KMITL Sci. J. Vol. 5 No. 1 Feb. 2005

Meta-modelling Approach to Traceability and Consistency for Complex Safety Critical Systems Engineering

Paul Mason
Computing Science Program, Shinawatra University, Pathumthani, Thailand

paul@shinawtra.ac.th

Keywords : Traceability, safety-critical systems, CASE tools

Abstract

Traceability is the common term for mechanisms to record and navigate relationships between development and assessment artifacts. Effective management of these relationships is essential to the success of projects involving complex safety critical systems.

Practitioners on such projects typically use a range of techniques to model and analyse the safety and reliability of the systems they are developing. Most have tool support, although poor integration leads to inconsistencies and limits traceability between their respective data sets. This paper proposes a framework that enables links to be established and consistency maintained across data from potentially disjoint safety analysis tools.

1. Introduction

The term safety critical applies to any system whose failure can lead to loss of life, injury or environmental damage [1]. Such systems are designed to satisfy a range of functional and non-functional requirements, including safety, reliability and availability. An important role for developers is therefore in depth analysis that such requirements have been met. In doing so a range of safety analysis techniques may be applied.

Two complementary strategies underpin safety analysis techniques; these have been termed deductive and inductive approaches [2]. Deductive techniques such as Fault Tree Analysis (FTA) start from a system failure and then reason about system or component states contributing to that failure. Conversely, inductive techniques such as Failure Modes and Effects Analysis (FMEA) consider a particular fault in a system component and then attempt to ascertain its consequences. FMEA is seen as a complimentary approach to Fault Tree Analysis such that a fault tree used to determine the causes of a particular hazard may utilise failure rates from relevant FMEAs.

Other commonly used techniques include HAZOPS (Hazard and Operability Studies), used in determining potential causes of failures (those that the afore-mentioned safety requirements are intended to mitigate against) and Event Tree Analysis (ETA) used in determining the accidents that may follow from such hazards.

Most techniques have CASE tool (Computer-Aided Systems Engineering) support, however a lack of well-defined approaches to integration leads to inconsistencies (if individual analyses are inconsistent, it follows the overall assessment will be inconsistent and hence untrustworthy) and limits traceability between their respective data sets.

Traceability is the common term for mechanisms to record and navigate relationships between artifacts produced by development and safety assessment processes. Effective management of these relationships is crucial to the success of projects involving complex safety critical systems.

This paper presents details of MATra, the Meta-modelling Approach to Traceability,

which addresses the two problems described above, namely establishing (and maintaining) *traceability* and *consistency* across data from potentially disjoint safety engineering tools. The framework is realised by exporting information from the internal models of tools, to an integrated environment consisting of: i) a Workspace comprising a set of structures (meta-models) expressed in a common modelling language representing selected safety techniques; ii) well-formedness constraints over these structures capturing properties of the notations; and iii) associations between the structures. To maintain consistency, the structures are verified against a physical system model.

With these principles in mind, the rest of the paper is organised as follows. Section 2 provides an overview of MATra, which is demonstrated by example in Section 3. Section 4 has some concluding remarks.

2. The MATra Traceability Framework

MATra is an object-based framework for tracing between artifacts distributed across disjoint tools supporting safety analysis of critical systems. The framework is based on a set of interconnected 'traceability structures' represented in UML [3], with constraints expressed in the Object-Constraint Language [4]. The example in Section 3 shows one possible means of realising the framework, with alternative representations and tool support considered in Section 4.

MATra is based on five key principles:-

1. A Workspace of *notation dependent structures* (meta-models) representing data exported from tools supporting various safety analysis techniques (see 2.1).
2. A *Meta-class model* to maintain consistency of definition across Workspace meta-models by providing a common underlying representation (described in detail in [5]).
3. A *tool2matra* (tool-to-MATra) mapping function providing data exportation from tools to the Workspace (see 2.2).
4. The *Product Data Synthesis* (PDS), a physical system model that defines the structure,

operational principles, properties of components and relationships between components of the target system; it maintains Workspace consistency by preventing 'bad' data from CASE tools entering via tool2matra (i.e., being mapped to a notation dependent structure) unless the PDS contains corresponding data elements, and by preventing violations once the data is 'inside' (see 2.3).

5. A *Framework Model* that manages common behaviour among MATra elements (described in detail in [5, 6]). Of interest in this paper are classes from the model for creating relations among notation dependent structures (see 2.4), and between these and the PDS (see 2.5).

2.1 Notation Dependent Structures

As previously stated, projects involving complex safety critical systems typically employ a range of techniques – mostly diagrammatic or tabular in their presentation of results - to analyse the systems under development. This paper concentrates on representation of two such techniques – FTA and FMEA – introduced in Section 1. Readers are referred to [5, 6] for comprehensive coverage of others.

When developing a meta-model, the concepts to be modelled must first be established; these are assumed to correspond to concepts underlying safety analysis tools. Having identified the relevant concepts, a meta-model is created using the Class Diagram view of UML. OCL constraints defining well-formedness and PDS consistency are then added. PDS checks verify integrity of the *tool2matra* export function by stating appropriate invariants for Workspace elements that must hold following its invocation. However, their main purpose is to preserve consistency (between the Product Data Synthesis and the Workspace) following changes to the PDS. Meta-model development is demonstrated in Section 3.

2.2 'tool2matra' Mapping Function

A lack of well-defined approaches to integration means practitioners often find it difficult to trace between data created and stored across disparate tools (figure 1).

One solution is to export data from the tools to a Workspace of notation dependent structures (introduced in 2.1) capable of receiving this data. By expressing these structures in a uniform format (i.e., in a common language), links (expressed in the same language as the structures) can be inserted that capture dependencies among data in safety analysis tools, within the Workspace.

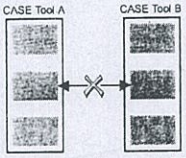

Figure 1. Inter-Tool Traceability Problem.

Ideally, mappings across the CASE tool/MATra interface (i.e., information capture) should involve limited human intervention. Here it is treated as a 'black-box', with all transfers considered in terms of an undefined function, *tool2matra*, that maps data from the internal models of tools onto notation dependent structures (figure 2). Feasibility of this approach is evident in work by project SEDRES [7] and the DOORS requirements management tool [8].

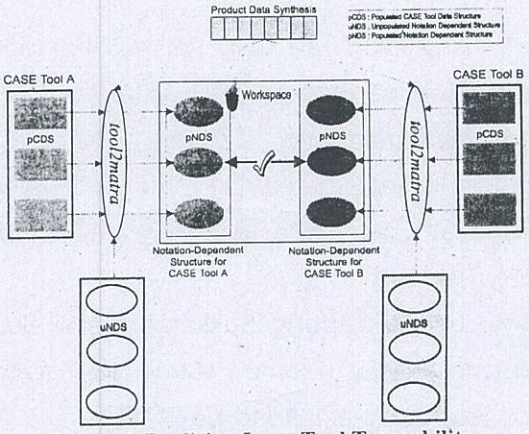

Figure 2. Realising Inter-Tool Traceability.

The *tool2matra* function takes as its input parameters, a populated CASE tool data structure (pCDS) with a corresponding un-populated notation dependent structure (uNDS) and the Product Data Synthesis (PDS), and returns a populated notation dependent structure

(pNDS). The *tool2matra* interface is defined as follows:-

pNDS *tool2matra* (pCDS, uNDS, PDS)

The resulting Workspace provides an integrated environment allowing traceability linkages to be established between otherwise disjoint data (see figure 2); by dropping from CASE tool level into the Workspace, engineers can move around the complete data set wherever links exist.

2.3. Product Data Synthesis (PDS)

As Sub-section 2.2 demonstrated, the Product Data Synthesis plays an important role in MATra by preventing bad data from entering the Workspace via *tool2matra*. However, it also maintains consistency once data is inside the Workspace, notably following updates to the PDS itself. This is ensured by rules over links associating elements of the PDS with corresponding Workspace elements.

Product Data Synthesis is a notation independent structure populated by engineers with design authority. It captures the physical system model in terms of structure, behavior, operational principles, properties of components and relationships between components.

Core PDS constituents are referred to as build elements (BuildElement) and build associations (BuildAssociation). Each build association subtype has single source and target build elements. Correct combinations of element and association subtypes is maintained using appropriate constraints.

BuildElement is specialised by Module, Function, Transaction, Interface and other types. As such, Module describes a system or 'component' of a system - be it hardware, software or human; the functional architecture of a module is described using the Function element, while Transaction is a combination of functions that perform some task. Modules or functions may be connected via an Interface (see [5, 6] for further details).

A subset of Build associations permitted between build elements are shown in Table 1. Most are self explanatory and partly based on existing literature [notably 9 and 10].

BuildAssociation	Source	Target
Encapsulates	Module	Function
HasSubmodule	Module	Module
HasInterface	Module Function	Interface Interface
UsesFunction	Transaction	Function

Table 1 – (Subset) of PDS Build Associations

2.4. Systems Engineering Associations

A mechanism is required for establishing associations between Notation Dependent Structures, and between elements of these structures. In MATra these associations are modelled as classes (figure 3).

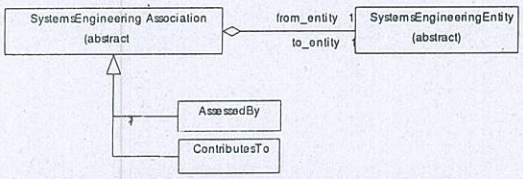

Figure 3. Systems Engineering Association.

SystemsEngineeringAssociation or SEA is an abstract class whose subtypes, e.g., AssessedBy and ContributesTo in figure 3 (which feature in the example in Section 3), realise traceability between two SystemEngineeringEntity subtypes (SEE) over rolenames from_entity and to_entity. SEE is simply a super-class subsuming all notation dependent structures, and elements of these structures. As with build associations, correct usage of SEA subtypes is maintained by constraints (not shown

2.5. Framework Link Entities

Framework Link Entities exist between the PDS and Workspace and take two forms, BEmodelSEE and BEelementSEE (figure 4). These represent links between build elements and meta-models, and between build elements and meta-model elements respectively.

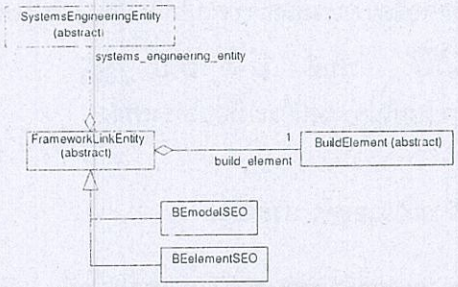

Figure 4. Framework Link Entity.

For example, in figure 5(A), BuildElement '(BE)X' is linked via a BEmodelSEE association to a Workspace meta-model for which it is the subject.

In figure 5(B), BuildElement '(BE)Y' is linked over a similar association to another Workspace meta-model in which (BE)X is named as a model element - namely '(ME)X'; (BE)X and (ME)X are therefore related via a BEelementSEE association. It can be seen also that in the PDS, build element (BE)Y is linked to (BE)X over an (unspecified) BuildAssociation, e.g. HasSubmodule.

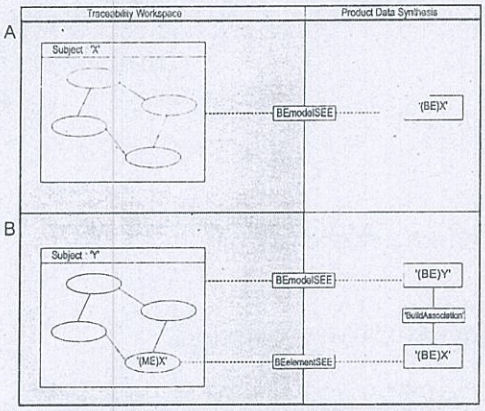

Figure 5. Framework Link Entity Concept.

3. Meta-models for Safety Analysis

This section illustrates development of a Notation Dependent Structure for the Fault Tree Analysis technique alluded to in Section 1.

As previously indicated, Fault Tree Analysis is a deductive technique in that it starts from one particular undesirable event - termed the *top event* - and provides an approach to

investigating potential causes. Analysts then examine the system (or as in this case, a model of the system) to determine ways in which a top event may occur.

The fault tree itself provides a graphical representation of event combinations that can lead to an undesirable event. A number of different event types commonly occur in fault trees, including *intermediate*, *basic*, *undeveloped*, *external* and *conditional* [2]. These are represented by rectangle, circle, diamond, house and ellipse symbols respectively. Events are connected together by logical operators known as *gates* that either enable or prevent the flow of a fault up a tree. The most common forms of gate are the *AND-gate* and *OR-gate* whose symbols are as used in traditional logic- circuits [2].

Once the fault tree logic is complete, it can be used to compute event probabilities. In order to do so, the tree must first be reduced to what is termed *minimal-cut-set-form*. That is, the smallest set of events capable of causing the top event.

Basic events are then annotated with their probability of occurrence. Calculation of intermediate probabilities then progresses up through the tree until probability of the top event can be calculated. Probabilities for output events of the two most common event connectives - and gates and or gates - are determined by the product and sum of their respective input events. Where applicable, it is also common to state failure rate and exposure time of basic events; event probabilities are then calculated from the product of these two values.

Fault trees in MATra support all of the standard concepts set out above. In addition, MATra fault trees reflect actual usage – especially within the nuclear, rail and aerospace domains where they are often used in conjunction with a 'standard' safety assessment process [e.g. 11]. Within such processes, fault trees are typically used to determine *budget probabilities* - i.e. safety objectives (established prior to design) as part of the Functional Hazard and Preliminary System Safety Assessments. Updated versions of these trees determining *actual probabilities* are subsequently produced by the System Safety Assessment to affirm whether the original safety objectives have been met; both the preliminary and updated trees form submissions to the appropriate regulatory body as evidence towards

certification.

Event labels in MATra fault trees differentiate between 'Simple', 'Composed' and 'Synchronisation' categories as derived from a taxonomy proposed in [12]. Simple events are described in terms of a subject entity and a condition; e.g., Valve • Stuck Open. Composed events meanwhile refer to the coincidence of two or more simple events; this is often expressed as two events joined using the "when" conjunction. For example, Valve • Stuck Closed **when** Pump • Stuck On. Finally, Synchronisation describes the temporal relationship between two simple events E1, E2 (normally separated by the preposition "upon") such that in boolean logic, E1 already holds when E2 becomes True; e.g. Tank • Full **upon** Engine • StartUp

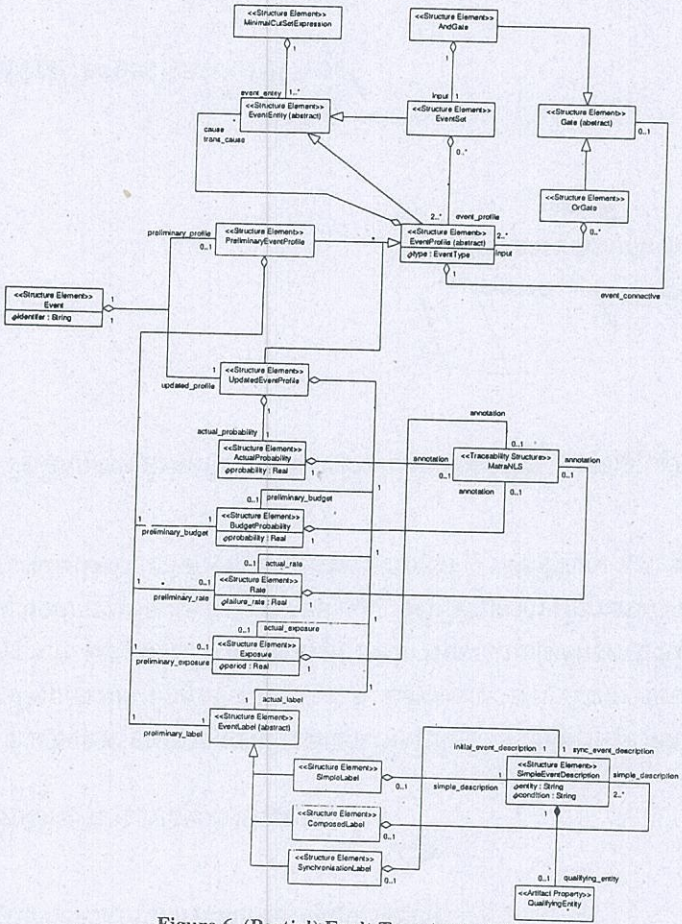

Figure 6. (Partial) Fault Tree Analysis Structure

The (partial) MATra Fault Tree meta-model is shown in figure 6 (and described further in [5]). The figure identifies key fault tree concepts including events, And/Or gate connectives, Minimal Cut Set Expression and event properties such as Exposure and Rate. Note inclusion of the MATra Natural Language Structure (MatraNLS) [13], a utility structure (for fine grained traceability of textual statements that also features in the FMEA meta-model in Section 3.2) enabling annotation of probabilities, exposure times rates, etc.

It can also be seen that MATra fault trees employ the notion of preliminary and updated event profiles in accordance with the domain usage scenarios alluded to earlier. As such, each event is made up of an UpdatedEventProfile and optional PreliminaryEventProfile - specialisations of the (abstract) EventProfile supertype, with attribute (event) type - restricted to int(ermediate), top, bas(ic), ext(ernal) or und(eveloped).

Both preliminary and updated event profiles are described by an EventLabel, or more specifically one of its subtypes, according to the afore-mentioned classification in [12]: namely SimpleLabel, ComposedLabel or SynchronisationLabel. Each label is described either by a single SimpleEventDescription instance for Simple events, two or more instances for Composed events, or two instances (differentiated using the initial_event_description and sync_event_description rolenames) for Synchronisation events. A SimpleEventDescription can be given scope by nominating an optional QualifyingEntity.

Event labels and Exposure objects created for a preliminary tree can normally be reused by the updated version, although the capability exists for new instances to be introduced if necessary; this facility is also available to Rate which is *expected* to change.

Finally, in terms of gates, OrGate takes as input two or more EventProfile elements, while AndGate takes a single EventSet (itself an aggregation of two or more event profiles). Both EventProfile and EventSet are further defined as subtypes of the abstract EventEntity class. This exists partly to allow propagation of the cause and trans_cause associations from EventProfile to both EventProfile itself and to EventSet. The cause and trans_cause associations are used in identifying paths through a fault tree and hence in the expression of constraints and deductive

rules that are dependent on this ability. One such rule populates the MinimalCutSetExpression.

A number of categories of constraints are expressed over the Fault Tree meta-model. These include the following:

- **Well-formedness of fault tree elements**
  For example to ensure non-primary events types have a child gate, while primary event types do not.

```
EventProfile invariant
self.allInstances->forall(e : not( ((e.type = "bas" or e.type = "ext" or e.type = "und") and
e.event_connective->size > 0) or ((e.type = "top" or e.type = "int") and e.event_connective->size = 0)))
```

and to ensure probabilities of output events for Or-gates equate to the sum of input probabilities.

```
EventProfile invariant
self.allInstances->reject(self.allInstances.type = "bas" or self.allInstances.type = "ext" or self.allInstances.type = "und")->forall(e |
self.event_connective->exists(g |
self.preliminary_budget->union(self.actual_probability)->not exists(p |
(e.event_connective->includes(g) and g.oclType = OrGate and
((e.oclType = PreliminaryEventProfile and e.preliminary_budget->includes(p) and p.probability <>
g.input.preliminary_budget.probability->sum) or
(e.oclType = UpdatedEventProfile and e.actual_probability->includes(p) and p.probability <> g.input.actual_probability.probability-
>sum))) )))
```

- **Fault tree 'safety-criteria'**
  For example, a restriction identifying *single failure* causes of a top event:

```
MinimalCutSetExpression
self.allInstances->forall(m | self.event_entity->forall(e | not (m.event_entity->includes(e) and e.oclIsKindOf(EventProfile))))
```

and a restriction identifying *common cause failures* through And-gates.

```
AndGate
self.allInstances->forall(g1, g2 | not((g1.input.event_profile->
intersection(g2.input.event_profile)->not Empty) and g1 <> g2))
```

Note event labels are verified against PDS system model elements using appropriate invariants. We return to this issue in Sub-section 3.3 when addressing consistency among safety analysis techniques..

The partial O-Telos code in figure 7 (embedded in the ConceptBase Object Management System [14]) implements the FTA meta-model:-

```
EventSet in StructureElement, SimpleClass
isA EventEntity with has_part
event_profile : EventProfile end

UpdatedEventProfile in StructureElement, SimpleClass isA EventProfile with
```

```
has_part
    actual_probability : ActualProbability;
    preliminary_budget : BudgetProbability;
    actual_rate : Rate;
    actual_exposure : Exposure;
    actual_label : EventLabel end

ActualProbability in StructureElement, SimpleClass with has_property
    probability : Real
has_structure
    annotation : MatraNLS end

BudgetProbability in …
    as per ActualProbability end

Rate in StructureElement, SimpleClass with
has_property failure_rate : Real …… end

Exposure in StructureElement, SimpleClass with has_property period : Real ……
end

SimpleLabel in StructureElement, SimpleClass isA EventLabel with has_part
    simple_description : SimpleEventDescription end

SimpleEventDescription in StructureElement, SimpleClass with has_property
    entity : String;
    condition : String;
    qualifying_entity : QualifyingEntity end
QualifyingEntity in ArtifactProperty, SimpleClass isA String end

AndGate in StructureElement, SimpleClass isA Gate with has_part
input : EventSet end

OrGate in StructureElement, SimpleClass isA Gate with has_part
    input : EventProfile end

Event in StructureElement, SimpleClass with
has_property
    identifer : String
has_part
    preliminary_profile : PreliminaryEventProfile;
    updated_profile : UpdatedEventProfile end
```

Figure 7. FTA Base Classes (Partial): O- Telos

3.1. Fault Tree Analysis Example

The following example instantiates the Fault-Tree Meta-model using a partial fault tree analysis fragment (figure 8) which considers underlying causes of 'inadvertent braking', a hazard arising from failures of an aircraft braking system control unit (BSCU)[11].

It can be seen from figure 8 that inadvertent braking is caused by a detectable failure in either of the BSCU computers (BSCU1 or BSCU2); note to avoid repetition we only show the branch relating to BSCU1 failures (BSCU1DETD). In summary, this event can occur if the power supply monitor is stuck valid (BS1PSMOFV) and the power supply failure causes bad data (BSCU1PSF). A (partial) O-Telos representation of this diagram follows:-

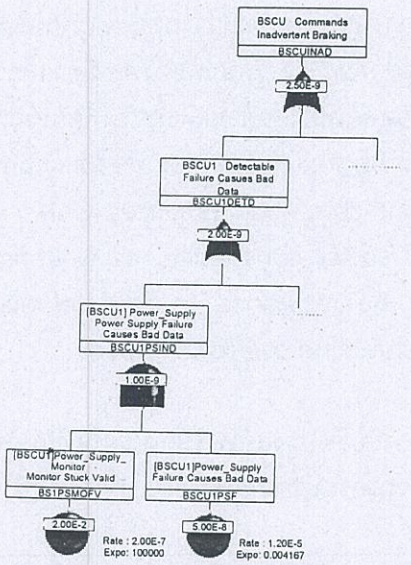

Figure 8. 'Example Fault Tree Analysis'

Note that the O-Telos objects (figure 9) are created **automatically** by *tool2matra*, and 'concealed' from users by an appropriate (graphical) fault tree interface.

```
Definition of Event 'BSCUINADD'
EventA in Event, Token with
Identifier _Identifier : "BSCUINADD"
preliminary_profile preliminaryProfile : EventAPreliminaryProfile end

EventAPreliminaryProfile in PreliminaryEventProfile, Token with
Type Type : "top"
event_connective eventConnective : OrGate1P
preliminary_budget preliminaryBudget : EventAPreliminaryBudget
preliminary_label
preliminaryLabel : EventAPreliminaryLabel end

EventAPreliminaryBudget in BudgetProbability, Token with probability
    _Probability : 2.50E-09 end

EventAPreliminaryLabel in SimpleLabel, Token with simple_description
simpleDescription : EventASimpleEventDescription end

EventASimpleEventDescription in SimpleEventDescription, Token with
Entity _Entity : "BSCU"
Condition _Condition : "Commands Inadvertent Braking" end

Gate definitions

OrGate1P in OrGate, Token with
Input _Input1 : EventDPreliminaryProfile .....
end

Definition of Event 'BSCU1DETD'

EventD in Event, Token with Identifier _Identifier : "BSCU1DETD"
preliminary_profile preliminaryProfile : EventDPreliminaryProfile end
```

```
.....
Definition of Event 'BSCU1PSIND'

EventF in Event, Token with identifier
    _Identifier : "BSCU1PSIND"
preliminary_profile preliminaryProfile : EventFPreliminaryProfile end

EventFPreliminaryProfile in PreliminaryEventProfile, Token with type _Type :
"int" event_connective EventConnective : AndGate1P ..... end
.....
Gate definitions

AndGate1P in AndGate, Token with input
    _Input : HIEventSetP end

HIEventSetP in EventSet, Token with event_profile
eventProfile1 : EventHPreliminaryProfile;
eventProfile2 : EventIPreliminaryProfile end

Definition of Event 'BS1PSMOFV'

EventH in Event, Token with identifier
_Identifier : "BS1PSMOFV" preliminary_profile
preliminaryProfile : EventHPreliminaryProfile
end

EventHPreliminaryProfile in PreliminaryEventProfile, Token with type
    _Type : "bas"
preliminary_budget preliminaryBudget : EventHPreliminaryBudget
preliminary_rate
    preliminaryRate : EventHPreliminaryRate
preliminary_exposure
    preliminaryExposure : EventHPreliminaryExposure
preliminary_label
preliminaryLabel : EventHPreliminaryLabel end

EventHPreliminaryBudget in BudgetProbability, Token with probability
_Probability : 2.00E-02 end

EventHPreliminaryRate in Rate, Token with
failure_rate
    failureRate : 2.00E-07 end

EventHPreliminaryExposure in Exposure, Token with period _Period : 100000 end

EventHPreliminaryLabel in SimpleLabel, Token with simple_description
simpleDescription : EventHSimpleEventDescription end

EventHSimpleEventDescription in SimpleEventDescription, Token with Entity
_Entity : "Power_Supply_Monitor"
qualifiying_entity qualifyingEntity : "BSCU1"
condition _Condition : "Monitor Stuck Valid" end

Definition of Event 'BSCU1PSF'

EventI in Event, Token with identifier
_Identifier : "BSCU1PSF" preliminary_profile
preliminaryProfile : EventIPreliminaryProfile
end .....
```

Figure 9. O- Telos Code for 'Example Fault Tree Analysis'

The above example illustrates our approach to representing (in this case graphical) techniques used in safety engineering. Given a common representation format such as this,

associations can be established to support traceability between techniques (one of two information management problems faced by safety practitioners discussed in Section 1), as the following sub-section demonstrates.

3.2. *Problem 1*: Tracing Between Notations

Consider a Workspace comprising meta-models developed using the above approach and so capable of receiving data from CASE tools via *tool2matra*; traceability associations (Section 2.4) can then be added between models, and elements of models.

Figure 10 gives a user level view of a traceability path depicting associations between a basic event from the fault tree in Section 3.1 (Monitor Stuck Valid) and an entry for a failure that '*contributes-to*' this event in a (Piece-Part[1]) FMEA table. Also shown is the source of the path, namely a comparator element from a Circuit Diagram which is '*assessed-by*' this particular FMEA-entry (along with others not shown).

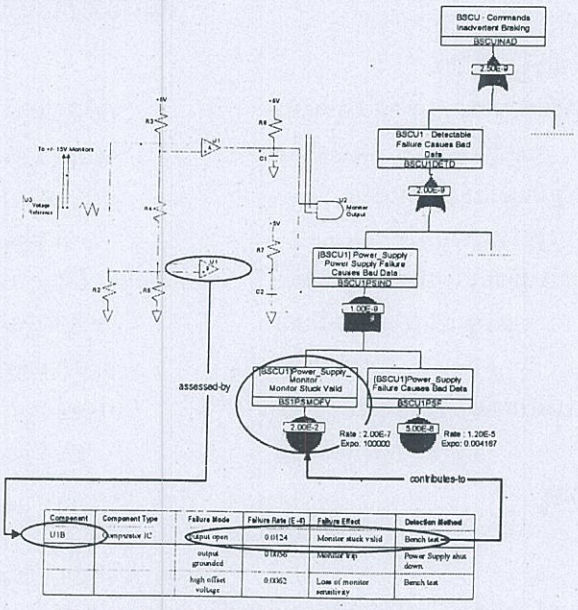

Figure 10. Example Traceability Associations

[1] Is common to distinguish two types of FMEA, *functional* (where investigation is of the lowest level assemblies in a system - typically functional or block-diagram level) and *piece-part* (where investigation is at the level of individual components within an assembly). Piece-part analyses are normally performed to refine failure rates produced from functional FMEA that do not allow a system or component to meet FTA budget failure probabilities.

Figure 11a and 11b shows part of the underlying FMEA meta-model. The former presents a high level view included here to facilitate understanding of navigational paths when discussing PDS consistency in Sub-section 3.3. Again, the complete FMEA structure appears in [5].

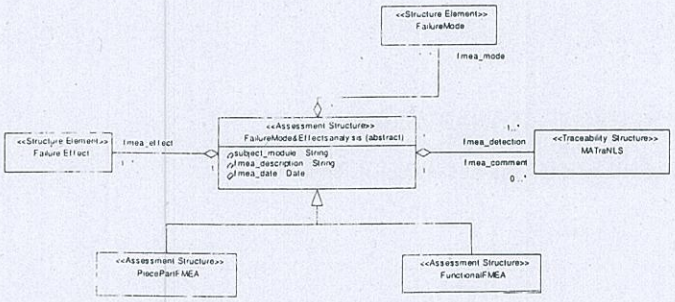

Figure 11a. FMEA Structure (Top Level View)

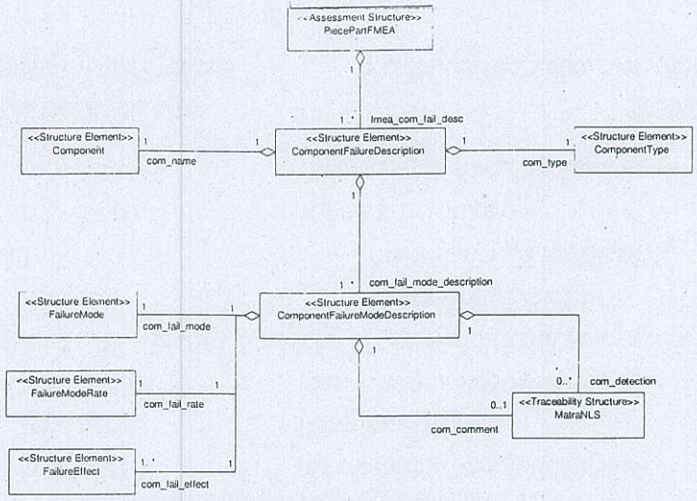

Figure 11b. (Partial) Piece-Part FMEA Structure.

Note, a full discussion on the MATra Circuit Diagram meta-model appears in [5 and 15]. However, the principles used in constructing this model are the same as those used in developing the FTA and FMEA safety analysis meta-models.

To realise the associations in figure 10, an implementation of the ContributesTo and AssessedBy classes (from figure 3, Sub-section 2.4) is instantiated. The source of AssessedBy

(figure 12) is a U1B Comparator element from an instance of a Circuit Diagram Meta-model, and its target, U1BFailModeDescription, an instance of ComponentFailureMode-Description (Figure 11b).

```
AssessedBy01 in AssessedBy, Token with
 from_entity fromEntity : U1B
 to_entity toEntity : U1BFailModeDescription end
```

Figure 12. Instance of Assessed-By Association.

Similalrly U1BFailModeDescription provides the source of our ContributesTo association (figure 13), while the target is EventH from the Fault Tree Analysis in 3.1.

```
ContributesTo01 in ContributesTo, Token with
 from_entity fromEntity : U1BFailModeDescription
 to_entity toEntity : EventH end
```

Figure 13. Instance of Contributes-To Association.

Thus, it can be seen the fundamental gain from the notation dependent structures (meta-models) is their ability to integrate data from disjoint CASE tools (forming what is termed a 'Workspace') which can be linked to create traceability paths as shown above.

3.3. *Problem 2*: Maintaining Consistency

As indicated in Section 1, maintaining consistency among the various safety analysis techniques is of paramount importance in ensuring the overall assessment is trustworthy. Recall, in MATra, (Workspace) consistency is managed using the PDS system model, with various constraints over each of the meta-models used in conjunction. The following is an example of one such constraint for the FMEA meta-model:

Invariant to ensure components exist in the PDS as sub-modules of the subject_module

```
Component invariant
self.allInstances->forall(c |
self.bEelementSEE.build_element->exists(m |
self.piecePartFMEA.bEmodelSEE.build_element->exists(be |
be.module_name = c.piecePartFMEA.subject_module and
c = m.module_name and be.hasSubmodule.target->includes(m))))
```

A subset of elements from the PDS, plus linkages to appropriate Workspace structures and their elements is shown in figure 14. Specifically, it shows instantiation of an implementation of the Module (Build Element) and HasSubmodule (Build Association) classes (from 2.3), as

well as Framework Link Entity classes for BEelementSEE and BEmodelSEE (from 2.5):-

```
Examples of PDS Element Instances
PowerSupplyMonitor in Module, Token with
  module_name moduleName : "Power Supply Monitor"
end

PowerSupplyMonU1B in HasSubmodule, Token
with from_entity fromEntity :
 PowerSupplyMonitor
 to_entity toEntity : U1B end

Examples of Framework Link Entity Instances
PowerSupplyMonitor_PSMonFMEA in BEmodelSEE
with build_element buildElement :     PowerSupplyMonitor
 system_engineering_entity sysEngEnt :
 PowerSupplyMonitorFMEA end

U1B_fmeaU1B in BEelementSEE
with build_element buildElement : U1B
system_engineering_entity sysEngEnt: U1B_FMEA_Component end
```

Figure 14. Examples of PDS and Framework Link Entity Instances.

Note, O-Telos implementation of the above OCL constraint (not shown) is enforced by navigation over BEelementSEE classes such as `U1B_fmeaU1B` in figure 14.

Figure 15 puts the issues addressed in Sub-sections 3.2 and 3.3 into context – effectively an instantiation of figure 5; on the right are PDS Elements, linked by Framework Link Entities to (icons for) the FTA and FMEA meta-models; the ContributesTo association described previously is also illustrated.

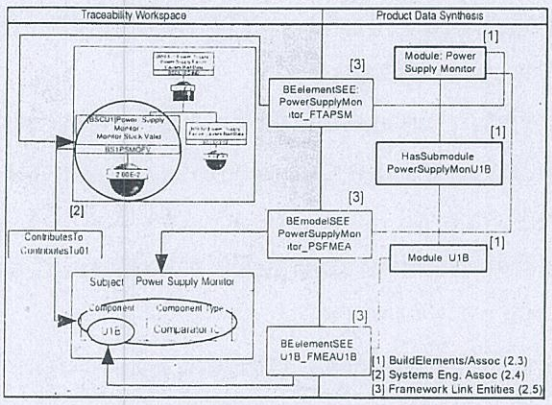

Figure 15. Relating MATra Elements.

Note, associations such as ContributesTo and AssessedBy currently require manual insertion. Yet, as all MATra models are formally defined, they can be potentially derived

using inference rules. This is relatively simple where associations link common elements. For example, it can be inferred from Figure 15 that because PDS element Power Supply Monitor is linked over BEmodelSEE and BEelemenSEE associations to elements of both the FTA and FMEA table, a ContributesTo association exists between the two Workspace elements.

4. Concluding Remarks

This paper addressed the issue of data integration to improve traceability and consistency of safety engineering artifacts. A framework (MATra) was proposed that enables engineers to trace between information residing in different CASE tools. This is achieved by exporting data from the internal models of these tools to a set of structures (meta-models) expressed in a common modelling language, enabling traceability links to be established between the models and between elements of these models; a further structure is used to maintain consistency.

The structures were expressed here in UML/OCL, and in turn implemented in O-Telos using ConceptBase. However this is largely incidental as they may just as well have been represented in for example, EXPRESS [16] to align our work with SEDRES [7], or in XML [17]. The use of commercial traceability tools as a target implementation platform was also investigated. In particular DOORS [8], which now employs a MATra-like approach by transferring data from CASE tools onto information models expressed in a common language (termed 'surrogate' modules) which can then be linked to support traceability.

The novelty of MATra lies in two areas:-

1. A framework for traceability and consistency across artifacts for safety critical systems engineering projects,
2. Meta-models for a representative set of safety assessment techniques, as well as development and project management notations and techniques used in safety critical systems engineering [5, 6, 17].

MATra has been applied to a range of case studies from the aerospace and rail industries, including commercial specifications for Fuel Management, Flight Control, and Wheel Braking systems [5].

The results of these studies are promising, while highlighting a number of areas for future research. These include the following:-

- Extension of Workspace notations
- Application of MATra to other safety-critical domains (e.g., nuclear, offshore oil and gas and chemical)
- Enriching the Product Data Synthesis
- Systems Engineering process issues for MATra
- Investigation of an inverse mapping function (*matra2tool*)
- Use of *tool2matra* to optimise Workspace revisions
- Incorporation of standards knowledge (e.g., [11]) into MATra

Finally, we highlight a brace of related works including [1] and [18] which provide tool support for managing traceability and consistency of reliability, safety and other concurrent engineering information. These are closest in spirit to MATra, however both rely on their own custom editors for FTA, FMEA, system modeling, etc. While this provides one solution to the problems of traceability and consistency, it is a rather binding approach given that experienced practitioners are unlikely to willingly relinquish existing tools for bespoke alternatives.

5. Acknowledgements

The financial support of the BAE SYSTEMS Dependable Computing Systems Centre (DCSC) is gratefully acknowledged.

Thanks are also due to Dr Amer Saeed (Advantage Technical Consulting), Dr Nick Rossiter (University of Northumbria) and Dr Julian Johnson (BAE SYSTEMS) for their guidance and comments on material presented in this paper.

6. References

[1] Wilson, S. P. & McDermid, J. A. - Integrated Analysis of Complex Safety Critical

Systems, *The Computer Journal*, 38(10), 1995

[2] Vesely, W., Goldberg, F., Roberts, N. & Haasl, D. - *Fault Tree Handbook*, Nureg 0492, US Nuclear Regulatory Commission

[3] Pierre-Alain Muller, Instant UML, Wrox Press., 1997.

[4] J. Warmer & A. Kleppe, The Object Constraint Language, Addison-Wesley, 1999

[5] P. Mason, *Meta-modelling Approach to Traceability*, University of Newcastle PhD Thesis, 2002.

[6] P. Mason, A. Saeed, S. Riddle & P. Arkley, Meta-modelling Approach to Traceability, *Procs. Int'l Conf. on the Engineering of Computer-Based Systems*, Alabama, Apr. 2003

[7] J. F. E. Johnson, The SEDRES Project: Producing a Data Exchange Standard Supporting Integrated Systems Engineering, Proc. of 8th INCOSE Symposium, Vancouver, Canada, pp. 367-374.

[8] Telelogic, *DOORS Ref. Manual*: v5.1, 2001

[9] M. Klein, Capturing Design Rationale In Concurrent Engineering Teams, *IEEE Computer*, Jan. 1993,pp 39-47

[10] D. Oliver, A Draft Integration of Information Models: Complement Model and Oliver Model, *Procs. of Tutorial and Workshop on Systems Engineering of Computer-Based Systems*, pp. 44-69, 1994

[11] European Organization for Civil Aviation Electronics - Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, EUROCAE document ARP 4761, Dec.

[12] Górski, J. & Wardziński, A. - Formalising Fault Trees, *Proc. Safety-Critical Systems Symposium*, Brighton, UK, pp. 310-327, 1995

[13] P. Mason, On Structuring Formal, Semi-Formal and Informal Data to Support Traceability in Systems Engineering Environments, *Proc. 13th International Conference on Information & Knowledge Management*, Washington D.C., USA, Nov. 2004

[14] M. Jarke, R. Gallersdorfer, M. Jeusfeld, M. Staudt & S. Eherer, ConceptBase: A Deductive Object Base for Meta Data, *Journal of Intelligent Info. Sys.*, Mar., pp.167-192, 1995

[15] D. Schenk & P. Wilson, *Information Modelling: The EXPRESS Way*, OUP 1994

[16] C. Goldfarb & P. Prescod, *The XML Handbook,* Prentice-Hall, 2000

[17] P. Mason, On Traceability for Concurrent Engineering Environments, *Procs. International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2004)*, Virgin Islands, USA Nov. 2004

[18] R. Collins & J. Dent, A Practical Case Study pf Reliability, Safety and other Concurrent Engineering Information, *Procs. Safety & Reliability Conference*, 1994