Towards Compositional Safety Analysis via Semantic Representation of Component Failure Behaviour

Konstantinos MOKOS [a], Panagiotis KATSAROS [a], Nick BASSILIADES [a], Vangelis VASSILIADIS [b], Maxime PERROTIN [c]
[a] *Department of Informatics, Aristotle University of Thessaloniki, Greece*
e-mails: mokosko@otenet.gr, {katsaros, nbassili}@csd.auth.gr
[b] *Gnomon Informatics S.A., Thessaloniki, Greece – e-mail: v.vassiliadis@gnomon.com.gr*
[c] *European Space Agency, ESTEC/TEC-SWE (Software & Standards), Noordwijk, The Netherlands*
e-mail: Maxime.Perrotin@esa.int

Abstract. In dependable systems engineering safety assessment of complex designs that involve software and hardware components is one of the most difficult tasks required. Due to the different modelling languages and models that are used for complementary tasks, the model and specification artefacts are not easily shared by the experts involved in the design process. Moreover, the structural and semantic differences of the used language representations open a possibility for inconsistencies between the corresponding models. This work explores the role of an ontology representation of component failure behaviour as a basis for automated model transformations, as well as a library of reusable knowledge artefacts to be used in different modelling languages and models. The presented approach was motivated by recent findings and requirements derived from European industrial-driven research and development projects[1].

Keywords. Ontology, Model Driven Software Development, Dependability, Safety Analysis

1. Introduction

In the design of complex critical systems, safety analysis has to be performed in a systematic manner and it is usually refined and updated iteratively as the design process proceeds. Safety assessment traditionally requires the combination of various results derived from various models. Due to the different modelling languages and models that are used for complementary tasks, software engineers research model transformation techniques [10] that enable them to cope with:

- the heterogeneity in textual representation, syntax, semantics, and scope of the modelling languages and
- the possible inconsistencies that are likely when we have concurrent development of models on the basis of different tools.

[1] This work is partly funded by the European Space Agency (ESA) Contract Ref: RFQ3-12080/07/NL/FM/na.

This work explores the role of an ontology representation of component failure behaviour as a basis for automated model transformations, as well as a library of reusable knowledge artefacts to be used in different modelling languages and models. We present the architecture of a client application that will generate *AltaRica specifications* from models described in the *Architecture Analysis & Design Language (AADL)*. AADL is a textual and graphical language introduced by the Society of Automotive Engineers (SAE) for the design and analysis of the software and hardware architecture of safety critical real-time systems. It can be used in conjunction with an *Error Model Annex* standard to add dependability-related information such as fault and repair assumptions and error propagations to an architecture model. AltaRica is a dependability language for formally specifying constraints automata that describe the behaviour of systems when faults occur. Several dependability tools can process AltaRica models, like for example symbolic simulators, model-checkers, fault tree generators, sequence generators and AltaRica graphical modellers (Cecilia OCAS & ARBOR), which are used to support industrial safety analysis processes.

To limit the effort of building dependability models several studies [7, 9] propose the definition of error models that will be applied to a number of system components and will be tailored with component-specific information. This approach assumes the maintenance of libraries with error models and component definitions that will be reused from one project to the other. Most of these projects potentially involve a number of geographically distributed suppliers, subcontractors, developers, certifiers and other stakeholders and this raises the need for an open knowledge base, which will enhance collaboration and information sharing intra and inter organizationally. We invest on an ontology based representation of component failure behaviour that supports the exchange of models among the different modelling tools and the involved stakeholders and opens prospects to meet the following requirements:

- To define a formal semantics in order to capture the meaning of error model definitions in a correct and complete manner.
- To support error model definitions easy to process by tools through an XML-based serialization format (which is not supported in current AADL tools).
- To define a formal semantics that will be deployed as a network resource, in order to validate error model definitions against a central schema (similar to the validation of XML documents against their XML schema).
- To provide a facility for describing the relationships with alternative models and languages in addition to AltaRica.
- To allow queries for potential solutions to design problems on the Web.

In Section 2 we report the findings of related previous work. Section 3 introduces basic concepts of compositional safety analysis and describes the AADL and AltaRica languages with their associated tools. Section 4 outlines the ontology based representation of component failure behaviour. Section 5 provides a simple case study to show the capturing of error models in AADL architecture designs and their transformation to AltaRica specifications. We conclude by summarizing the scope of the ongoing work towards the development of the outlined Semantic Web based safety engineering process.

2. Related work

The ASSERT European Integrated Project (Automated proof-based System and Software Engineering for Real-Time systems) has investigated, elaborated and experimented advanced methods to integrate failure propagation Altarica models to system designs developed in AADL.

During this project, a major concern was raised related to the possibility of inconsistencies between the used failure propagation models, Altarica models and the other models that were developed according to the ASSERT process. In order to avoid these inconsistencies, Onera investigated the feasibility of a tool that generates an Altarica model from a model described in AADL and proposed the development of libraries of Altarica nodes that will be reused from one project to the other. The tool was based on a model transformation approach that extracts from the AADL model, the functional and hardware architecture of the system. The development of libraries worked well only for families of systems that did not differ too much, and generally, it was found difficult to model certain types of failure propagations in physical domain.

Similar to Onera, LAAS investigated, during the first phase of the ASSERT project, an alternative approach where an AltaRica specification is enriched with failure propagations coming from AADL code written following the Error Annex standard. According to the ASSERT reports the Altarica code was found to be much more complex than before, but feasible, as long as component relationships and various kinds of analysis are defined [9].

3. Compositional Safety Analysis

A service delivered by a system is its behaviour as it is perceived by its users and in the safety analysis domain this is well known as the external state. Correct service is delivered when the service implements the intended system function. Related to the abnormal system operations, three basic concepts are used: failure, error, and fault. A system fails when it deviates from the intended functions and behaviours. Service failure means that at least one or more external states of the system deviates from the correct service state. The forms of deviations are characterized in terms of failure modes. An error is the part of total system states that can result in a system failure. Error events are used to model internal state (behaviour of basic component). A failure occurs when an error propagates from internal states to external states, causing the whole system to fail. The causes of an error, either adjudged or hypothesized, are referred to as faults. A fault produces one or multiple errors when it is activated. A fault is active when it causes an error, otherwise it is considered as dormant [1, 2, 4].

Failure condition (FC) refers to a combination of failure modes applied to functions of the system under study. We considered the following generic failure modes of functions: total loss, partial loss and erroneous behaviour. The FC may also include conditions that describe the current mode of the system. A FC may be permanent or transient [9] and it is characterized by the following attributes:

- SEVERITY classification that includes the categories Catastrophic, Hazardous, Major, Minor and No Safety Effect.

- QUANTITATIVE OBJECTIVE, which is a failure rate value that can be stated per mission hour, or for a given mission phase. Typical values are 10^{-9} /hour for Catastrophic FC, 10^{-7}/hour for Hazardous. Other categories are dependent on project, reliability objectives, etc.
- QUALITATIVE OBJECTIVE essentially describes the number N of individual faults which are considered for a given FC: “No combination of events with less than N individual faults shall lead to FC” with N = 2 for Catastrophic FC, N=1 for Hazardous and Major FC.

Safety analysis provides information about the consequences of components/ system failures and is related to all other types of analysis. The aim is to identify hazards, to access the risk, and to support hazard control. The analysis requires error models and models capturing failures and environmental conditions. Classical techniques for safety analysis are the so-called HAZOP (Hazard and Operability Studies), FMEA (Failure Modes and Effects) and FTA (Fault Tree Analysis) [4, 5].

3.1. AADL

The SAE Architecture Analysis & Design Language (AADL) is a textual and graphical language used to design and analyze the software and hardware architecture of safety critical real-time systems [3, 4]. AADL is used to describe functional interfaces to components (such as data inputs and outputs), performance-critical aspects (such as timing), how components interact, how application software components are allocated to execution platform components, as well as the dynamic behaviour of the runtime architecture (by supporting the concept of operational modes and mode transitions).

AADL is designed to be extensible. By allowing users to extend the core language with additional features it is also possible to specify safety levels, criticalities, fault tolerance, and error handling. Extensions take the form of new properties and analysis specific notations that can be associated with components. The user just imports Annex libraries which extend the language and customize an AADL specification to meet project- or domain-specific requirements. AADL is currently provided with two basic annex libraries: the behaviour annex and the error model annex [7].

The Error Model Annex defines additional properties that describe reliability of the system components and a state machine that specifies error states. Thus the Error Model Annex can be used in conjunction with the description capabilities of AADL to add dependability-related information, such as fault and repair assumptions, error propagations and fault-tolerance policies. More precisely, error model types are used for modelling fault/error/failure behaviours. They may declare error states (possibly identified during a hazard analysis), failure modes (possibly identified during a Failure Mode and Effects Analysis), error events (used to model internal faults), as well as internal repairs and error propagations (which are used to model failure effects).

The Error Model Annex supports a compositional approach to error modelling:

- It enables reuse of error models.
- Makes it easier to modify architecture specifications and automatically regenerate safety and reliability models.

- It facilitates abstraction and mixed-fidelity modelling.
- It enables improved traceability between architecture specifications and models and analysis results.

The language features can be used to specify errors that propagate between components and connections, depending on the structure of the architecture model. There also language features used to specify how components vote to detect and mitigate errors in their subcomponents or in the components on which they depend. Provided the availability of appropriate tool support the language semantics can be used to check for consistency, completeness and traceability between the error models of interacting components, as well as between the error models of components and their subcomponents.

AADL is supported by two open source tools: OSATE and TOPCASED. The latter is a Metamodelling Framework developed by Airbus and 20 companies and currently supports AADL Graphics, AADL XML, model transformation and the Behaviour Annex. The former is an Eclipse based tool developed by the Software Engineering Institute (SEI) and currently supports full language editing and semantic checking, multiple analysis plug-ins and integrated text and graphical editing with TOPCASED.

3.2. AltaRica

AltaRica [6] is a dependability language designed in the University of Bordeaux to formally specify constraints automata (formal models which describe the behaviour of systems when faults occur). Several dependability tools can process AltaRica models [4, 5, 6, 9]: symbolic simulators, model-checkers, fault tree generators, sequence generators, AltaRica graphical modelers (Cecilia OCAS & ARBOR), etc. In AltaRica the models have the following structure:

- A model of a system consists of hierarchies of components called nodes. A node gathers flows, states, events, transitions and assertions.
- A component has is own variables (internal or external), plus some flow variables it can only read and are shared with other components. Flows are the visible parameters of the component whereas states are internal variables.
- Assertions are boolean formulae that state the constraints linking flows and internal states. Assertions are system invariants (they must always be true).
- Transitions describe how the initial states may evolve. They are characterized by a guard (a boolean constraint over the component flows and state), an event name and a command part (value assignation of some state variables).
- The system is primarily separated to physical subsystems, which are described as nodes and capture all events and behaviour of the subsystems.

Our goal is the fault model to have as much information as possible about the various events in which the components of the system can malfunction. A failure can be defined as an event which can affect the state of the node. Failure modes are called the different forms of deviation from correct service and can be defined using a transition taken to the particular failure event. An AltaRica model-checker checks the set of properties that eliminate the failure events. Safety requirements are given as failure conditions specified in the form of Linear Temporal Logic formulae that have to be proved. An AltaRica fault-tree generator takes an AltaRica model as input and an unexpected event and subsequently generates a fault-tree describing the failure situations that lead to the unexpected event. Once the fault tree is generated, a fault tree analyzer can be used to compute the set of prime implicants on the non-temporal failure conditions.

4. Ontology based representation of component failure behaviour

The presented modelling languages (AADL and AltaRica) are both specification languages with different textual representations, syntax and scope, thus creating a wide semantic gap. Most current model transformation approaches (e.g. Model-driven software development [10] and OMG's Model-driven Architecture) address the problem of different model representation formats by the use of metamodels, which in fact can express only the logical syntactical structures of the corresponding models and cannot specify their semantics [12]. However, true model integration can only be achieved on a semantical level [8].

Ontologies are considered a key element for semantic interoperability because they act as shared vocabularies that utilize a grammar for using this vocabulary. The grammar specifies (with formal constraints) what it means to be a well-formed specification and how the terms in the ontology's controlled vocabulary can be used together. The semantics of the concepts of interest is specified in a logic-based representation language thus resulting in unambiguous and machine-processable specifications. Moreover, the used representation languages come with executable calculi enabling querying and reasoning at run time.

The proposed safety engineering process (Figure 1) uses a domain ontology built in the Web Ontology Language (OWL) to provide a classification of all possible component failure modes [15] and associate them with constraints that state their meaning. The whole architecture includes two bindings that specify the semantic mapping from the AADL/Error Annex and AltaRica metamodels to the domain ontology. An AADL/Error Annex model is an instantiation of the corresponding metamodel and it is semantically interpreted by the underlying domain ontology. The logical and syntactical structures of the AADL models should obey to a number of metamodel constraints. These constraints are associated with OWL property types and the semantic mappings to the domain ontology are used by reasoners that apply inference rules on the relationships of the ontology concepts. Inference rules will help to detect lacks of model elements and semantically inconsistent parts. Rules will also infer additional assertions, which are not specified explicitly but will be derived through the AADL and failure mode semantics (e.g. derived error models [7]). We use the term model inference to refer to the aforementioned model validation, which is possible only in an ontology-based model transformation process.

Reasoning functionality will be exploited also in semantic web clients other than the AADL to Altarica model transformation. One obvious prospect is the definition of the notion of component failure behaviour matching based on relationships like refinement and simulation [13].

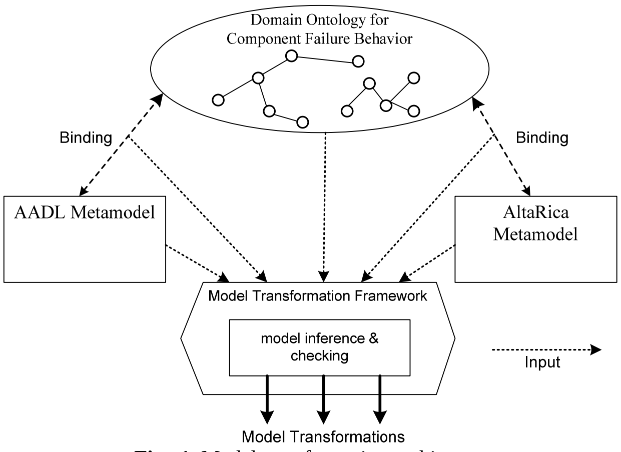
Domain Ontology for Component Failure Behavior
Binding
Binding
AADL Metamodel
AltaRica Metamodel
Model Transformation Framework
model inference & checking
Input
Model Transformations

Fig. 1. Model transformation architecture

In the proposed framework, additional rules related to the syntactic mapping of AADL models to AltaRica specifications will be implemented within the used model transformation tool chain (e.g. openArchitectureWare [11]). Interoperability between the model transformation tool chain and the ontology part of the shown architecture will be achieved through the EMF Ontology Definition Metamodel [14] that essentially provides a set of programming APIs for creating, modifying, and navigating an OWL ontology in which all objects are also treated as Eclipse Modelling Facility (EMF) model objects. This makes possible to utilize the comprehensive development facility of Eclipse that is required by the selected model transformation tools.

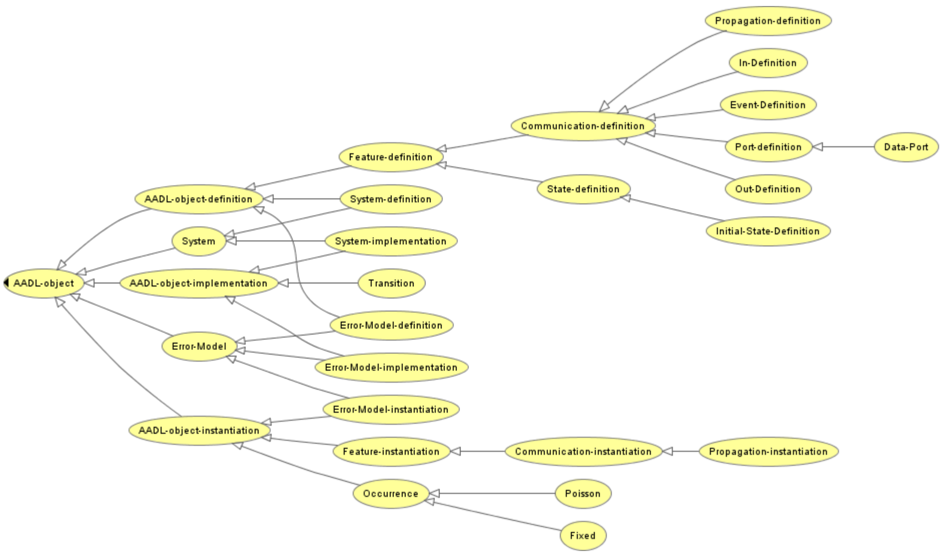

Fig. 2. Class hierarchy of the AADL/Error Annex ontology

Figure 2 shows the class hierarchy of the AADL/Error Annex ontology and Figure 3 presents a snapshot of the instances in the model transformation case study of the next section. The ontology follows a meta-modelling approach, defining classes for artefact definitions, implementations and instantiations. Actually two major artefacts are modelled, namely system and error model. Systems can consist of nested sub-components. System and error model definitions have features, namely in/out ports, error events-propagations and states. Definitions are associated with implementations, which can further refine the models. Error model implementations define transitions between states, based on incoming events. Finally, artefact instantiations associate artefact implementations with a specific pattern of event/data flow occurrence.

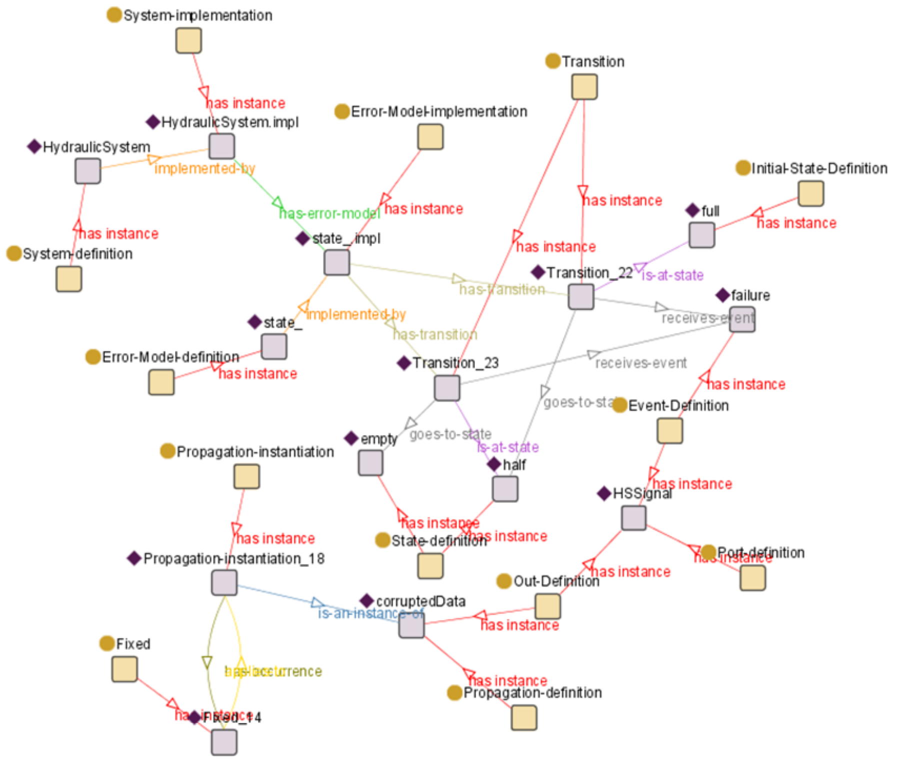

Fig. 3. Ontology representation of an Hydraulic System failure behavior

5. Case study

Our prototype architecture will be tested by simple and more complex case studies. A quite simple system design is the Aircraft Hydraulic Leakage Detection System shown in Figure 4. The purpose of this system is to detect and stop leakages in the two hydraulic systems, which feed the moving parts of an aircraft. Leakages in the hydraulic systems could in the worst case result in such low pressure that the airplane becomes uncontrollable. To avoid this, some of the branching oil pipes are protected by shut-off valves. These valves can be used to isolate a branch in which the leakage has been detected. Then, although the leaking branch will no longer function, the other branches will still keep the pressure and be able to supply the moving parts with power.

The reading of oil level sensors and the controlling of the four shut-off valves are handled by three electronic components, as depicted in Figure 4. The H-ECU (Hydraulic Engine Control unit) is a software component that continually reads the oil reservoir levels of the two hydraulic systems (HS1 and HS2) and determines which shut-off valve to close accordingly. HS1 sensors determine valves B1 and C1, while HS2 sensors determine valves B2 and C2. However, it would be very dangerous if some electrical fault caused more than one valve to close at the same time. For this reason, two programmable logic devices, here called PLD1 and PLD2, continually read the signals to and the statuses of the valves, and if the readings indicate closing of more than one valve, they will disallow further close operations. Thus, PLD1 and PLD2 add fault tolerance to the shut-off subsystem implemented in the H-ECU. PLD2 will only accept a request from the H-ECU for closing a particular valve if the check, which is partially done in PLD1, indicates that everything is fine. A valve will close only when both the low-level signal, which is the shut-off signal directly from the H-ECU, and the high-side signal, which is the checked signal from PLD2 are present.

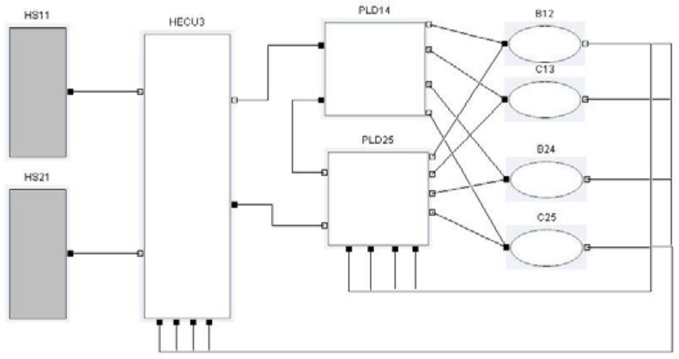

Fig. 4. Hydraulic leakage detection subsystem

The Hydraulic Leakage Detection System was modelled in AADL using the OSATE (Eclipse) graphical interface and the model was subsequently transformed to an AltaRica specification. The specification excerpt of Figure 5 represents the Hydraulic System (HS).

Figure 3 shows the AADL/Error Annex ontology representation of the Hydraulic System component. There are two definition objects, one for the system `HydraulicSystem` and one for the error model `state_`, and two equivalent implementation objects. These are connected to the required feature, transition and state objects. Finally, there is one instantiation object for the occurrence pattern of the corrupted data error propagation.

6. Conclusion

This paper introduces a Semantic Web architecture that supports geographically distributed industry groups in performing compositional safety analyses on components/systems during the requirements definition and design phases. We presented the details of the model transformation architecture under development, but we also stated the scope of our proposal that in fact is versatile:

- To support product/component manufacturers in sharing specifications of component failure behaviours intra and inter-organizationally and in looking for components that match a particular failure behaviour.

- To cope with the heterogeneity in syntax, semantics, and scope of the modelling languages used in compositional safety analyses and the possible inconsistencies, that arise when we have concurrent development of models with different tools.

```
system HydraulicSystem
  features
    HSSignal: out event port;
  annex error {**
         error model state_
         features
         full: initial error state;
         half,empty: error state;
         failure: error event;
         CorruptedData: out error propagation
{Occurence => fixed 0.1};
         end state_;

  **};
end HydraulicSystem;

system implementation HydraulicSystem.impl
  annex error {**
error model implementation state_.impl
transitions
         full- [failure] -> half;
         half- [failure] -> empty;
end state_.impl;
         **};
end HydraulicSystem.impl;
```

```
node HDL_HS1
 flow
  icone : [1,1] : out ;
  HSSignal : HDL_type_HS : out ;
 state
  state_ : {full,half,empty} ;
 event
  failure ;
 init
   state_ := full ;
trans
state_ = full |- failure -> state_ := half;
state_ = half |- failure -> state_ := empty;
assert
(if state_ = full then HSSignal = Full
 else if state_ = half then HSSignal = Half
else HSSignal=Empty);
extern
   law <event failure> = exponential(1e-1) ;
edon
```

Fig. 5. AADL and AltaRica specifications for the HS1 and HS2 component failure behavior

7. References

[1] Leveson, N. G. Safeware: System safety and computers. Addison-Wesley Publishing Company, 1995

[2] Avizienis, J.-C., Laprie, Landwehr, B. Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and secure Computing*, Vol. 1 (1), pp. 11-33, 2004

[3] Society of Automotive Engineers, SAE Architecture Analysis and Design Language, SAE standard AS5506. Available in: http://www.aadl.info.

[4] Joshi, A., Miller, S. P., Whalen, M., Heimdahl, M. P. E. A proposal for model-based safety analysis, 24th Digital Avionics Systems Conference (DASC), 2005

[5] Bieber, P., Castel, C., Seguin, C. Combination of Fault Tree Analysis and Model Checking for safety assessment of complex system, 4th Europ. Dependable Computing Conf., LNCS 2845, pp. 19-31, 2002

[6] Safety assessment with AltaRica – Lessons learnt Based on Two Aircraft System Studies. 18th IFIP World Computer Congress, Toulouse, France, 2004

[7] Feiler, P., Rugina, A. Dependability Modeling with the Architecture Analysis & Design Language (AADL), Software Engineering Institute (SEI), 2007

[8] Dietrich, J., Elgar, C. An Ontology Based Representation of Software Design Patterns, In: Design Pattern Formalization Techniques, Toufik Taibi (Ed.), IGI Publishing, pp. 258-279, 2007

[9] Bieber, P., Blanquart, JP., Durrieu, G., Lesens, D., Lucotte, J., Tardy, F., Turin, M., Seguin, C., Conquet, E. Integration of formal fault analysis in ASSERT: Case studies and lessons learnt, 4th European Congress on Embedded Real Time Software (ERTS 2008), Toulouse, France, Jan 2008

[10] Stahl, T. and Volter, M. Model-driven software development: Technology, Engineering, Management, Wiley, 2005

[11] The Official openArchitectureWare Homepage, http://www.openarchitectureware.org/

[12] Saeki, M. and Kaiya, H. On Relationships among Models, Meta Models and Ontologies, Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), Un. of Jyväskylä, Finland 2006

[13] Pahl, C. An ontology for software component matching, International Journal of Software Tools Technology Transfer, Vol. 9, Springer, pp. 169-178, 2007

[14] EODM - EMF Ontology Definition Metamodel, http://www.eclipse.org/modeling/mdt/?project=eodm

[15] McDermid, J. and Pumfrey, D. A development of Hazard Analysis to aid software design, 9th Annual Conference on Computer Assurance (COMPASS), 1994