Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 6, 801-810 (2020)

www.astesj.com

ASTESJ

ISSN: 2415-6698

A Model-Driven Approach for Reconfigurable Systems Development

Ismail Ktata[*,1,2], Naoufel Kharroubi[1]

[1]*Computer Science Department, Khurma University College, Taif University, Taif, 21944, Kingdom of Saudi Arabia*

[2]*CES laboratory LR11ES49, National Engineering School, University of Sfax, Sfax, 3038, Tunisia*

ARTICLE INFO

Article history:
Received: 27 September, 2020
Accepted: 18 November, 2020
Online: 24 November, 2020

Keywords:
Model-Driven Engineering
Ontology
Semantic Rules
SWRL Reasoner
Reconfigurable Architecture
Scheduling

ABSTRACT

Reconfigurable systems are considered as promising technology that enables the design of more flexible and dynamic applications. However, actually existent design flows are either low-level (so complex) or they lack support for automatic synthesis. In this paper, we present an ontology-based modeling approach for reconfigurable systems. Our approach is based on model-driven engineering process and addresses dynamic features on application-level enabling early exploration of the execution behavior of the system. The model is characterized by a logical and syntactical description conform with application domain knowledge and respect a number of metamodel constraints. These elements are semantically presented by an ontology language. We successfully implemented a system model with several tasks and resources and made scheduling test for the application graph.

1. Introduction

The system designers have been relying two choices for computing system implementing, varying largely in terms of performance and flexibility [1]. One is considered for the general-purpose systems built to accommodate flexibility problem. The compromise, in this case, is the system performance because of the absence of hardware resources dedicated for specific application. The second is carried out to develop an optimized hardware circuit dedicated and adapted to meet the demands of a specific application. Such system will be at issue following a minor need for change, because a costly redesign will be carried out in order to adapt new system parameters.

The present work introduces a new paradigm based on configurable computing that can solve performance problems as well as flexibility problems, and hardware design. We offer the performance of dedicated circuits by changing hardware configurations. The reprogram ability of the computing hardware explains that new architecture needs to emulate different computer architectures [2]. Mapping software operation is facilitated with getting the ability to reconfigure its connections. A number of manufacturers introduce FPGAs characterized by partial and dynamic reconfiguration abilities [3]. When a part of FPGA logic resources and interconnections is reconfigured while the remainder device continues operating, the partially reconfigurable FPGA is. Dynamic reconfiguration deals with logic and interconnections reconfiguration in FPGA from an application to the next, that is what we call dynamic reconfiguration or run-time reconfiguration (RTR) [4]. In such situation, without adding external physical (hardware) resources, in a static or dynamic state at run time, when the system can change/adapt its basic computational structure to suit the changing requirements of a program, this is referred to as a Configurable Computing System (CCS) [5]. Our current CCS realizations has a general-purpose microprocessor augmented set with an FPGA. Using reconfigurable logic as a resource shared by multiple applications, a hardware operating system (HwOS) facilitates system setting operations to overcome the problems due to real time considerations. Real time multi-task systems' requirement is tasks scheduling to complete their execution depending on their deadlines [6]. Tasks known periodic in advance are scheduled off-line. Instead, irregular and unknown tasks in advance are scheduled on-line. That makes specific constraints concerning computing resource recovery, preemption and interruption time, tasks' migration between different executing resources, limited memory and power use, hardware costs, etc. [7]. To overcome such complexities, high-level development techniques are required. That researching methods dealing with modeling and simulation of complex and heterogeneous systems are called model driven engineering approaches.

[*]Corresponding Author: I. KTATA, i.ktata@tu.edu.sa, ismail.ktata@enis.rnu.tn

Model-Driven Engineering (MDE) methods tackle with the specific constraints of a specific domain and execute model checking that may identify many errors and limit failure risks at an early period of the system's life cycle [8]. In this context, the present paper focuses on the use of an ontology model [9] in order to describe the dynamic behaviors of the components of a system within the schedulability analysis and verification context [10].

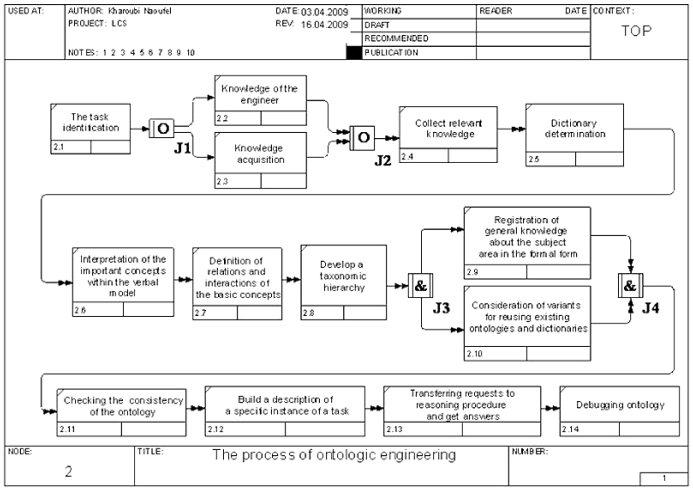

Figure 1: Process of Ontology Engineering

The proposed ontology is based on component behavior representation (Figure 1). Motivation behind our proposal is:

- System abstraction at a high level expressed in an ontology language enabling a prior evaluation and analysis of the impacts of possible online changes and reconfigurations.
- As a consequence of different multiple tools use, probable incoherencies may appear between design model and safety (normal) model.
- For architectural design, there is a heterogeneity in textual description, semantic representations, and syntax of the existing modeling languages.
- The need of reusable knowledge repository.
- Generating a safe executing model to be implemented.

For safety assessment, we elaborate, an ontology-based model description [11], [12]. We have several error behavior items that are stored in the ontology. Such reusable elements are a result of nominal models mixed with failure modes: each object is executable design system specification components under nominal behavior models. We transformed into formal safety models the obtained extended system models. A series of constraints (such as temporal and precedence constraints) are then added to the model in order to make it eligible to be analyzed and simulated by different tools, and so be reliable to making right decisions. All constraints are mapped onto an OWL domain ontology [13] written with the Protégé editor (Figure 2) [14]. Incorporated OWL description checked, the inference rules logic reasoned, and the ontology are able to detect mainly semantically inconsistent parts [15] as well as lack of model elements.

All errors interpreting cost are avoided: these errors encountered in the extended system model formal representation, are hardly to conceive. During the work, proposed architecture design is formulated in the Architecture Analysis and Design Language (AADL) [16], [17]. As a result, precise execution semantics are expressed for modeling software systems and their target hardware architecture. Safety models are generated in the AltaRica formal language [18]. We have opted for AADL and AltaRica languages, but we might as well have opted for other similar formalisms. The AADL language is adapted to describe functional and software architectures of embedded systems. The AltaRica language is an event-centric formal language suited for safety functioning modeling. Its translation is implemented in the form of model transformation. It would also have been possible to use other software such as ATL for example. There are series of analysis tools processed this language [19]-[21]. The proposed approach can be applied to validation contexts.

This work takes advantage of all previously cited domains: it is related to the scheduling problem in reconfigurable systems with a high degree of dynamicity. To overcome such complexity, we propose to work at a high abstraction level using an MDE approach combined with ontology description. The considered ontology language is OWL which gives a formal description of the database and the knowledge base and a semantic hierarchy of the whole system domain. In addition, after implementing the application constraints as rules, the ontology reasoning engine enhances the knowledge/data bases with new inferred axioms and data. All this is in order to have a perfect scheduling decision that takes into account all parameters/conditions and is flexible and dynamic according to the rule changes that may take place.

Related works are reviewed in Section 2. The domain ontology and the model-driven engineering process is presented in the Section 3. Section 4 is for the model transformation process and in the Section 5 we deal with the experimental case study. Then we conclude with a summary of the proposed process and we comment on its potential impact.

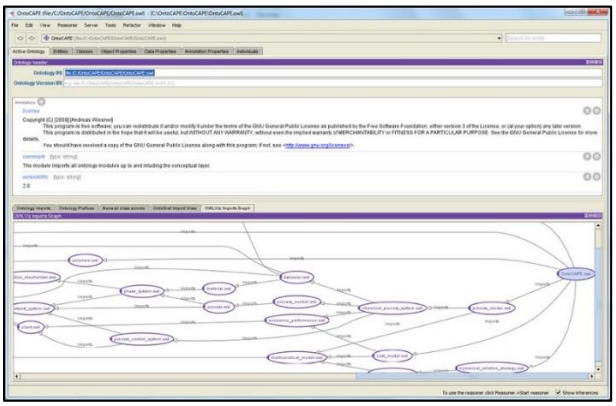

Figure 2: Protégé Framework Interface

2. Related Works

Reconfigurable embedded systems are very complex and include different Hw/Sw components. This led researchers to address new high-level design approaches to abstract that complexity, to facilitate development and to give more degree of flexibility when making changes and fixing errors. In this way, most research works on two principle modeling approaches: Architecture Analysis & Design Language and MARTE

(Modeling and Analysis of Real-Time Embedded systems) [22]. Those two methods consider only predefined reconfiguration models, so they cannot respond to the dynamicity of reconfigurable systems. MDE methods cope with that and permit to transform the model to another model until reaching a target one which responds better to a specific architecture and application domain. Moreover, the ontology is considered as a language that gives a formal description of a variety of knowledge (e.g. properties, features, relations) and different components (e.g. data, entities) forming a specific domain. Ontology languages are used in many domains, such as system engineering, semantic web application, logistics, artificial intelligence, system architecture, etc. It combines different techniques (e.g. semantic search, model matching) and formulates rules in order to tackle with systems complexity and domain conflicts. In [23] authors focus on e-health applications and propose a description for the whole software based on model-driven-architecture (MDA) approach. They exploit all steps of that approach, but especially the first stage called Computation-Independent-Model, with the integration of domain ontologies to represent particular information and all structures and relations existing in the system. In [24] authors apply the MDE approach on semantic web applications and provide an output model that generates annotated user interfaces and reduces some repetitive processes that may occur on the web pages.

To our knowledge, works related to model transformation based on an ontology language are not abundant. The main work related to that subject is [25]. Even in that paper, the purpose was to enhance cross-organizational modelling by adopting automatic generation then evolution of transformed model, a concern which is out of the scope of this paper. Two works on AADL transformation to AADL Altarica were made [26]: firstly, a model transformation [27], [28] based on system hardware architecture from the AADL and selected model reused from a project to another. Secondly, transformations were enriched with failure propagations written in AADL Error [29] and derived from AADL code. Transformation operation can be done if analysis and component relationships are defined. The important difference of our ontology-based model can be resumed in the semantic connection between the AADL and Altarica languages which permits to overcome the difference between them in term of syntax and scope [30], [31].

The focus of this work is to transform traditional representations of reconfigurable systems and make them more significant and related to their application domain. This led us to implement MDE approach combining the ontology languages (describing domain constraints in semantic rules) and applied them to solve scheduling problem.

3. Ontology Domain and the Model-Driven Approach

We propose an engineering process to model a dynamic application mapped on configurable computing system. The process is based on (1) modelisation of the application to be created, (2) analysis and verification of the rules relating different models, and (3) test of the schedulability of the application from the resulting model. We assume that a CCS is computer-based system consisting of hardware and software components, integrated in a physical environment, and requiring different timing/ressource requirements on the final outcome.

3.1. Application Modeling Approach

We aim in this work to exploit domain knowledge in design models. We want to formalize the system domain design, its annotations and its knowledge in a single model. To do this, we followed a four-step approach (shown in Figure 3):

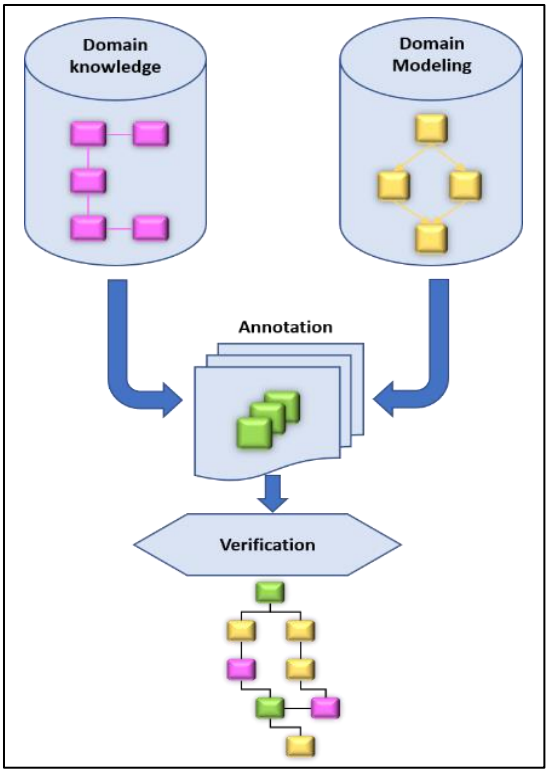

Figure 3: Proposed Application Modeling Approach

- The first step is to formalize and explicitly define the information (knowledge) of the domain of application in a formal ontology (entities, relationships, constraints, etc.). The properties of the ontology thus designed are domain-linked and completely independent of any context of use. In many cases, this ontology can be constructed from pre-existing standard ontologies.
- The second phase concerns the definition of design models. This means the formal definition of the properties corresponding to a given specification of the application domain.
- The third step, called annotation, is to define specific relationships between the different entities of the two previous models (design and ontology). Different relationships are available, they have their specific properties and describe specific rules related to the application domain.
- A verification step is required in any approach. This consists of validating the context of design models and domain properties expressed in the ontology, as well as the logic of the rules defined in the annotation step.

At the end, the final designed model has the advantage of well describing the domain knowledge with more details and properties derived from the ontology and rules generated during the annotation.

3.2. Scheduling Model

As mentioned above, we consider highly dynamic applications with soft real-time constraints. With such applications, missing a deadline does not cause catastrophic consequences, but only a performance degradation (quality of service parameter). In order to schedule these applications, we need a description that exhibits dependencies between the tasks and their own characteristics.

Each task is defined as a class in the ontology model. It represents a computational activity that needs to be performed according to a set of constraints (called slots in the ontology model). Each task i, for $i = \{1, 2, \ldots, k\}$ is defined by:

- ξ represents a set of active tasks T_i forming the application. Considered tasks could be sporadic and aperiodic.
- a_i : stand for the arrival time of task T_i.
- C_i : stand for the maximum computation (execution) time of task T_i.
- c_i : stand for the computation time of task T_i, i.e., the remaining worst case execution time WCET is needed for a computing elements (processor or logic circuit), at the current time, to complete the execution of task T_i without interruption.
- d_i : stand for the absolute deadline of task T_i.
- D_i : stand for the relative deadline of task T_i.
- S_i : stand for the first start time of task T_i.
- s_i : stand for the last start time of task T_i.
- f_i : stand for the estimated finishing time of task T_i.
- L_i : stand for the laxity of task T_i.
- R_i : stand for the remaining time of task T_i.

Baruah *et al.* [32] present a necessary and sufficient test for synchronous tasks with pseudo-polynomial complexity. For that reason, we present the following equations defining relations among the parameters defined above:

$$d_i = a_i + D_i \tag{1}$$

$$L_i = d_i - a_i - C_i \tag{2}$$

$$R_i = d_i - f_i \tag{3}$$

$$f_1 = t + c_1;\ f_i = f_{i-1} + c_i \ \forall\, i > 1 \tag{4}$$

$$R_1 = d_1 - t - c_1 \tag{5}$$

$$R_i = R_{i-1} + (d_i - d_{i-1}) - c_i \tag{6}$$

For any other task T_i, with $i > 1$,

$$f_i = f_{i-1} + c_i \tag{7}$$

and, by equation (3), we have:

$$R_i = d_i - f_i = d_i - f_{i-1} - c_i = d_i - (d_{i-1} - R_{i-1}) - c_i = R_{i-1} + (d_i - d_{i-1}) - c_i \tag{8}$$

The information above present the major inputs of the task class ontology. In our model, we define the scheduling ontology basic classes as follows:

- Basic scheduling elements (including tasks, resources and activities) and relation between them.
- Basic required components and constraints (including hard/soft constraints, temporal restrictions and resource limitations).
- Ontology of instances: each class has one or more related subclasses.

The Protégé editor provides the definitions for basic object types and properties such as application graph (Figure 4), tasks set properties (Figure 5), relations, numbers, etc. Figure 6 shows the data properties of a created task refereed to its temporal features in the ontology framework. Figure 7 is related to processor creation, where processor is either a software processor or a CLB for hardware task. Figure 8 represents slots where defined the properties of a task, their status and needed executing resources.

We mean by resources all hardware components that are responsible for the execution of tasks. Resources could be a processor (for software tasks) or a discrete number of logic circuit called CLB (configurable logic blocks) for hardware tasks in an FPGA architecture. Resources are defined in separate classes and are characterized by their execution time and/or number of CLB.

Rules and constraints are modelled as distinctive class. The class constraint has the same definition for both hard and soft constraints, and is applied to tasks or resources. The hard constraints are the rules that cannot be violated under any conditions, while the soft ones have to be satisfied by the completion time of the scheduler. For example, if we consider the temporal constraints in the EDF (Earliest Deadline First) scheduler [33], the semantic rule would be:

$$\begin{aligned} &\text{Task}\,(?T_i) \wedge \text{has\_D}_i\,(?T_i, ?d_i) \\ &\rightarrow \text{sqwrl:select}\,(?T_i) \wedge \text{sqwrl:orderBy}\,(?d_i)\,, \end{aligned} \tag{9}$$

as EDF is a preemptive algorithm which assigns the highest priority to be executed to the task T_i that has the lowest deadline D_i.

The both soft and hard constraints are applied to a task as well as a resource through the class schedule, which helps to satisfy schedulability conditions of the application execution.

3.3. Model Implementation

The Ontology-based model engineering architecture (Figure 9) is based on the Architecture Analysis & Design Language (AADL) and describes how the proposed ontology organizes the error models and the components into structural and functional interdependent hierarchies. AADL is considered as a textual

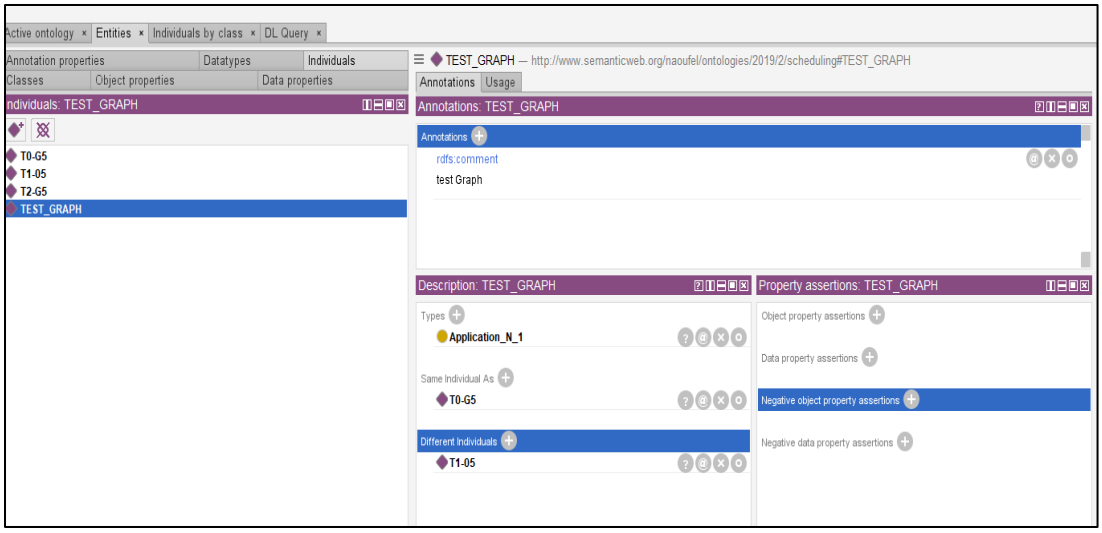

Figure 4: Application Graph Description

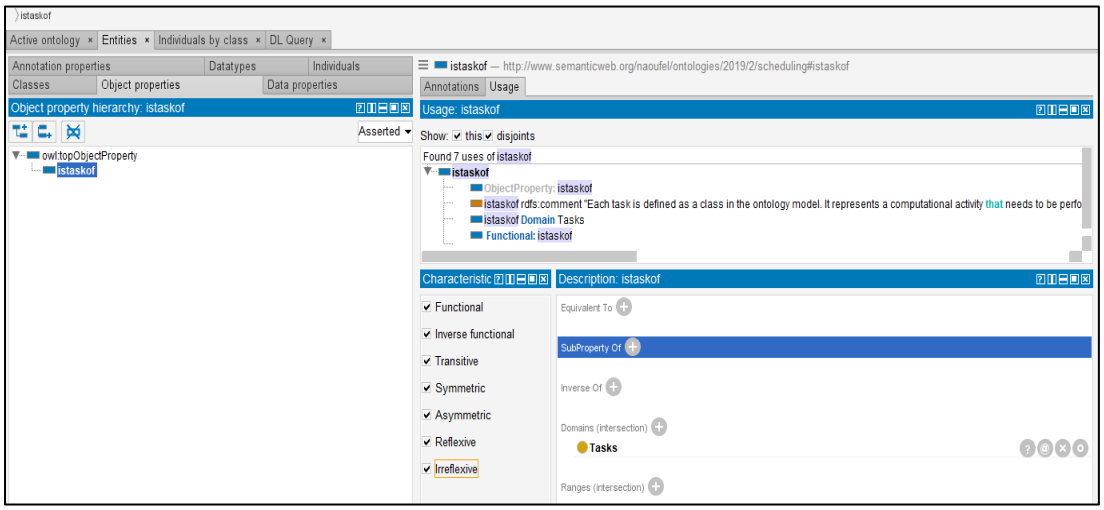

Figure 5: Task Class Properties

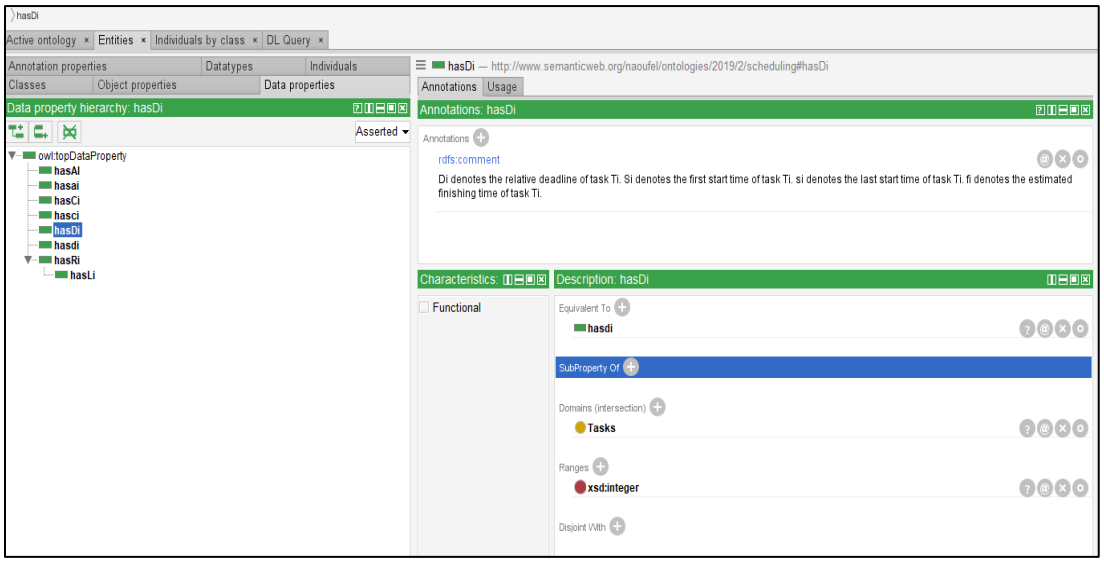

Figure 6: Task Creation With its Temporal Features

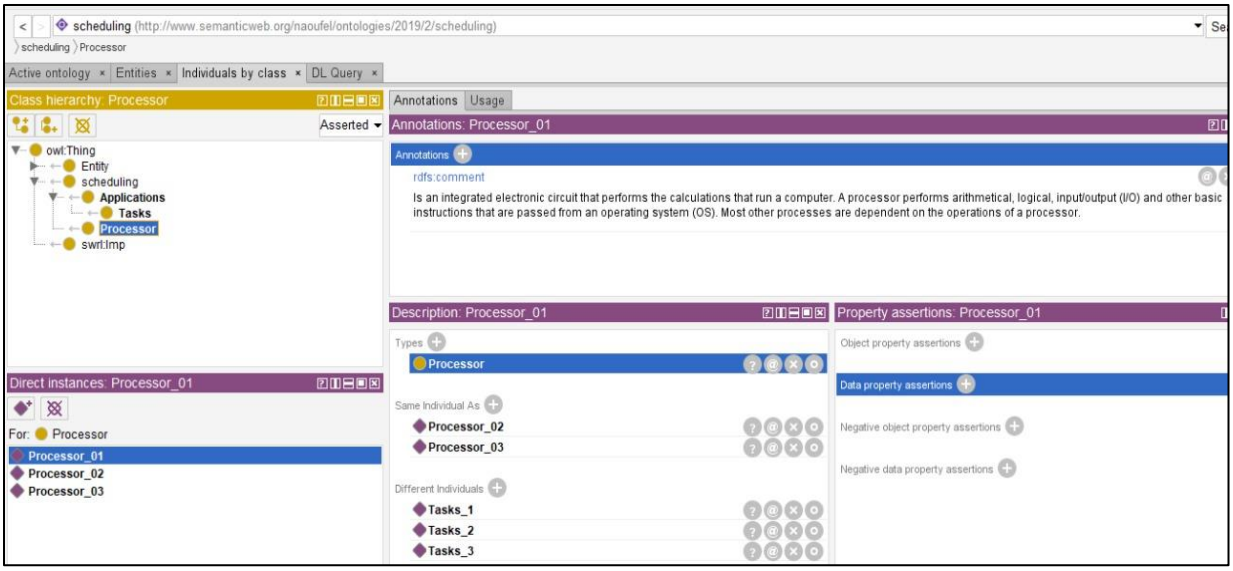

Figure 7: Processors Class Creation

language and graphical one too. It is used to specify the software (SW) and hardware (HW) architecture for the needed critical level of real time systems [34].

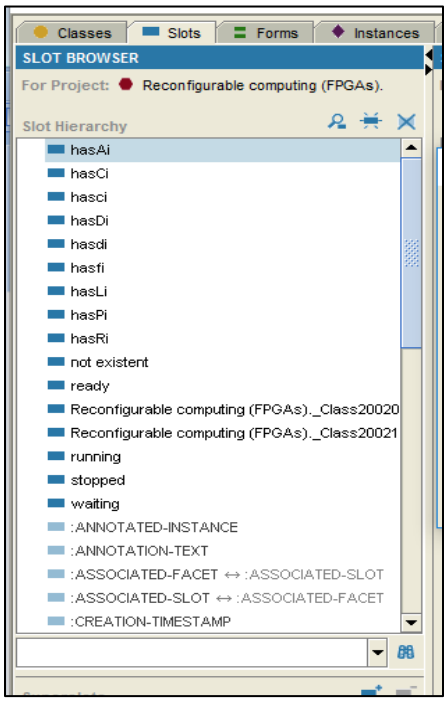

Figure 8: Task Properties, Status and Resources

In AADL language, each system is defined by its components. Each component identifies a number of elements of the actual system architecture. In our case, systems based on reconfigurable architecture are composed of a hardware part and software one. The AADL language permits to define software components (process, thread, data), as well as hardware components (device, processor, memory, etc.). Moreover, this design language defines precise legality rules that control the different component assemblies, and this in both static and dynamic (on-execution) way. To describe the communication between components, AADL defines the connectors and ports. A port performs a particular task in the context of the connector that connects components. All instances of connectors (including their ports and roles), compositions of components and components define the implementation operation. To enable linking flows to internal states, a model describes what initial states may evolve and it includes events, states, transitions to perform various analyses with different constraints on the model.

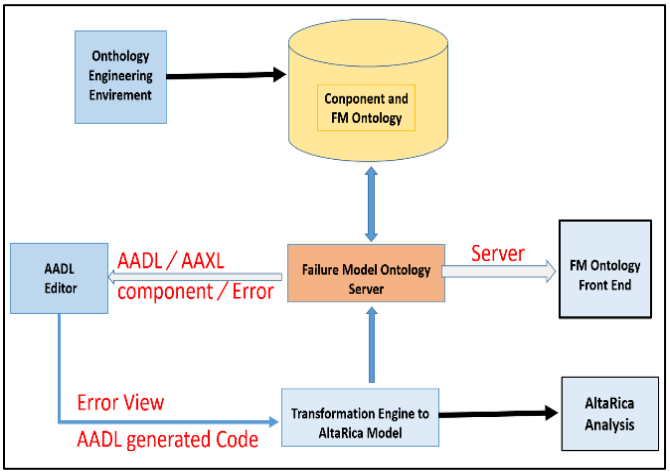

Figure 9: Model Engineering Process Based on Ontology

The objective of the design of the ontology is to prove the correlation of constructs related to the AADL specification, the retrieval, the storage operation's permission, and to offer the desirable reasoning capabilities among them. We specified the ontology [33] in the OWL DL fragment of OWL and we applied Pellet DL reasoner combined with the Protégé ontology editor, and this to customize inference rules' determination.

The proposed ontology is mainly based on three aspects [36]. Firstly, specify all necessary constructs for AADL core that permit to model components (e.g. data) with their corresponding characteristics and so facilitate ontology to add or extract new components. Secondly, for each AADL error, ontology offers the needed concepts to represent error models in terms of states and events, as hardware errors, computational problems and memory

exceptions, and thirdly, the ontology offers the connections and subcomponents (constructs) for component implementation. To achieve this task, we use ontology-based representation of component to detect rules custom inference [37]. The system detects three inconsistencies types making certain formal analysis models invalid: error of incomplete or missing transition; conflict transition that may occur when more than one event is triggered simultaneously; and finally, error in the state of a component due to an error in the failure scheduling scenario. The rules related to inconsistent semantics permit new inconsistent levels to be added. Meta-modeling design approach followed the ontology and determines instantiations and implementation steps. We represent the AADL component as an ontology class. The error models are presented as the ontology hierarchy subclass. The proposed model-driven engineering process enforces rules and constraints on the associations between the error models and the components' constraints. It permits to select and resume the failure modes from the error model hierarchy, then associate them with the nominal mode. The ontology model is then checked for possible component inconsistencies and transform the extended architecture model to the safety one and finally analyze the safety model with tools that provide model checking and simulation.

4. Model Transformation Process

For discovering structurally equivalent constructs, existing modeling experience is considered a basic step for model transformation, and the first set of rules is dedicated to the transformation of AADL components into AltaRica nodes. The transformation rules are therefore driven by the mapping of knowledge with the constructs used through the underlying domain ontology. The benefit of the transformation from AADL models to AltaRica is to expand the set of safety evaluation tools for AADL. Hence, all system components (tasks, data, CLBs, processors, memory, etc.) are transformed to AltaRica nodes while conserving their same original features.

Then, AltaRica state statements are generated based on the AADL components' properties. We manage a set of rules which concern the components' error models. The AADL error, states and transitions are transformed to corresponding AltaRica which are filled with assignments found in the matched AADL. Another set of rules focuses on failure propagation. Since no support failure capabilities are made, a transition declaration creates additional component variants and other transformation rules set related to the used architecture design process and the associated AADL editor is made. The problem is that the editor tool follows a control-flow based approach, which make a semantic gap compared to the data-flow approach of the AltaRica specification.

An AADL component can be of two kinds from a safety point of view: either its properties are filled or are not filled *{lost}*. In the following, we will focus on the generic transformation. The names of the variables vary according to the connection ports of the component and the formulas of the component assertion depend on the data flow path. Indeed, any component of an AADL system:

$C = \{D^{in}, D^{out}, K, Ty\}$, where D refers to data dependencies related to the task data flow, K refers to the knowledge related to properties of safe operation of the component, and Ty refers to the type of the component,

is translated into an AltaRica node:

$N = \{ F, S, D^{in}, D^{out}, \Sigma, \sigma, I\}$, where F is a field of finite values of the variables, S is for state, Σ is a set of events, σ is an affirmation function $S \times D^{in} \rightarrow D^{out}$, and I refers to initial conditions,

such as:

- the state variable s takes its *{correct}* value if the component is working normally otherwise its value is equal to the name of the failure mode *{lost}* declared in AADL ;
- the dependency connections D^{in} and D^{out} form the AADL (and also the AltaRica) component interface;
- a failure transition is produced by events $Evts \subset \Sigma$ leading to a failure mode;
- for each outgoing flow variable $Out_j \in D^{out}$, we consider the set $\{In_i \mid (In_i, Out_j) \in D^{in} \times D^{out} \}$ of incoming flow variables on which the Out_j variable depends. Then, if we consider only the loss of a component, the statement associated with Out_j is the following Out_j = *{if s = correct and* In_i *= correct then correct, else lost)*;
- finally, we consider that, initially, components states are correct *I(s) = {correct}*.

Considering an AADL system $\boldsymbol{\psi} = \{D^{in}, D^{out}, \boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_n, A, R\}$, $\boldsymbol{\psi}$ is transformed into an AltaRica node $N = \{N_0, N_1, \ldots, N_n, V\}$ with:

- each subsystem $\boldsymbol{\psi}_i$ is transformed into an AltaRica N_i node;
- N_0 has for dependencies variables D^{in} and D^{out} ;
- R refers to connections between components and is equivalent to the σ function;
- the synchronization vector V is given by the allocation relation A: if a task $\boldsymbol{\psi}_i$ is allocated to a processor $\boldsymbol{\psi}_j$ then $(evt_i, evt_j) \in V$ for $evt \in \Sigma$.

5. Experimental Case Study

We choose as a case study a 3D image synthesis application. It is a complex application with some flexibilities related to the computation time of some tasks and their executing occurrence. The application class hierarchy is implemented in OWL Protégé editor shown in Figure 10. The input of the application graph is a set of the coordinates of the different polygons' summits that represent the 3D object. All coordinates are defined relative to a local space where the 3D object is located. Those coordinates are manipulated by different arithmetic functions (tasks) that create the animation, such as : Loading, Scaling, Adding, Rotating, Translating, etc. Tasks are implemented with their temporal features, and precedence relation according to the application graph. Rules are implemented by Semantic Web Rule Language (SWRL) interface (Figure 11) and combined with the scheduling ontology in order to improve the domain knowledge. Examples of some temporal rules are defined in equations (1) to (8), other rules related to allocation are like in (10) to indicate that processor can hold only one task to execute.

$$Tasks(?T) \wedge hasLi(?T, ?L) \wedge Processor(?P) \wedge hasProcessor(?T, ?P) \rightarrow sqwrl:select(?T) \wedge sqwrl:orderBy(?L) \wedge sqwrl:groupBy(?P) \quad (10)$$

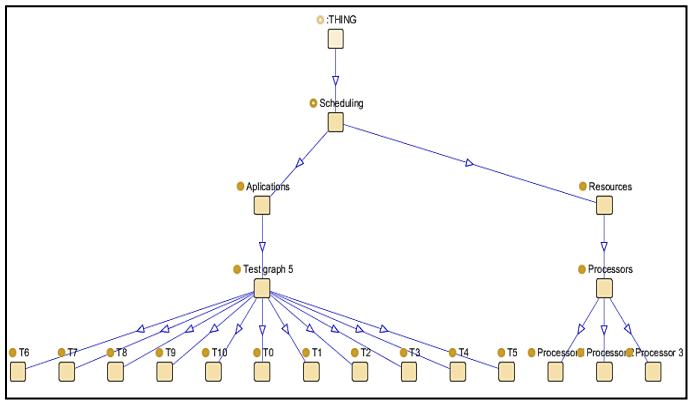

Figure 10: Class Hierarchy in Scheduling Ontology

After creating the SWRL rules, the reasoner component is able to infer the user query based on the ontology knowledge and predefined rules. To type queries, we tested two methods:

- The first method is simple and use existing information in the knowledge base without the need to any inferences. We use such method to verify existing properties in our ontology knowledge. For example, if we want to verify which processor is available for accepting a task execution, or the remaining computation time of some tasks.
- The second query method triggers the rule-based reasoner for the inference process with the knowledge which may lead to enhance and enrich the knowledge base. For example, if we consider tasks migration between processors or CLBs, and we want to decide migration of task T executing on processor1 when processor1 is overloaded. Hence, the rule-based reasoner has to infer the possibility of such migration based on the status of each processor and check that whether this change (so this query) is feasible or not.

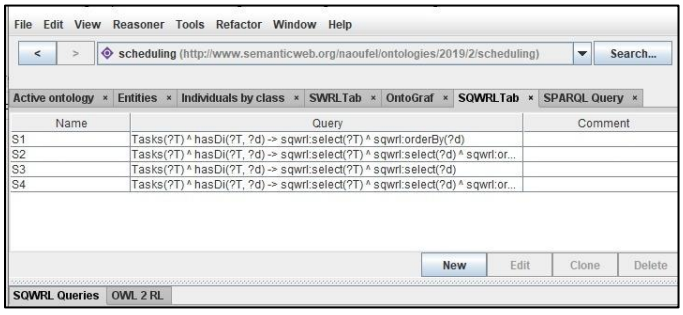

Figure 11: Example of SWRL Rules

Inference results are resulting from SPARQL queries. The SPARQL service permits to check different information as the status of tasks and it provides feedback concerning events and decision failures. The tested query results are correct, and the rule-based reasoner is useful for the scheduling problem. Based on that formalized temporal/resource rules combined with the semantic reasoner and inference rules, the ontology model gives the resulting possible tasks scheduling (Figure 12).

Some examples of successfully implemented scheduling rules (EDF algorithm (9), Least Laxity First LLF algorithm (11), Rate Monotonic algorithm RM (12):

$$Tasks(?T) \wedge hasLi(?T, ?L) \rightarrow sqwrl:select(?T) \wedge sqwrl:orderBy(?L) \quad (11)$$

$$Tasks(?T) \wedge hasCi(?T, ?C) \rightarrow sqwrl:select(?T) \wedge sqwrl:orderBy(?C) \quad (12)$$

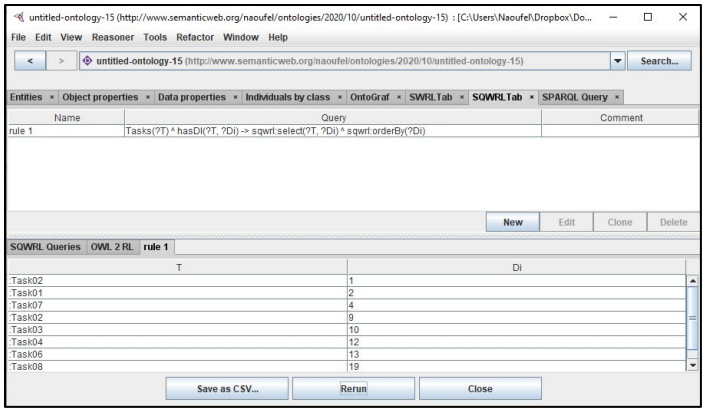

Figure 12: SQWRL Rules Result

By using AADL, nominal models were designed and combined also with failure models of the domain ontology. The system's adding process executes a simple add function, and it is defined in the Adding task. The operation is called performed when the required data context of the former task is provided to the data context of the latter task. To validate operation, scheduler check rules and constraint violation in the error model represented by the Overflow Event State, shown in Table1.

Table 1: Example of incompleteness transition Error

Implementation	Error model
error model implementation OVERFLOW_2.Notionala transitions Error_Free -[overflow]-> Over_Flow; Error_Free -[overflow]-> Ovf_Temp; end OVERFLOW_2.Notionala;	error model OVERFLOW_2 Features Error_Free: initial error state; Ovf_Temp: error state; Over_Flow: error state; overflow: error event; end OVERFLOW_2;

6. Conclusion

We faced in this article the problem of modeling the complexity of reconfigurable systems while taking advantage of its flexibility and dynamic behavior. For this, we proposed a model-driven engineering process that follows a set of steps. The proposed modeling process makes a modular and extensible representation of the system architecture. Model profits from the ontology language capability to represent complex models and address heterogeneous domains information of the dynamically reconfigurable systems. We formalized the known temporal

scheduling problem in reconfigurable systems domain. The ontology knowledge model was implemented using Protégé editor. Moreover, the developed ontology is improved with SWRL inferred rules. Model editing is enhanced, since errors can be detected without the need to perform complex analyses. The model and the rules reasoner and the resulting scheduling decision were verified with a case study.

Future works aim to extend model transformation functionality and involve more dynamic system characteristics such as tasks migrations between hardware and software resources and the total and partial reconfigurability. A filtering process should be added to the reasoner in order to eliminate unnecessary rules and so avoid ambiguity or conflict when making scheduling decision. Another objective is to test more scheduling algorithms with the incorporation of additional types of error models.

Acknowledgment

This work has been approved by the research grant program of the deanship of scientific research at TAIF University.

References

[1] M.A. Cardin, "Enabling flexibility in engineering systems: A taxonomy of procedures and a design framework," Journal of Mechanical Design, Transactions of the ASME, **136**(1), 2014, doi:10.1115/1.4025704.

[2] S. Paul, S. Bhunia, S. Paul, S. Bhunia, "A Survey of Computing Architectures, Springer New York": 11–27, 2014, doi:10.1007/978-1-4614-7798-3_2.

[3] S. Vassiliadis, D. Soudris, "Fine-and coarse-grain reconfigurable computing", Springer Netherlands, 2008, doi:10.1007/978-1-4020-6505-7.

[4] L. Gong, O. Diessel, "Functional verification of dynamically reconfigurable FPGA-based systems", Springer International Publishing, 2015, doi:10.1007/978-3-319-06838-1.

[5] R. Tessier, K. Pocek, A. DeHon, "Reconfigurable computing architectures," Proceedings of the IEEE, **103**(3), 332–354, 2015, doi:10.1109/JPROC.2014.2386883.

[6] G. Gracioli, A.A. Fröhlich, R. Pellizzoni, S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time OS," Real-Time Systems, **49**(6), 669–714, 2013, doi:10.1007/s11241-013-9183-3.

[7] M. Alam, A. Khan, A.K. Varshney, "A review of dynamic scheduling algorithms for homogeneous and heterogeneous systems," in Advances in Intelligent Systems and Computing, Springer Verlag: 73–83, 2018, doi:10.1007/978-981-10-8533-8_8.

[8] A. Rodrigues Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," Computer Languages, Systems and Structures, **43**, 139–155, 2015, doi:10.1016/j.cl.2015.06.001.

[9] L. Yang, K. Cormican, M. Yu, "Ontology-based systems engineering: A state-of-the-art review," Computers in Industry, **111**, 148–171, 2019, doi:10.1016/j.compind.2019.05.003.

[10] B. Selić, S. Gérard, "Model-Based Schedulability Analysis", Elsevier: 201–220, 2014, doi:10.1016/b978-0-12-416619-6.00010-9.

[11] V. Mascardi, A. Locoro, P. Rosso, "Automatic ontology matching via upper ontologies: A systematic evaluation," IEEE Transactions on Knowledge and Data Engineering, **22**(5), 609–623, 2010, doi:10.1109/TKDE.2009.154.

[12] J. Li, J. Tang, Y. Li, Q. Luo, "RiMOM: A dynamic multistrategy ontology alignment framework," IEEE Transactions on Knowledge and Data Engineering, **21**(8), 1218–1232, 2009, doi:10.1109/TKDE.2008.202.

[13] G. Chen, T. Jiang, M. Wang, X. Tang, W. Ji, "Modeling and reasoning of IoT architecture in semantic ontology dimension," Computer Communications, **153**, 580–594, 2020, doi:10.1016/j.comcom.2020.02.006.

[14] M.A. Musen, "The Protégé Project: A Look Back and a Look Forward". AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence, 1(4), 2015, doi:10.1145/2557001.25757003.

[15] D.L. McGuinness, F. van Harmelen, "OWL Web Ontology Language - Reference," W3C Recommendation [Online], Available at: Http://Www.W3.Org/TR/2004/REC-Owl, 1–53, [Accessed: 02 February 2016], 2004.

[16] P.H. Feiler, B.A. Lewis, S. Vestal, "The SAE architecture analysis & design language (AADL) a standard for engineering performance critical systems," in Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, CACSD, 1206–1211, 2007, doi:10.1109/CACSD.2006.285483.

[17] SAE: Architecture Analysis and Design Language (AADL) AS-5506B. Tech. rep., The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.1 (September 2012).

[18] Prosvirnova, T., Batteux, M., Brameret, P.A., Cherfi, A., Friedlhuber, T., Roussel, J.M., Rauzy, A.: "The altarica 3.0 project for model-based safety assessment". In: Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2013. IFAC, York (Great Britain) (September 2013).

[19] P. Bieber, J. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, E. Conquet, "Integration of Formal Fault Analysis in ASSERT: Case Studies and Lessons Learnt," 4th European Congress on Embedded Real-Time Software (ERTS), 1–9, 2008.

[20] P. Hönig, R. Lunde, F. Holzapfel, "Model Based Safety Analysis with smartIflow †," Information, **8**(1), 7, 2017, doi:10.3390/info8010007.

[21] D. Bošnački, A. Wijs, "Model checking: recent improvements and applications," International Journal on Software Tools for Technology Transfer, **20**(5), 493–497, 2018, doi:10.1007/s10009-018-0501-x.

[22] M. Faugère, T. Bourbeau, R. De Simone, S. Gérard, "MARTE: Also an UML profile for modeling AADL applications," in Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS, 359–364, 2007, doi:10.1109/ICECCS.2007.29.]

[23] N. Laaz, K. Wakil, Z. Gotti, S. Gotti, S. Mbarki, "Integrating domain ontologies in an MDA-based development process of e-health management systems at the CIM level," in Advances in Intelligent Systems and Computing, Springer: 213–223, 2020, doi:10.1007/978-3-030-36664-3_25.]

[24] N. Laaz, S. Mbarki, "OntoIFML: Automatic Generation of Annotated Web Pages from IFML and Ontologies using the MDA Approach: A Case Study of an EMR Management Application," in Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, Scitepress: 353–361, 2019, doi:10.5220/0007402203530361.]

[25] S. Roser, B. Bauer, "Automatic generation and evolution of model transformations using ontology engineering space," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, Berlin, Heidelberg: 32–64, 2008, doi:10.1007/978-3-540-92148-6_2.

[26] G. Guizzardi, R.A. Falbo, R.S.S. Guizzardi, "The role of Foundational Ontologies for Domain Ontology Engineering: a case study in the Software Process Domain," IEEE Latin America Transactions, **6**(3), 244–251, 2008, doi:10.1109/tla.2008.4653854.

[27] N.F. Noy, M.A. Musen, "Ontology versioning in an ontology management framework," IEEE Intelligent Systems, **19**(4), 6–13, 2004, doi:10.1109/MIS.2004.33.

[28] W. Alakwaa, A. Salah, "Model Transformation from Ontology Model to Content Analysis Model," International Journal of Computer Applications, **7**(3), 5–12, 2010, doi:10.5120/1147-1501.

[29] P. Wang, Z. Jin, L. Liu, G. Cai, "Building toward capability specifications of web services based on an environment ontology," IEEE Transactions on Knowledge and Data Engineering, **20**(4), 547–561, 2008, doi:10.1109/TKDE.2007.190719.

[30] M.R. Khondoker, M.R. Khondoker, P. Mueller, "Comparing Ontology Development Tools Based on an Online Survey," 2010.

[31] T.C. Eskridge, R. Hoffman, "Ontology creation as a sensemaking activity," IEEE Intelligent Systems, **27**(5), 58–65, 2012, doi:10.1109/MIS.2012.101.

[32] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, "On-line scheduling in the presence of overload," in Annual Symposium on Foundations of Computer Science (Proceedings), Publ by IEEE: 100–110, 1991, doi:10.1109/sfcs.1991.185354.

[33] G.C. Buttazzo, "Hard Real-Time Computing Systems", Springer US, 2011, doi:10.1007/978-1-4614-0676-1.

[34] Y. Zhao, D. Ma, "Embedded real-time system modeling and analysis using AADL," in ICNIT 2010 - 2010 International Conference on Networking and Information Technology, 247–251, 2010, doi:10.1109/ICNIT.2010.5508520.

[35] T.C. Jepsen, "Just what Is an ontology, anyway?," IT Professional, **11**(5), 22–27, 2009, doi:10.1109/MITP.2009.105.

[36] Y. Ma, L. Liu, K. Lu, B. Jin, X. Liu, "A graph derivation based approach for measuring and comparing structural semantics of ontologies," IEEE Transactions on Knowledge and Data Engineering, **26**(5), 1039–1052, 2014, doi:10.1109/TKDE.2013.120.

[37] K. Jetinai, N. Arch-int, S. Arch-int, "Ontology Mapping and Rule-Based Inference for Learning Resource Integration," Journal of Information and Communication Convergence Engineering, **14**(2), 97–105, 2016, doi:10.6109/jicce.2016.14.2.097.