A Reusable Modular Toolchain for Automated Dependability Evaluation

Leonardo Montecchi, Paolo Lollini, Andrea Bondavalli
Dipartimento di Matematica e Informatica
University of Firenze
I-50134 Firenze, Italy
{lmontecchi,lollini,bondavalli}@unifi.it

ABSTRACT

Model-transformation techniques have increasingly gained attention in the design and evaluation of high-integrity systems, with the purpose to provide (semi-)automatic tools for non-functional analysis. Analysis models are automatically derived from an architectural description of the system in a UML-like language. One of the main challenges is designing tools which can be reused: the modeling language, the analysis tools, and possibly the analysis method itself are going to evolve over time (e.g., due to different domains, new software versions, updates to standards). In this paper we describe the design and implementation of the toolchain for state-based dependability analysis developed within the CHESS project. The toolchain, which also provides back-annotation facilities, has been designed to be adapted to different modeling languages and analysis tools. The tool has been implemented as a plugin for the Eclipse platform, and it is publicly available on the CHESS website.

Keywords

CHESS, non-functional analysis, reusability, toolchain, dependability

1. INTRODUCTION

Model-Driven Engineering (MDE) techniques [25] have gained popularity in the design of high-integrity systems, and provide a useful means to automatize the analysis of non-functional system properties. Approaches and tools for automated derivation of dependability [5] models exist in literature, but solutions are often bound to specific analysis techniques, languages, domains, or tools. As the MDE world is constantly evolving, with new models, languages, and tools being constantly introduced or updated, a great effort is being spent to develop more general approaches.

In this paper we describe the design and implementation of the "CHESS Plugin for State-Based Analysis" (CHESS-SBA), a toolchain for automated state-based dependability analysis developed within the CHESS project [2]. The toolchain has been designed to be flexible, and be easily adapted to different modeling languages and analysis tools. It is implemented as a plugin for the Eclipse platform, and it is publicly available on the CHESS website [2], as part CHESS framework.

The paper is organized as follows. Related work is discussed in Section 2, while Section 3 introduce the context, purpose, and requirements of the toolchain that we present in this paper. Section 4 describes the abstract, reusability-oriented, architecture on which the toolchain is based, and how it has been concretely implemented within the CHESS project. The reusability properties of the toolchain are discussed in Section 5, while conclusions are drawn in Section 6.

2. RELATED WORK

Several works in the literature adopt MDE approaches to perform dependability analysis, using different techniques, including Bayesian rules [11], Fault Trees [12], Stochastic Petri Nets [1, 10, 20]. Tools implementing transformation approaches have been also developed. To cite a few, OpenS-ESAME [26] generates SPN models from diagrams expressing dependencies between components; ADAPT [24] implements a model transformation from AADL models to GSPN models; the work in [17] describes a tool for availability and reliability prediction based on Markov chains. A recent survey on UML-based approaches targeting dependability can be found in [7].

Our toolchain does not only implement a model-transformation workflow for dependability analysis, but it also aims at being reusable (at least parts of it), in order to survive the evolving MDE world.

3. PURPOSE AND REQUIREMENTS

The purpose of the toolchain we introduce in this paper is to automatically perform state-based stochastic dependability analysis starting from a high-level description of the system architecture in a UML-based language. With the term "state-based analysis" we refer to model-based dependability evaluation using state-based stochastic methods (e.g., see [23]). In such methods, a stochastic model of the system is constructed, describing its states and the possible transitions between them. State-based stochastic models are used to evaluate different kind of properties, including performance, reliability, availability, power efficiency.

Formalisms for this kind of analysis include Markov chains, as well as higher-level formalisms like Stochastic Petri Nets (SPNs) [6] or Stochastic Process Algebras (e.g., PEPA [16]). SPNs and their extensions are widely used for

VALUETOOLS 2013, December 10-12, Torino, Italy

DOI 10.4108/icst.valuetools.2013.254395

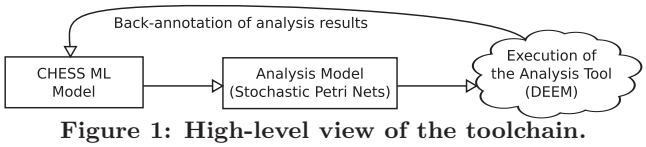


Figure 1: High-level view of the toolchain.

this kind of analysis: they feature an intuitive graphical representation, a formal semantics and a large set of supporting tools.

The context which drove the development of the tool is the ARTEMIS-JU "CHESS" project [2]. The project aimed at developing an industrial-quality MDE infrastructure that permits high-integrity embedded systems to be assembled in a component-based fashion, while retaining guarantees in terms of functional and non-functional properties. Practically, the project defined a UML-based language with specific restrictions and extensions in order to implement its system design methodology [9]. Within CHESS, the development process is supported by different kinds of analyses, which allow the feasibility of the system's design to be assessed from different points of view. In accordance with MDE principles, the analysis models are automatically derived from the high-level model that describes the system's architecture. In order to support an an iterative and incremental development process, analysis results are used to enrich the initial architectural model from which the analysis has been triggered, in a process usually called "back-annotation".

The toolchain described in this paper implements one of such techniques, namely *state-based dependability analysis*, using SPNs with general probability distributions as the analysis formalism. The reason behind this choice is due mainly to the intent of supporting: i) non-exponential occurrence of faults (e.g., for mechanical components), and ii) periodic maintenance schedules. The model is then evaluated using discrete-event simulation, using the simulator extension of the DEEM tool [8].

From a high level perspective, the plugin should be able to automatically i) generate an analysis model which complies with the architectural description of the system, ii) analyze the model for the specified metrics of interest, and iii) propagate the obtained results back in the architectural description received as input; the corresponding workflow is depicted in Figure 1. Additionally, our objective was to create a plugin as much reusable as possible, in order to be adapted with reduced effort to other contexts. This led us to define two set of requirements for the toolchain: "functional" and "reusability" requirements (Table 1).

4. TOOLCHAIN ARCHITECTURE

The designed architecture for the analysis plugin is sketched in Figure 2(a) and it is described in the following. For greater flexibility the workflow is divided into a client and a server process, communicating through a TCP/IP network. Of course, the server and client processes may reside on the same physical machine as well.

4.1 Client Process

The client has the responsibility of performing the required model transformations in order to i) generate the analysis model in a format readable by the tool, and ii) perform the back-annotation of analysis results.

The client process takes as input an architectural model from the design environment, and it generates the analysis model targeted to a specific analysis tool. The model is then transmitted to the server process, which executes the analysis tool and forwards the results back to the client. The obtained results are then back-annotated into the original architectural model that has triggered the analysis, and they can be possibly used as input for further analyses.

Table 1: Functional (F*) and reusability (R*) requirements for the toolchain.

F1	*The plugin should take as input a description of the system architecture in a UML-like language, including properties needed for the analysis and target metrics.*
F2	*If the input model contains all the necessary information, the plugin should be able to generate an analysis model for the evaluation of specified metrics.*
F3	*The plugin should be able to analyze the generated model using external tools.*
F4	*The plugin should be able to extract the results from the analysis tool, and propagate them back into the architectural model that was received as input.*
R1	*It should be possible to adapt the plugin when using different architecture description languages (e.g., UML, SysML, AADL...).*
R2	*It should be possible to adapt the plugin when using different state-based dependability analysis formalisms.*
R3	*It should be possible to adapt the plugin when using different analysis tools.*
R4	*The plugin should not depend on a specific platform or operating system.*

The transformations chain within the client process involves the use of five different metamodels ($m1 \dots m5$), and four model-transformation algorithms ($t1 \dots t4$). In the CHESS implementation, the client process is written in Java, and it is realized as a plugin for the Eclipse framework [13]; therefore, it can be used on any platform for which Eclipse is available.

4.2 Client Process – Metamodels

Architectural Model (m1)

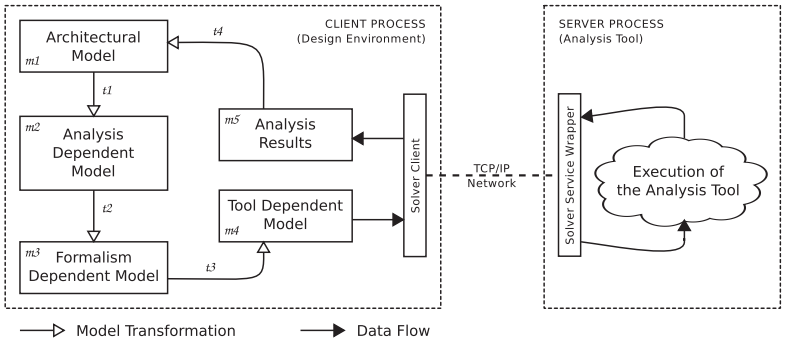
An *m1* model contains an architectural description of the system in some kind of system engineering modeling language (e.g., UML, SysML). The model should contain all the information that is needed to perform the analysis, e.g., by using some ad-hoc extension to the a general purpose modeling language, or by using a domain-specific language that is able to represent all the required information. Typically, at this level the model contains a large number of details that are unnecessary for the analysis, which are expression of different concerns of different stakeholders.

This metamodel can be reused in toolchains that use the same architectural language to describe the system architecture.

▷ CHESS Implementation. In the concrete implementation the "Architectural Model" is the CHESS ML language, a component-based architecture description language based on UML, SysML, and MARTE. In particular, state-based dependability analysis is supported by a set of language extensions that are summarized in the "CHESS Dependability Profile" [9].

The main elements supporting state-based analysis are:

- A set of predefined stereotypes used to describe common class of components, and having dependability-



(a) ***Abstract elements of the toolchain architecture***

Abstract Element		Implementation
Client Process		Java (Eclipse Plugin)
Architectural Model	*m1*	CHESS ML
Analysis-Depenent Model	*m2*	IDM
Formalism-Depenent Model	*m3*	PNML
Tool-Depenent Model	*m4*	DEEM Input File
Analysis Results	*m5*	DEEM Results File
m1 ⟶ *m2*	*t1*	ATL Module
m2 ⟶ *m3*	*t2*	ATL Module
m3 ⟶ *m4*	*t3*	ATL Query
m4,m1 ⟶ *m1*	*t4*	Java
Server Process		Java (Standalone)
Analysis Tool		DEEM Simulator

(b) ***Concrete elements in the CHESS plugin***

Figure 2: The abstract architecture of our toolchain for automated dependability analysis ((a)), and how they have been implemented in the CHESS plugin ((b)).

related attributes like the component's *fault occurence rate* or *mean time to repair*.

- The CHESS Error Model, which is implemented as a particular kind of StateMachine diagram stereotyped with the «ErrorModel» stereotype, and supports a detailed specification of faults, errors and failure modes affecting system components.
- The «Propagation» stereotype, with can be used to enrich potential propagation paths between components with propagation probability and delay.
- A set of stereotypes to model maintenance activities (both preventive and corrective).
- The «StateBasedAnalysis» stereotype, which allows metrics of interest to be specified.

A detailed description of these modeling elements can be found in [9]. The CHESS ML language is defined as a UML2 profile, and it has been implemented using the Eclipse Modeling Framework (EMF) and Ecore.

Analysis-Dependent Model (m2)

An *m2* model contains all the information that is necessary *and sufficient* to perform the intended analysis, filtered with respect to the original engineering model, and organized in a convenient way to facilitate the subsequent model-transformation steps. For example, when performing dependability analysis, only dependability-related information is retained; similarly, for performance analysis, the metamodel should be able to describe performance-related information. In other words, this model is an analysis model of the system which is not yet dependent on a specific analysis formalism. The KLAPER language of [15] is an example of *m2* model.

In developing toolchains that perform, in different contexts, the same kind of analysis this metamodel can be reused.

▷ CHESS Implementation. For the *m2* level we adopted the Intermediate Dependability Model (IDM) of [22]. The IDM has been defined exactly with the purpose of being an intermediate language to support model transformations for state-based dependability analysis.

The IDM is a lightweight component-based language to model dependability properties of systems. It allows dependability properties of components to be specified, without the additional details of software engineering languages. The elements of the IDM metamodel are grouped in five logical packages: *Statistics*, *Dependable Components*, *Threats & Propagation*, *Maintenance & Monitoring* e *Dependability Analysis*. It supports the modeling of system components and propagation paths between them, including a detailed specification of the fault/error/failure chain. Maintenance and error detection activities are also supported, as well as the ability to define the details of the measures of interest that should be evaluated on the system.

A case study of a fire detection system modeled with the IDM language has been described in [21].

Formalism-Dependent Model (m3)

An *m3* model contains an implementation of the analysis model in the formalism that has been selected for the analysis (e.g., Stochastic Petri Nets, Fault Trees, PEPA. . .). The model is however still an abstract representation and it is not yet bound to any specific analysis tool.

This metamodel can be reused in toolchains that use the same formalism for performing the target analysis technique. It should be noted that this metamodel can be reused even if the kind of analysis is different; for example, Stochastic Petri Nets can be used both for dependability and for performance analysis.

▷ CHESS Implementation. In our implementation we have selected PNML as *m3* language. The Petri Net Markup Language (PNML) [19] is a proposal for a Petri net interchange format based on XML that is under development as an ISO/IEC standard.

More in details, ISO/IEC 15909 aims to provide a standard for the representation of Petri Nets models, and it is organized in three parts, describing: 1) formal definitions and graphical notations, 2) the transfer format (i.e., the concrete PNML language), and 3) Petri net types and extensions. Such interchange format is a good choice for implementing the *m3* metamodel in our toolchain: it is an ISO standard, it is specific of the formalism selected for the analysis, and it is not tailored to any specific analysis tool.

However, currently only Part 1 [18] and Part 2 [19] of the standard have been published, while Part 3 is still under development and it has not been disclosed yet. Part 2 defines the PNML language and the way to represent basic *non-timed* P/T Petri nets, as well as the extension mechanisms to attach additional properties to Petri net elements (e.g, the firing ditribution of transitions). The actual standardized extensions to support different classes of Petri nets will

however be included in Part 3 of the standard, and are not yet available.

As such, an ad-hoc PNML extension (conforming to ISO/IEC 15909-2) has been defined to represent the class of SPNs needed for our analysis, which adds to the basic P/T Petri nets defined in the standard the following features: i) timed transitions; ii) inhibitor arcs; iii) priorities for immediate transitions; iv) weights for immediate transitions; v) metric of interest for the evaluation.

Tool-Dependent Model (m4)

An *m4* model is the concrete analysis model in a format specifically tailored to the selected analysis tool. Typically, this model is a file that can be directly provided as input to the tool.

This metamodel can be reused in toolchains that use the same tool to perform the analysis. It should be noted that it may be possible to reuse this metamodel even if a different formalism is used for the analysis. It may occur for example when using multi-formalism tools (e.g., Möbius [14]) as the analysis tool.

▷ CHESS Implementation. A "DEEM Input File", which is used as the *m4* model, is essentially a text file composed of different sections, containing (in the following order): i) the header, which is almost fixed for any input file; ii) the definition of variables to be used in the study definition; iii) the studies to be performed on the model, i.e., the combination of different values for the variables specified above; iv) the list of places; v) the list of transitions; vi) the list of arcs; vii) the list of measures of interest that should be evaluated.

Analysis Results (m5)

An *m5* model is a model describing the results provided by the analysis tool, and it is used for the back-annotation process. As above, when using multi-formalism tools, this metamodel can be reused even if the adopted formalism varies. Moreover, it can be reused even when different tools are used for the analysis, but use some standard interchange format for the produced output (e.g., CSV, XML).

▷ CHESS Implementation. Similarly to the input file, the "DEEM Results File" is also a text file, containing the results of the evaluation. More in detail, the file contains: i) a header, which is almost fixed, ii) the parameters that have been used to run the analysis, including model parameters and simulator parameters, ii) the time for which the analysis has been run, and finally iv) a set of evaluated metrics.

For each metric the tool provides its mean, as well as the confidence interval, and the number of samples on which it has been computed.

4.3 Client Process – Transformations

Filtering (t1)

The first model-transformation has the task of filtering out the information required for the analysis from the mass of information that is typically present in the architectural model of the system. Usually, this is the most complex algorithm of the entire chain, since it requires to navigate the entire engineering model, which may consist of several different diagrams, and relate concepts that refer to the same system entities. This model-transformations is applied on *m1* models to generate *m2* models.

Typically, in the resulting *m2* model the needed information is spread across multiple domain elements (to facilitate futher transformation steps), while in the original *m1* model it is highly aggregated (for modeling convenience).

▷ CHESS Implementation. The full transformation algorithm to generate an IDM model starting from an CHESS ML model is defined in [10, 22] and it is not described here because of space constraints. The transformation algorithm can be summarized in the following steps: i) projection of atomic components, i.e., components for which the internal structure it not considered for the analysis; ii) projection of "error models" associated with components; iii) projection of composite components; iv) projection of propagation paths; v) projection of maintenance information; vi) projection of metric of interest. In each step, a set of elements and relation in the IDM model are created.

This transformation step is implemented as a "Module" in the ATL language [4].

Analysis Model Implementation (t2)

The second transformation implements the analysis model in the selected analysis formalism. The definition of this model-transformation step requires an expert in the selected analysis technique, and knowledge of the analysis formalism. This model-transformation is applied on *m2* models to generate *m3* models.

▷ CHESS Implementation. From an high-level perspective, the algorithm is composed of the following steps: i) projection of components, ii) projection of threats, iii) projection of propagation relations, iv) projection of activities, v) projection of analysis metrics. This transformation step is detailed in [10] and it is implemented as a "Module" in the ATL language [4].

Code Generation (t3)

The third model-transformation has the task to generate the actual input file needed for the analysis tool. For this reason, this step is typically more oriented towards code generation rather than model-transformation, since the final goal is to generate a source file that should be read by the adopted analysis tool. This model-transformation is applied on *m3* models to generate *m4* models.

▷ CHESS Implementation. The DEEM Input File is generated from the PNML model by looping through the list of places, transitions, and arcs, and generating a string for each of them: each element of the PNML model corresponds to a specific string in the DEEM Input File. The first three sections of the file (header, variables, and studies) are mostly fixed.

This transformation step is implemented as a "Query" in the ATL language [4]. While ATL modules perform model-transformations, queries are used to compute primitive values from source models. When *strings* are the primitive type, ATL queries can be used to perform code generation.

Back-Annotation (t4)

The back-annotation is a particular kind of model-transformation that has the task to propagate the results of the analysis back into the model that has triggered it. This transformation takes as input also the original model, which is refined with the new information, i.e., it is applied on a (*m1*,*m5*) pair of models, to generate a modified *m1* model.

▷ CHESS Implementation. This transformation step takes as input the specific output of the DEEM tool and the CHESS ML model that triggered the analysis, and modifies the latter by adding the results of the analysis.

At UML level (*m1* model) the metric to be evaluated is defined by means of the «StateBasedAnalysis» stereotype. The name of the UML classifier to which the stereotype is applied provides the search key to lookup the results value in the DEEM Results File. In the current implementation, once the resulting value has been identified, it is copied back the obtained result is back-annotated in the *measureEvaluationResult* attribute of the «StateBasedAnalysis» stereotype that defined the metric. Given its simplicity, this transformation step is implemented as pure Java code.

4.4 Server Process

The server process has the task of actually executing the analysis tool on the model generated by the client process, and communicate the results back to it. Having a separate process for executing the analysis tool has a number of important advantages.

First of all, the user is not constrained to the platform required by the selected analysis tool. Complex analysis tools often require a specific environment in order to work properly, i.e., specific operating systems, libraries, or tuning of system configurations. With this approach, the analysis tool can be installed on a properly configured ad-hoc machine (possibly even a virtual machine), while the user can continue to use its current environment. Second, it lowers the hardware requirements of the user machine, since it makes it possible to move model evaluation, which is typically a resource-intensive task, to a dedicated machine. At the same time, this approach does not not prevent setting up a local-only configuration. Finally, this approach facilitates the management of licensing issues; for example it is possible to distribute the code of the frontend as open source, even if the backend relies on some proprietary tool.

▷ CHESS Implementation. The server process is written in Java as well, and it consists in a wrapper to the DEEM Simulator; it implements the TCP/IP communication with the client, and interacts with the tool. The server starts by listening on a predefined TCP port (9977), and it waits for connections from client processes.

The server is multi-threaded, thus allowing multiple instances of the DEEM Simulator to be executed in parallel, taking advantage of multi-processor or multi-core systems. When a new client connects, the server starts a new thread to handle the connection, and returns in a listening state. The thread receives the *m5* model (i.e., a "DEEM Input File") from the client and saves it to a temporary folder on the server machine. To avoid conflicts, the name of the folder is derived from a combination of the current time on the server, and the IP address of the client. The thread then runs the DEEM Simulator with the provided input file, and waits for the analysis to complete. While the analysis is running, the server periodically updates the client on the current progress. Once the simulation is finished, the results file transmitted back to the connected client, the temporary directory is removed, and the connection is closed.

5. REUSABILITY DISCUSSION

While it easy to verify that functional requirements (F1–F4) are satisfied, we provide here some insights on the reusability of the developed toolchain (requirements R1–R4).

The adaptation of the toolchain to ADLs other than CHESS ML (**R1**), e.g. AADL, SysML, requires to modify only few elements of the architecture, namely *m1*, *t1*, and *t4*. All the other elements (including the server process) can be reused as they are. Of course, the new *m1* language should be able to express all the dependability information that is needed for the analysis. Similarly, when reusing the toolchain with other evaluation tools for SPNs (**R3**), metamodels *m1*, *m2*, *m3*, as well as model transformations *t1* and *t2* can be completely reused.

Also when moving to another analysis formalism (**R2**), e.g. PEPA, some of the existing toolchain elements can be reused. In particular, metamodels *m1* and *m2*, as well as the associated *t1* transformation can be reused as they are. In certain cases it may be possible to reuse also *m4*, *m5*, *t3*, *t4*, e.g., when the analysis tool is a multi-formalism tool supporting the new formalism. Although the need to reuse the toolchain for other analysis purposes was not part of initial requirements, parts of the toolchain can in principle be reused also for other kind of analyses. For example, SPNs and similar formalisms are also used for performance, and other kinds of analyses. In such case, metamodels *m3*, *m4*, *m5*, transformations *t3*, *t4*, as well as the server process can be reused. Of course, the source language *m1* would be a different language containing information for that particular kind of analysis, which would then be filtered into a different analysis-dependent intermediate model (*m2*).

Platform independency (**R4**) is achieved by using Java, and by dividing the toolchain into a client and a server process. Although the server process is bound to the platform required by the evaluation tool (Linux in case of the DEEM Simulator), the client process runs within the Eclipse framework [13], which is available for a wide range of platforms. The final user of the toolchain, which will interact with the client process only, is therefore not forced to use any specific platform.

6. CONCLUDING REMARKS

In this paper we introduced the "CHESS Plugin for State-based Analysis", an Eclipse plugin for automated state-based dependability analysis. Its architecture is based on modularity and reusability, to adapt to the evolving MDE world. Another important benefit of the adopted toolchain architecture is the convenience in its implementation: the different elements of the plugin can be developed in isolation, and can therefore be assigned to different teams or individuals, possibly based on their skills in different areas of expertise (e.g., metamodeling, model-transformation, Java coding).

The client process of the toolchain described in this paper is available as the "CHESS Plugin for State-based Analysis", and can be freely downloaded from the website of the CHESS project [2]. The server process is not provided for download on the website; however, at the time of writing, an instance of the server process running the DEEM Simulator can be reached at the TCP address `rcl.dsi.unifi.it:5900`, thus allowing the use of the whole toolchain. The application of the plugin on a real case study is also showcased in a demonstration video [2]. The case study showcased in the video is an onboard system in the railway domain [10]. Using the toolchain described in this

paper, the reliability of such system is automatically evaluated, using different parameters.

A first benefit of its reusable architecture will be the reuse of the developed toolchain in the recently started CONCERTO project [3], which will extend and consolidate the work developed within the CHESS project.

Acknowledgment

This work has been partially supported by the ARTEMIS-JU CONCERTO project (n.333053), by the TENACE PRIN Project (n. 20103P34XC), funded by the Italian Ministry of Education, University and Research, and by the RACME-MAAS project, funded by the Tuscany Region within the framework POR CReO FESR. The authors would like to thank Stefano Puri and Nicholas Pacini for their support in the implementation of *t1*, and Fabio Duchi for the initial implementation of *t2*.

References

[1] N. Addouche, C. Antoine, and J. Montmain. "UML models for dependability analysis of real-time systems". In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 6. 2004, pp. 5209–5214.

[2] *ARTEMIS-2008-1-100022 CHESS: "Composition with guarantees for High-integrity Embedded Software components aSsembly"*. `http://www.chess-project.org/` (Accessed: 13/11/2013).

[3] *ARTEMIS-2012-1-333053 CONCERTO: "Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multi-core Systems"*. `http://www.concerto-project.org/` (Accessed: 13/11/2013).

[4] *Atlas Transformation Language (ATL)*. `http://www.eclipse.org/atl/` (Accessed: 13/11/2013).

[5] A. Avižienis et al. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing* 1 (2004), pp. 11–33.

[6] G. Balbo. "Introduction to Stochastic Petri Nets". In: *Lectures on Formal Methods and PerformanceAnalysis*. Vol. 2090. LNCS. Springer, 2001, pp. 84–155.

[7] S. Bernardi, J. Merseguer, and D. C. Petriu. "Dependability modeling and analysis of software systems specified with UML". In: *ACM Comput. Surv.* 45.1 (Dec. 2012), 2:1–2:48.

[8] A. Bondavalli et al. "DEEM: a Tool for the Dependability Modeling and Evaluation of Multiple Phased Systems". In: *IEEE Int. Conference on Dependable Systems and Networks (DSN)*. 2000, pp. 231–236.

[9] *CHESS Deliverable 2.3.2 "Multi-concern Component Methodology (MCM) and Toolset"*. 2012.

[10] *CHESS Deliverable 3.2.2 "Transformations and analysis support to dependability"*. 2011.

[11] V. Cortellessa, H. Singh, and B. Cukic. "Early reliability assessment of UML based software models". In: *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*. Rome, Italy: ACM, 2002, pp. 302–309.

[12] A. D'Ambrogio, G. Iazeolla, and R. Mirandola. "A method for the prediction of software reliability". In: *6th IASTED Software Engineering and Applications Conference (SEA'02)*. 2002.

[13] *Eclipse Platform*. `http://www.eclipse.org/` (Accessed: 13/11/2013).

[14] S. Gaonkar et al. "Performance and dependability modeling with Möbius". In: *SIGMETRICS Perform. Eval. Rev.* 36 (4 2009), pp. 16–21.

[15] V. Grassi, R. Mirandola, and A. Sabetta. "From design to analysis models: a kernel language for performance and reliability analysis of component-based systems". In: *Proc. of the 5th international workshop on software and performance (WOSP '05)*. ACM, 2005, pp. 25–36.

[16] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[17] A. Immonen and A. Niskanen. "A tool for reliability and availability prediction". In: *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. 2005, pp. 416–423.

[18] *ISO/IEC 15909-1:2004, "High-level Petri nets – Part 1: Concepts, definitions and graphical notation"*. 2004.

[19] *ISO/IEC 15909-2:2011, "High-level Petri nets – Part 2: Transfer format"*. 2011.

[20] I. Majzik, A. Pataricza, and A. Bondavalli. "Stochastic Dependability Analysis of System Architecture Based on UML Models". In: *Architecting Dependable Systems*. Vol. 2677. Lecture Notes in Computer Science (LNCS). Berlin, Heidelberg, New York: Springer-Verlag, 2003, pp. 219–244.

[21] L. Montecchi, P. Lollini, and A. Bondavalli. "Dependability Concerns in Model-Driven Engineering". In: *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. Newport Beach, USA, 2011, pp. 254–263.

[22] L. Montecchi, P. Lollini, and A. Bondavalli. "Towards a MDE Transformation Workflow for Dependability Analysis". In: *IEEE International Conference on Engineering of Complex Computer Systems*. Las Vegas, USA, 2011, pp. 157–166.

[23] D. Nicol, W. Sanders, and K. Trivedi. "Model-based evaluation: from dependability to security". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 48–65.

[24] A.-E. Rugina, K. Kanoun, and M. Kaâniche. "The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation". In: *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*. 2008, pp. 85–90.

[25] D. C. Schmidt. "Guest Editor's Introduction: Model-Driven Engineering". In: *Computer* 39.2 (2006), pp. 25–31.

[26] M. Walter, C. Trinitis, and W. Karl. "OpenSESAME: an intuitive dependability modeling environment supporting inter-component dependencies". In: *Proc. 3rd Pacific Rim International Symposium on Dependable Computing (PRDC'01)*. 2001, pp. 76–83.