T Kelly, P Williams, "Safety Case Composition Using Contracts -Refinements based on Feedback from an Industrial Case Study", in Proceedings of 15th Safety Critical Systems Symposium(SSS'07), February 2007 (Proceedings published by Springer)

[101] Susan Stepney, Fiona Polack, Heather Turner, *Engineering Emergence,* ICECCS 2006, IEEE 2006.

[102] J. Dehlinger, R. Lutz, "Bi-Directional Safety Analysis for Product-Line, Multi-Agent Systems", ITCES'06, April 4, 2006, San Jose, California, USA. ACM 1-58113-000-0/00/0004.

[103] J. P. Corriveau, D. Arnold, S. Basharoust, V. Radonjic, "Automated Support for Validaiton and Verification of .NET Systems", in proceedings of *the International Conference on Software Engineering*, 2007.

[104] US Department of Homeland Security, Secure Software Assurance "A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software", editor S Redwine.

[105] Y. Papadopoulos, J.A. McDermid , "HiP -HOPS: Hierarchically Performed Hazard Origin and Propagation Studies," *SAFECOMP '99, 18th Int'l Conf. on Computer Safety, Reliability and Security, Toulouse*, LNCS, 1698:139-152, Sep 1999.

[106] R. A. Weaver. "The Safety of Software - Constructing and Assuring Argument", PhD thesis, Department of Computer Science, university of York, YCST-2004-01.

[107] US Department of Defence. System Safety Program Requirements, 1993, AMSC Number F6861.

[108] DoD. Architecture Framework Working Group "DODAF version 1 – Deskbook", Department of Defence 2004.

[109] Ministry of Defence, The Acquisition Handbook Edition 6, October 2005.

[110] M. Hall-May, T Kelly, "Using Agent-based Modelling Approaches to Support the Development of Safety Policy for Systems of Systems", in Proceedings of the 25th International Conference on Computer Safety, Reliability and Security (SAFECOMP '06) Gdansk, Poland, September 2006 (Springer-Verlag in LNCS).