Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain

Alessio Bucaioni*†, Antonio Cicchetti*, Federico Ciccozzi*, Saad Mubeen*,
Alfonso Pierantonio‡*, Mikael Sjödin*
* MRTC, Mälardalen University, Västerås, Sweden
Email: firstname.lastname@mdh.se
† Arcticus Systems, Järfälla, Sweden
‡ DISIM, Universitá degli Studi dell'Aquila, L'Aquila, Italy
Email: alfonso.pierantonio@univaq.it

***Abstract*—Models and model transformations, the two core constituents of Model-Driven Engineering, aid in software development by automating, thus taming, error-proneness of tedious engineering activities. In most cases, the result of these automated activities is an overwhelming amount of information. This is the case of one-to-many model transformations that, e.g. in design-space exploration, can potentially generate a massive amount of candidate models (i.e., solution space) from one single model. In our scenario, from one design model we generate a set of possible implementation models on which timing analysis is run. The aim is to find the best model from a timing perspective. However, multiple implementation models can have equally good analysis results. Therefore, the engineer is expected to investigate the solution space for making a final decision, using criteria which fall outside the analysis' criteria themselves. Since candidate models can be many and very similar to each other, manually finding differences and commonalities is an impractical and error-prone task. In order to provide the engineer with an expressive representation of models' commonalities and differences, we propose the use of modelling with uncertainty. We achieve this by elevating the solution space to a first-class status, adopting a compact notation capable of representing the solution space by means of a single model with uncertainty. Commonalities and differences are thus represented by means of uncertainty points for the engineer to easily grasp them and consistently make her decision without manually inspecting each model individually.**

I. Introduction

Model-Driven Engineering (MDE) [1] is rapidly evolving in academia and have gained considerable foothold in industrial software-development projects. While model transformations can relieve software developers from significant engineering effort and mitigate errors typical of manual translations, they can also potentially create an overwhelming amount of information. Especially, design-space exploration techniques, characterised by one-to-many model transformations, have the potential to generate hundreds, thousands, or more, candidate solutions (i.e., models) from one single model. Despite automated analyses can be employed for evaluating the appropriateness of each candidate solution – as done in our previous work [2] – their usefulness for the engineer can be limited as the solution space is never really unveiled in the process. In fact, while the analysis refines the solution space by sorting out solutions not complying to given requirements, the engineer still has multiple choices and remains *uncertain* about the one to take: a decision can only be made by manually inspecting and comparing all candidate models. However, since candidate models can be very similar to each other, a manual traversing of the solution space is impractical and error-prone. This is worsened by the fact that the number of alternatives, as well as their size, may grow exponentially for several reasons, e.g., more complex source models.

In this paper, we propose the use of modelling with uncertainty in order to explicitly represent the uncertainty that typically accompanies many stages of the development process [3]. More specifically, we revise our methodology [2] in order to accommodate a compact notation capable of representing the solution space by means of a single model (with uncertainty). The intent is to provide the engineer with an expressive representation of all candidate models with their commonalities and distinctions by means of *uncertainty points*. The engineer can therefore easily grasp the differences among candidate models and consistently make her decision without manually inspecting each model individually. Such a support is provided by employing the metamodel-independent technique presented in [4]. Moreover, an industrial application from the automotive domain is used to illustrate the advantages of the proposal.

Outline. The remainder of the paper is organised as follows. Section II illustrates the context of this work, while the subsequent section describes a motivating examples taken from the automotive domain. Section IV introduces the uRubus metamodel, i.e., execution models with uncertainty. Section V discusses the pros and cons of modelling with uncertainty. Section VI presents related work documented in literature while Sect. VII draws conclusions and future work.

II. Background

In the automotive domain, the adoption of models and MDE led to the standardisation of an architectural description language, called EAST-ADL [5]. EAST-ADL proposes a top-down development approach relying on four abstraction levels – vehicle, analysis, design and implementation – which implicitly ensure separation of concerns through the engineering phases. Each abstraction level is described by means of metamodelling constructs and hides unnecessary information from lower abstraction levels. EAST-ADL has been developed with particular focus on the functional and structural modelling. However, it does not focus on execution and timing

modelling [6][1]. To this end, EAST-ADL is usually complemented, at implementation level, with additional notations that explicitly support execution and timing modelling. Among other alternatives, Rubus Component Model (RCM) [7] is a modelling language which gained industrial recognition as an EAST-ADL complementary technology. RCM was developed by Arcticus Systems[2] in collaboration with Mälardalen University and it is currently used by several international companies, e.g., Volvo CE[3], BAE Systems[4], for execution and timing modelling of distributed resource-constrained real-time software systems. EAST-ADL provides means for abstraction and separation of concerns, but it does not provide explicit support for automation among the different abstraction levels.

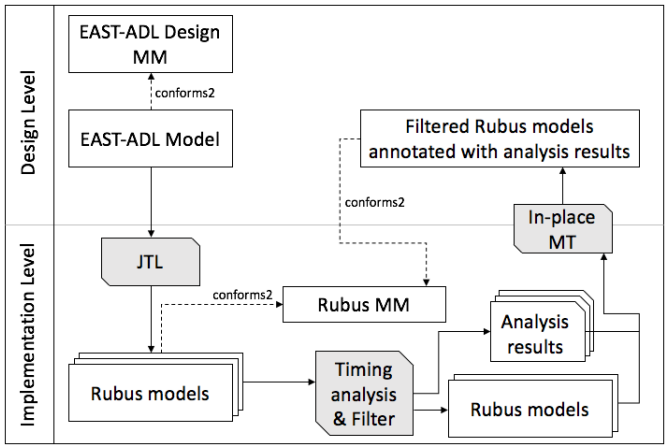

Fig. 1: Original methodology supporting timing analysis

Therefore, in our previous work [2], we have described a methodology for seamlessly linking the modelling language used at EAST-ADL design level and RCM with the aim of enabling high-precision timing analysis[5] at EAST-ADL design level. The methodology before this contribution, depicted in Fig. 1, leveraged model-driven techniques as follows. Starting from an EAST-ADL design model, the methodology generated the set of all the corresponding meaningful Rubus models, which represented our candidate solutions. The generation was performed by means of a model transformation defined in JTL [8] that non-deterministically generated all the models satisfying the constraints encoded in the transformation itself. At this point, timing analysis was run on the generated Rubus models resulting in a set of analysis results. These results were checked against a non-empty set of timing requirements expressed on the vehicle functionality. The result which better met the given timing requirements was selected and the corresponding Rubus model was conveyed back[6] to the engineer. It is worth noting that, when selecting among analysis results, multiple results and thereby Rubus models could be selected if timing requirements were met. However, when this happened, the engineer was required to manually inspect the set of *filtered* Rubus models annotated with analysis results individually. From a broader perspective, this operation is frequent in human-in-the-loop processes where domain knowledge is needed to meet decisions that cannot be made by the tools. Therefore, providing (semi-) automated support that prevents the engineer from manually traversing the solution space is key to success.

To this end, we realised that there was need for our methodology to entail a compact notation to represent the solution space (e.g., Rubus models) by means of a model with uncertainty. In such a representation, model differences are enucleated in uncertainty points that provide the engineer with a straightforward locality for understanding how models differ one with another.

III. Motivating Scenario

Let us apply the methodology introduced in Sect. II on the automotive application called Intelligent Parking Assist (IPA) system. The IPA system assists drivers in parking their vehicles. To this end, it uses a warning system, composed of a set of proximity sensors and backup cameras, for detecting obstacles and calculating optimum manoeuvres.

For the sake of simplicity, we consider only a portion of the software architecture consisting of two nodes, namely *IPAssistant* and *Actuator* connected to a single network that implements the Controller Area Network (CAN) protocol [9] (Fig. 2). Figure 2 depicts the EAST-ADL design level model of the partial software architecture[7]. In the hierarchy of an EAST-ADL design model, the so-called design function prototype (DFP) represent a specific instance of a vehicle functionality[8]. The partial IPA architecture consists of seven DFPs in a chain. *Proximity_Sensor_DFP*, *Input_Process_DFP*, *Path_Calculator_DFP* and *CAN_Send_DFP* DFPs are part of the software architecture of the IPAssistant node. The remaining three DFPs in the chain, *CAN_Receive_DFP*, *Control_DFP* and *Brake_Actuator_DFP* are part of the software architecture of the Actuator node. Please note that CAN_Send_SWC sends a network message that is received by CAN_Receive_SWC. There is a periodic constraint of 10 ms that is specified on each DFPs in the chain. However, the information about whether each DFPs is activated independently or by its predecessor is not available. The following timing requirement is specified too:

- *"The calculated age and reaction delays [11] shall not exceed 20 ms and 15 ms, respectively."*

Within EAST-ADL, timing requirements are specified by timing constraints [12]. Therefore, there are two timing constraints, namely Data Age (AgeChain2) and Data Reaction (DRChain2), that are specified from the input flow port of Input_Process_SWC to the output flow port of Control_SWC as shown in Fig. 2.

[1]Lately, EAST-ADL has been extended for supporting the modelling of timing requirements [5].

[2]https://www.arcticus-systems.com

[3]http://www.volvoce.com/dealers/sv-se/swecon/Pages/homepage.aspx

[4]http://www.baesystems.com

[5]In the remainder of the paper, *high-precision timing analysis* is referred simply as *timing analysis*

[6]Back-propagation was achieved through in-place model transformations that annotated filtered Rubus models with related analysis results.

[7]We have modelled the IPA system with the help of Rubus ICE [10]

[8]EAST-ADL implements the type-prototype pattern. Therefore, a DFP represents a specific instance of a design function type, which defines its type. the complete explanation of the EAST-ADL metamodel is not in the scope of this work. The interested reader is referred to [5] for details

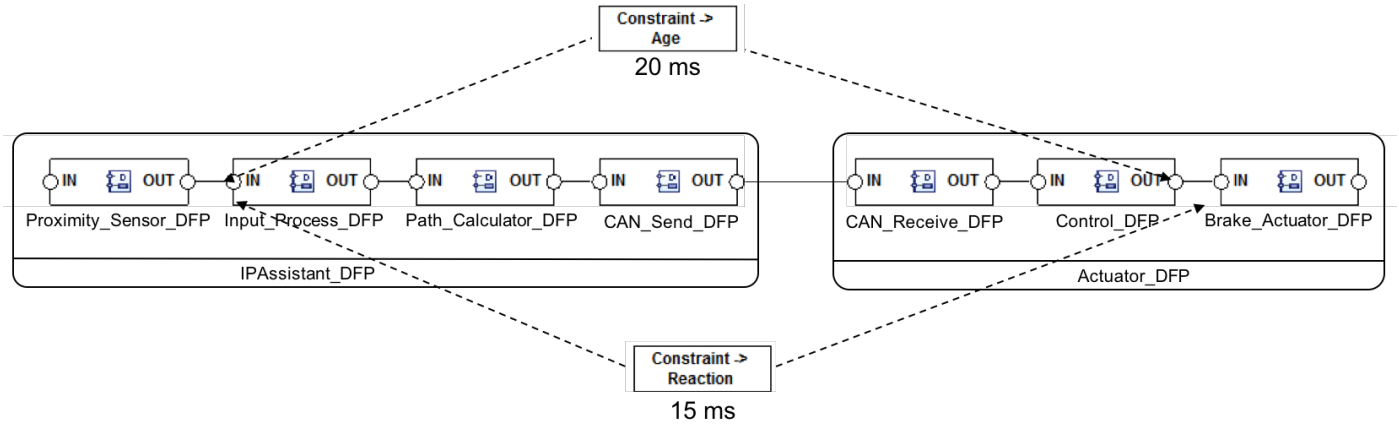

Fig. 2: Partial software architecture of the two nodes in IPA system at the design-level of EAST-ADL.

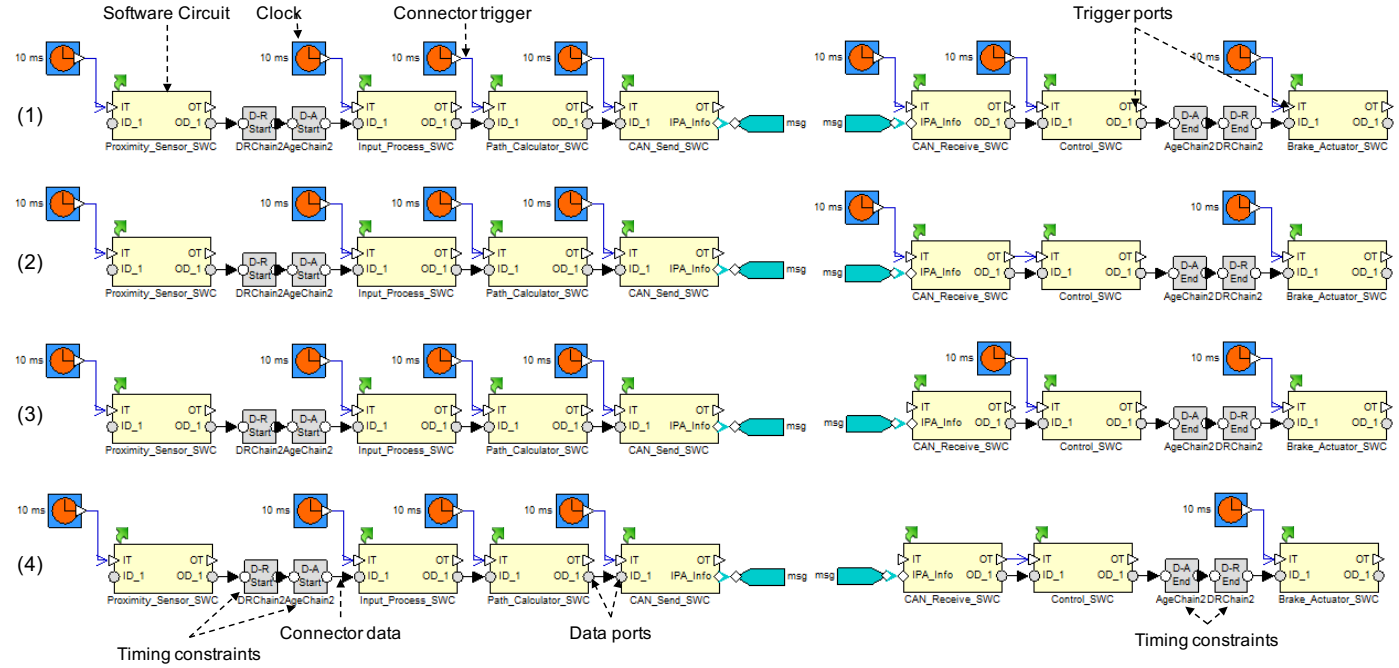

Fig. 3: 4 of the 32 Rubus models generated from the EAST-ADL model

So far according to our original methodology (Fig. 1), the EAST-ADL model in Fig. 2 is transformed in 32 Rubus models. The first 4 Rubus models[9] are depicted in Fig. 3. In the hierarchy of a Rubus model, a software circuit (SWC) encapsulates basic software functions. RCM distinguishes between data and control flow therefore a SWC has data and trigger ports. Within RCM, data connectors link data ports while trigger connectors link trigger ports. Clocks and trigger sinks are used to initiate and terminate the execution of a SWC, respectively. A simplified version of the Rubus metamodel is presented in Sect. IV. All these models differ from each other depending upon whether a SWC is activated independently by a clock element or by its preceding SWC. Considering the age constraint of 20 ms specified in Fig. 2, only 14 out of 32 Rubus models satisfy it whereas only 1 Rubus model satisfies the specified reaction constraint of 15 ms[9]. Despite the automated analysis has filtered the solution space, there are still 14 Rubus models which must be inspected by the engineer for deciding which one should be selected for proceeding in the development process. However, with the current support, such an inspection might be a daunting task as the selected Rubus models greatly overlap one with another. For instance, let us consider the Rubus models marked with *(1)* and *(2)* in Fig.3. The only difference between these two models is on how the Control_SWC SWC is activated: in the model marked with (1) it is activated from a clock element, while in the model marked with (2) is activated from its preceding SWC. If these small differences are hard to catch when dealing with a reasonably small number of models and model elements, they are nearly

[9]The interested reader can find the whole set of artefacts at http://www.mrtc.mdh.se/SEAA2016.

impossible to spot when dealing with hundreds or thousands models and model elements.

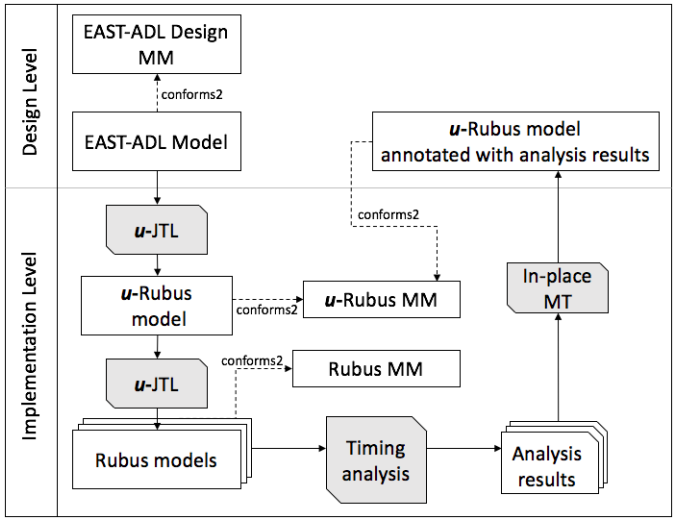

Fig. 4: New methodology supporting timing analysis and uncertainty

Contribution. In order to ease the inspection of the solution space represented by Rubus models, we enhanced our methodology (Fig. 4) by introducing the *u*-Rubus metamodel, a compact notation to represent the solution space by means of a single *u*-Rubus model (that can represent uncertainty). Uncertainty points are employed for representing commonalities and distinctions of the Rubus models. The *u*-Rubus metamodel was generated by means of an automated transformation defined in the revised version of JTL, which we hereafter call *u*-JTL [4]. Since our timing analysis is currently not able to run on *u*-Rubus models, we exploit a concretiser operator (in the sense of [3]) provided by *u*-JTL that, starting from a *u*-Rubus model, returns all Rubus models encoded in it and on which timing analysis can be run. Analysis results are then back-propagated as annotations to the *u*-Rubus model through an in-place model transformation.

IV. *u*-RUBUS

In this section, we present the *u*-Rubus metamodel. Such a modeling notation is obtained by endowing Rubus with uncertainty elements in order to deal with the multitude of Rubus models presented above. The intent is to provide the engineer with a representation that permits to deal with a set of Rubus models as if they were a single model and do reasoning with all the possible models at the same time.

With reference to the small Rubus fragment[9] in Fig. 5, an execution model consists of `Circuit`(s), that have exactly one `Interface` with `Connector`(s). In turn, connectors can be either `ConnectData` or `ConnectorTrig` to denote data- and control-flow linking a circuit to another. In Fig. 3 part of the Rubus models generated with a JTL program with the original methodology are shown. In many cases it has been observed that generated models share most of their model elements, making engineer's life harder as comprehending the differences among the models is not always straightforward.

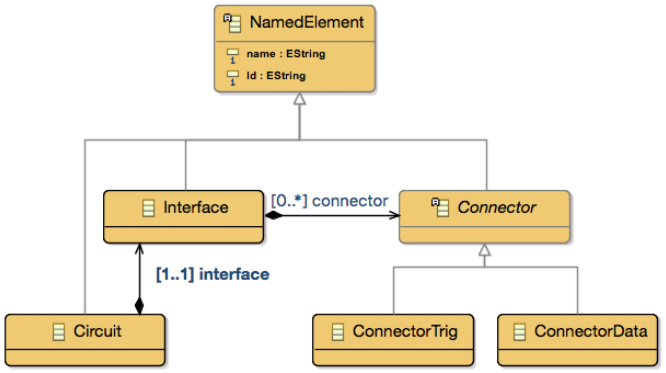

Fig. 5: A Rubus metamodel fragment

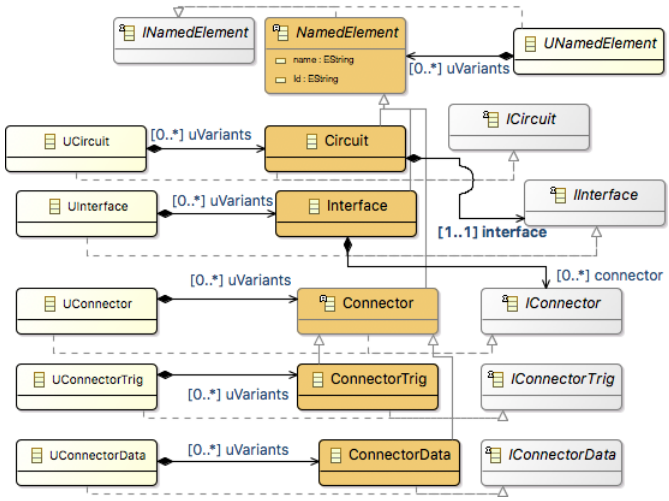

Fig. 6: A *u*-Rubus Metamodel fragment

Recently, the JTL language has been given an *intensional* semantics in order to generate models with uncertainty [4]. Transformations, instead of delivering myriads of models, can generate models with uncertainty, i.e., models denoting *multiple possibilities*. As a result, engineers do not need to manually compare models to discern between them anymore, but rather they can combine the variants associated to uncertainty points to explore the solution space. In order to consistently represent *uncertain* elements, i.e., elements that are optional or mutually exclusive, the Rubus metamodel has to be extended with additional constructs. This is performed by an automated transformation (see [13]) that, starting from uRubus, generates the *u*-Rubus metamodel shown in Fig. 6 as follows:

i) any *class* in Rubus is added to *u*-Rubus; in addition

ii) auxiliary classes *Uclass* and *Iclass*, with *class* and *Uclass* subclasses of *Iclass*, are added to *u*-Rubus;

iii) association $uVariants : Uclass \diamond\!\!\longrightarrow class$ is added to *u*-Rubus;

iv) for each association $a : class_1 \diamond\!\!\longrightarrow class_2$ in Rubus, an association $a : class_1 \diamond\!\!\longrightarrow Iclass_2$ is added to *u*-Rubus.

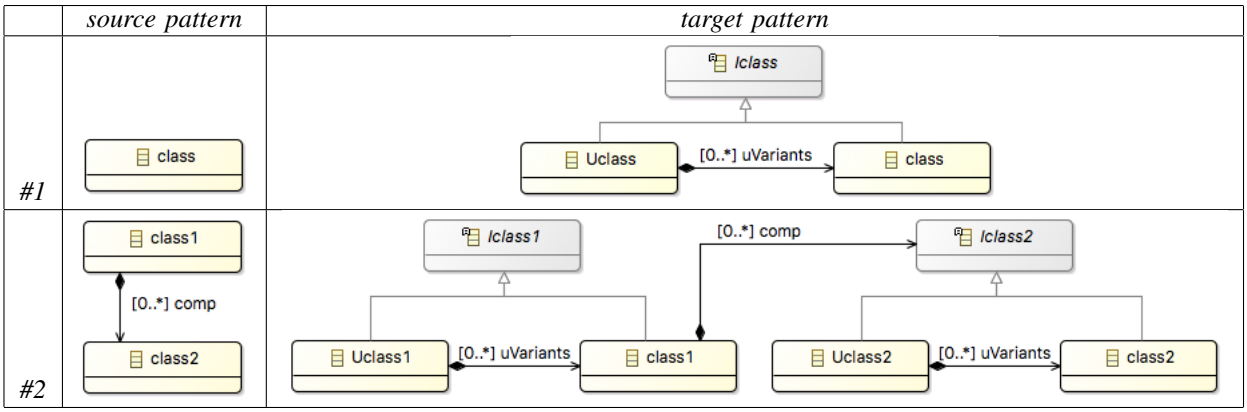

Fig. 7: Rubus to *u*-Rubus mappings

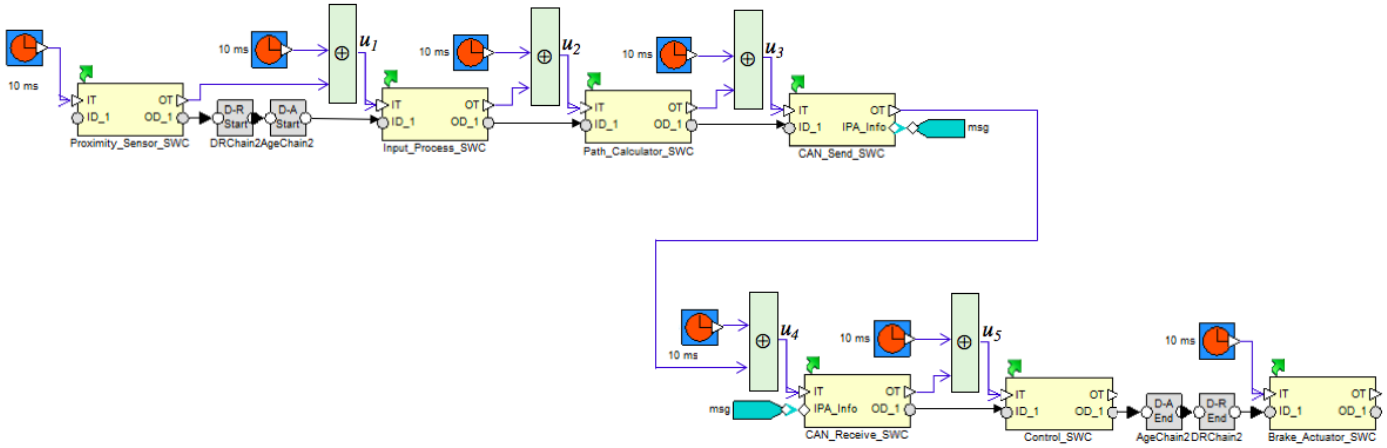

Fig. 8: The *u*-Rubus model generated by *u*-JTL representing the solution space

In particular, the procedure can also be illustrated in term of pattern rewriting rules as illustrated in Fig. 7. The first three steps (*i-iii*) realize the mapping from the source to the target pattern in the first row, while the last step (*iv*) is represented in the second row. For instance, when the first mapping is applied to `Circuit` in Rubus, then it is propagated to *u*-Rubus together with the newly created `UCircuit` and `ICircuit` metaclasses as shown in Fig. 6. The `UCircuit` metaclass represents uncertainty points where to anchor multiple alternative `Circuit` instances. Whereas, the role of the `IInterface` is to let the propagated *interface* composition in *u*-Rubus to refer to either a single *Interface* instance or to multiple instances through the *UInterface* (as subclass of *IInterface*).

As aforementioned, the new methodology makes use of *u*-Rubus models for representing the solution space as for instance illustrated in Fig. 8. In particular, the green elements $u_1, \ldots, u_5$ are `UConnectorTrig` uncertainty points representing two (mutually excluded) connectors each: a timed one triggered by a clock element, say u_i', and another directly triggered by the preceding circuit, say u_i''. Such a model represents an overall number of 2^5 Rubus models[10]. Currently the timing analysis can only be performed on sets of individual Rubus models, therefore the multivalued *concretisation* operator (see [14]), part of the *u*-JTL environment, and defined as

$$concr : \text{uRubus} \rightarrow \mathcal{P}(\text{Rubus}),$$

returns the set of all Rubus models encoded in the corresponding *u*-Rubus model. It is worth noting that the original JTL transformation, in charge of generating a set of Rubus models from an EAST-ADL model, did not have to be modified to generate *u*-Rubus models. This is due to the fact that the new *u*-JTL transformation engine is semantically equivalent to the one of JTL, although the way models are represented is different. Once the Rubus models are obtained by concretising the *u*-Rubus model, we can perform timing analysis. Without being too specific, the outcome of such an analysis is a subset of Rubus models satisfying given timing requirements.

Each Rubus model obtained by means of the *concr* operator is univocally identified by the variants chosen for each uncertainty point u_i. For instance, in the case shown in Fig. 8 the Rubus model with only clock elements is given by the 5-tuple $< u_1', \ldots, u_5' >$. Therefore, the tuples identifying all the models, which satisfied the timing analysis, are translated back into annotations in the *u*-Rubus model together with analysis results.

[10]The overall number comes from the number of possible variants (2) to the power of the number of the connector uncertainty points u_i (5).

Besides being able to better locate the differences among the models, the main advantage of this proposal consists in harnessing the possibility to reason with all the models as a whole. In fact, the engineer can search through the models, which passed the timing analysis, by discarding those with characteristics non conforming to criteria that fall outside the analysis itself. For instance, in the context of the automotive application presented in Sect. III, timing variations are of crucial relevance that the engineer cannot neglect. At times, it might be very important to prefer models presenting more independent clocks because they can better accommodate the branching and merging of data along the chain. Also, independent clocks suit better to SWCs having more than one data input port. On the other hand, it might be desirable in some other models to have dependent activation of SWCs receiving messages from the network, e.g., CAN_Receive_SWC in Fig. 8 as they ensure that fresh data from the network traverses through the rest of the model.

V. Discussion

In this paper, we have proposed a compact notation for representing a solution space by means of a model with uncertainty. We have described how the proposed notation can ease the esploration of the solution space, especially when candidate solutions display minimal variations among themselves. This contribution enhances our methodology for seamlessly linking EAST-ADL and RCM. More specifically, the notation, addressed as Rubus with uncertainty (*u*-Rubus), allows the developer to inspect the solution space represented by the set of Rubus models automatically generated from a single EAST-ADL design level model as valid implementation alternatives. In the scenario presented here, timing analysis is run on the initial set of generated Rubus models. Thus, the set of valid alternatives, from a timing perspective, is selected. These alternatives can be very similar to each other, and it may be hard for the engineer to effectively compare them and comprehend how they differ one with another. This difficulty is exacerbated by the number of alternatives (as well as the their modelling elements) that may grow exponentially due to, e.g., loose timing requirements. *u*-Rubus spawns the means for the engineer to grasp at a glance the solution space of interest through a compact visualisation of uncertainty points represented in a single model. The exploration of the solution space remains manual. In fact, this contribution represents a first step towards an analysis-based and semi-automatic design-space exploration mechanism for guiding the engineer towards selecting the most suitable Rubus model among a possibly huge set of alternatives. We have already started investigating the possibility to extend our methodology (Fig. 9) for running timing analysis on a *u*-Rubus model instead of the set of Rubus models. Doing so, we would be able to entirely act on *u*-Rubus model, from start to end, with no need for concretising/de-concretising mechanisms from/to a *u*-Rubus model. Moreover, with a *u*-Rubus model as sole artefact, we could be able to provide an analysis mechanism that, while analysing the candidate solutions (in terms of the *u*-Rubus model) also gives the possibility to the engineer to interactively decide upon uncertainty points (when and whether she wishes so) based on partial analysis results.

While we showed how *u*-Rubus can be exploited for investigating the solution space of Rubus models representing

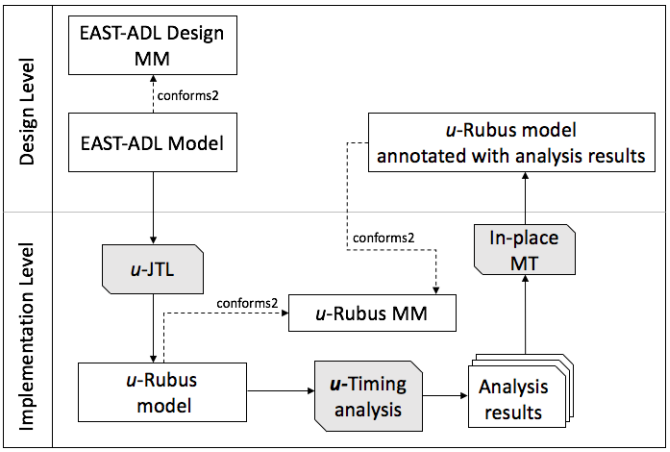

Fig. 9: Future methodology supporting timing analysis for uncertainty

timing, this does not necessarily means that a decision on a single alternative must always be taken. In fact, the idea of a compact notation can be exploited for successively exploring the solution space of models in relation to various properties. Let us imagine that timing and power consumption are two properties of interest. The engineer could start with running timing analysis for reducing the solution space. *u*-Rubus for timing (shown in this paper) would then be exploited for selecting among equally good alternatives (from a timing perspective). It can happen that, even with the help of *u*-Rubus, the engineer is not able or does not want to solve all the uncertainty points. At this point, unsolved uncertainty points, representing a set of Rubus models, would be analysed to measure expected power consumption. The results of this analysis will provide the engineer with additional information for her to decide in two ways: *i)* by decorating the model that instantiates *u*-Rubus for timing with power-related details (if timing-related details are still needed) or *ii)* by creating a specifically generated *u*-Rubus for power consumption to switch the focus of investigation and selection to power only (in case decisions on timing are considered established). To summarise, timing analysis can represent one step in a potential exploration chain [15], where solution spaces are successively investigated based on different properties prioritisation.

VI. Related Work

The problem of generating, analysing, and optimising multiple design alternatives has been largely investigated and is usually referred to as design-space exploration (DSE) [16]. Rule-based DSE [17], [18] can be considered as a possible implementation of the exploration in a MDE context: the space of available solutions is expressed in terms of a model and transformations are used to derive the corresponding alternatives. Depending on the characteristics of those models and transformations, in [15] the authors introduced a catalog of exploration patterns: our approach complies to the model generation pattern, that is it performs an exhaustive derivation of implementation models (lower level of abstraction), enriched with timing details, as derivable from the system architecture designed through EAST-ADL, and constrained by

domain-specific rules. The generation is not meant to provide optimisation hints/solutions at architectural level [19]; rather, it implements a quality-driven model transformation [17], [20] to select all the suitable (timing configuration) results given a certain system architecture as input.

In general, the goal of DSE mechanisms is reaching an optimal solution in terms of certain properties of interest, therefore the available works usually focus on generating appropriate candidates in an effective way. Moreover, based on user's choice and/or heuristics, potential solutions are pruned and the exploration is driven towards optimal alternatives. Abdeen et al. [18] propose to combine genetic algorithms with rule-based DSE in order to achieve a domain-independent multi-objective optimisation process. The approach is fully automated, hence aiming at reaching the optimal solution without user intervention. Instead, in [21] the authors embed search-based mechanisms in a model transformation language to generate optimal models as solutions to design problems. Generated models are evaluated through specific metrics, typically encoded in the transformation as rules and constraints. User's input is mentioned as a solution evaluation possibility, however no further clarification is given with respect to dealing with the presentation of the alternatives.

In Schätz et al. [17] rule-based DSE for the development of embedded systems is supported by means of a declarative generation approach. A model transformation mechanism hosted in Prolog is exploited to both define exploration rules and constrain the set of suitable alternatives. Differently to our approach, the user has to mentally render the set of available choices and write corresponding predicates to narrow down the solution space, which in our opinion can be a more complex task than visually comparing the generated alternatives.

The DESERT tool [22] provides support for DSE based on constraints, where the exploration and pruning rules have to be manually defined by the user. Similarly to our proposal, DESERT offers a more compact representation of available design alternatives in terms of ordered binary decision diagrams. However, this approach translates to an element-by-element choice which is devoted to the selection of a sigle preferred solution among the generated ones. Instead, our models with uncertainty allow to examine a solution as a whole and to keep multiple design alternatives until the necessary maturity was achieved to take a more constrained decision. In this respect, Kang et al. [23] advocate the need of cost-effective DSE by avoiding the exploration of design aspects irrelevant for a certain phase of the design process. In fact, depending on the maturity of the design, some alternatives might look equivalent to the user whom is not yet concerned with some of the details about the system that are changed. The authors introduce also a tool, FORMULA, supporting a user-defined equivalence specification among solutions that guarantees the sole generation of non-isomorphic alternatives with respect to the existing equivalence relationships. The uncertainty representation introduced in this work supports FORMULA's vision in the sense that we permit to keep some choices open until a definitive decision can be taken. Moreover, the definition of uncertainty itself can be exploited as a definition of aspects to be explored, and uncertainty resolution techniques [24] can be used as alternative generation mechanism.

There exists a number of additional approaches, such as [25], [26], [27] to mention a few, which propose generic representations of the solution space tailored to DSE from different perspectives, i.e. targeting multiple optimisation aspects. Notably, Saxena and Karsai [25] introduce a generic DSE framework based on the extension of a domain-specific language for exploration purposes. Such extension is then translated to an appropriate intermediate format that can be exploited by multiple constraint solvers to compute disparate optimisations. The resulting solutions are listed and can be visualised in the tool, however a one-by-one browsing of the alternatives might be difficult to handle for the user, especially when their number grows and the difference between them was minimal. A similar approach is adopted in Octopus [27], a tool that supports DSE for software intensive embedded systems. A domain-specific model is translated towards a DSE tailored intermediate representation, which is exploited to perform several exploration tasks as analyses, searches, and diagnostics. The underlying goal is to implement an iterative development and refinement of the application model until the desired set of properties is completely satisfied. The intermediate representation can be also exploited to perform optimisations through parametrisation of selected properties, however the management of the potential uncertainty raised by the optimisation is not discussed in the work.

GASPARD [26] is a framework for the development of massively parallel embedded systems, and shares several solution mechanisms with what is described in our contribution. In particular, GASPARD provides a higher abstraction level modelling support based on UML and the MARTE profile; starting from such design level, the framework prescribes a workflow made-up of subsequent analyses and refinement steps, from higher to lower abstraction levels. Similarly to the automotive development process described in this paper, some analyses and refinements can be performed at the (EAST-ADL) design level, while others require lower abstraction details (notably timing). Moreover, the transition from higher to lower abstraction levels naturally raises the issue of managing multiple lower level alternatives for the same higher level model. Indeed, also in [26] the authors advocate for a refinement process able to support the growing number of alternatives that should be reduced step-by-step by an analysis method and the corresponding pruning of inadequate solutions. However, the authors do not provide any detailed discussion about the management of multiple alternatives at each step, and the refinement process seems to rely on the selection of a single candidate for each level of abstraction. In such a context, exploiting models with uncertainty would disclose the opportunity of keeping equally good alternatives for the next (lower) abstraction level, until an analysis would definitively discard a certain solution.

VII. Conclusion and Future Work

As software systems increase in size, complexity and heterogeneity there is a growing consensus on the need to leverage existing techniques, methods, and tools across abstraction. In the context of automotive software, model-based techniques underpin analyses that typically refine solution spaces, which consists of hundred, thousand, or more candidate solutions. Nevertheless, often the engineer might want to comparatively inspect the models in order to consider additional requirements that fall outside those taken into account by the analyses.

In this paper, we enhance our previous methodology by introducing the *u*-Rubus metamodel: a compact notation for formalizing the whole solution space in terms of models with uncertainty. The advantages of the proposal consists in letting the engineer *i)* to reason about multitudes of models as a whole; and *ii)* to better locate the differences among the models for identify models fulfilling criteria dictated by the engineer's domain expertise. This work represents a first attempt in leveraging abstraction and automation in design space exploration. Future work will investigate how timing analysis for individual Rubus models can be lifted to *u*-Rubus models in order to have better analysis performance and provide the engineer with more immediate feedback.

ACKNOWLEDGMENTS

This work is supported by the Swedish Knowledge Foundation (KKS) through the SMARTCore project, by the Swedish Research Council (VR) through the SynthSoft project, and by the Swedish Foundation for Strategic Research (SSF) through the PRESS project. We thank our industrial partners Arcticus Systems AB and Volvo CE, Sweden. Moreover, the authors are grateful to Gianni Rosa for his comments and insights during technical discussions.

REFERENCES

[1] D C Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

[2] Alessio Bucaioni, Antonio Cicchetti, Federico Ciccozzi, Romina Eramo, Saad Mubeen, and Mikael Sjödin. Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.

[3] Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: Towards modeling and reasoning with uncertainty. *ICSE*, pages 573–583, 2012.

[4] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 49–58. ACM, 2015.

[5] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[6] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture, http://dx.doi.org/10.1016/j.sysarc.2013.10.008*, Oct. 2013.

[7] Kaj Hänninen, Jukka Mäki-Turja, Mikael Sjödin, Mats Lindberg, John Lundbäck, and Kurt-Lennart Lundbäck. The rubus component model for resource constrained real-time systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[8] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Jtl: A bidirectional and change propagating transformation language. In *Software Language Engineering*, volume 6563, pages 183–202. 2011.

[9] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.

[10] Rubus-ICE: Integrated component Development Environment, 2013. http://www.arcticus-systems.com.

[11] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Proceedings of the IEEE Real-Time System Symposium ? Workshop on Compositional Theory and Technology for Real-Time Embedded Systems,*, 2008.

[12] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

[13] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. Uncertainty in bidirectional transformations. In *6th International Workshop on Modeling in Software Engineering, MiSE 2014 - Proceedings*, pages 37–42, New York, New York, USA, January 2014. University of L'Aquila, L'Aquila, Italy, ACM Press.

[14] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 224–239. University of Toronto, Toronto, Canada, April 2012.

[15] Ken Vanherpen, Joachim Denil, Paul De Meulenaere, and Hans Vangheluwe. Design-space exploration in model driven engineering. Technical report, SOCS-TR-2014.4, McGill University, 2014.

[16] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, December 2004.

[17] B. Schätz, F. Hölzl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*, pages 173–182, March 2010.

[18] Hani Abdeen, Dániel Varró, Houari Sahraoui, András Szabolcs Nagy, Csaba Debreceni, Ábel Hegedüs, and Ákos Horváth. Multi-objective optimization in rule-based design space exploration. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 289–300, New York, NY, USA, 2014. ACM.

[19] Martin Walker, Mark-Oliver Reiser, Sara Tucci-Piergiovanni, Yiannis Papadopoulos, Henrik Lnn, Chokri Mraidha, David Parker, DeJiu Chen, and David Servat. Automatic optimisation of system architectures using east-adl. *Journal of Systems and Software*, 86(10):2467–2487, 2013.

[20] Mauro Luigi Drago, Carlo Ghezzi, and Raffaela Mirandola. Towards quality driven exploration of model transformation spaces. In *Procs. of the 14th Int. Conf. on Model Driven Engineering Languages and Systems*, MODELS'11, pages 2–16, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] Joachim Denil, Maris Jukss, Clark Verbrugge, and Hans Vangheluwe. *System Analysis and Modeling: Models and Reusability: 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, chapter Search-Based Model Optimization Using Model Transformations, pages 80–95. Springer International Publishing, 2014.

[22] Sandeep Neema, Janos Sztipanovits, Gabor Karsai, and Ken Butts. *Embedded Software: Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003. Proceedings*, chapter Constraint-Based Design-Space Exploration and Model Synthesis, pages 290–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[23] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers*, chapter An Approach for Effective Design Space Exploration, pages 33–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[24] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 224–239, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] Tripti Saxena and Gabor Karsai. *Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, chapter MDE-Based Approach for Generalizing Design Space Exploration, pages 46–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embed. Comput. Syst.*, 10(4):39:1–39:36, November 2011.

[27] Twan Basten, Martijn Hendriks, Nikola Trčka, Lou Somers, Marc Geilen, Yang Yang, Georgeta Igna, Sebastian Smet, Marc Voorhoeve, Wil Aalst, Henk Corporaal, and Frits Vaandrager. *Model-Based Design of Adaptive Embedded Systems*, chapter Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems, pages 189–244. Springer New York, New York, NY, 2013.