Bridging XML and UML - An Automated Framework

Arthur Kühlwein[1], Sebastian Reiter[1], Wolfgang Rosenstiel[2] and Oliver Bringmann[2]
[1]*FZI Forschungszentrum Informatik, 76131 Karlsruhe, Germany*
[2]*Eberhard Karls Universität Tübingen, 72076 Tübingen, Germany*

Keywords: Extensible Markup Language, Unified Modeling Language, XML Schema Definition.

Abstract: A large variety of data is serialized and exchanged using XML. Model-driven activities can benefit from XML data, as shown by various approaches to IP-XACT and UML integration. However, these approaches are inherently time consuming, error-prone, and inflexible due to the manual effort involved. We propose an automated framework for integrating arbitrary XML data into UML models using an automatically generated UML profile corresponding to the structure of the XML data. User-defined XML-to-UML mappings further enhance this integration. Our approach mitigates the aforementioned issues while providing the same benefits.

1 INTRODUCTION

During the last decade, model-driven engineering (MDE) has increasingly gained traction in both research and industry. Among modeling languages, the Unified Modeling Language (UML) (OMG, 2015) and its derivates are certainly the most prominent representatives. UML allows modeling of software structure and behavior and is widely used in the software engineering domain. In MDE, it forms the basis for a large host of model-driven methodologies, frameworks, and tools. To specialize the language for specific domains, UML supports extending its metamodel via so-called profiles. An example is the profile for Modeling and Analysis of Real Time Systems (MARTE) (OMG, 2011), which targets embedded real-time systems and augments the expressiveness of UML with detailed timing and hardware platform specifications.

The Extensible Markup Language (XML) (W3C, 2008) is a widely used textual human- and machine-readable data serialization and exchange format. Well-formed XML data always conforms to a certain structure. Although not obligatory, this structure can be made explicit in an XML Schema Definition (XSD) (W3C, 2012a,b). If no XSD exists, one can be automatically inferred from the structure of the XML data (Bex et al., 2007).

A variety of information is represented in the XML format, ranging from simple web service data[1], to complex network configurations[2] and system descriptions[3]. This includes ad-hoc formats used exclusively within a company or even a single department, but also standardized formats, such as the IEEE 1685 (IP-XACT) standard (IEEE, 2010), which provides detailed XML-based descriptions of intellectual property (IP) and is widely used in industry.

There is much to be gained from bridging the worlds of XML and UML by augmenting UML models with XML data.

In the embedded systems domain, a number of proposals exist for leveraging synergies between XML and UML. For example, approaches for using IP-XACT together with UML and MARTE models have been proposed, aiming at enhancing code generation, model-driven analyses, and HW/SW codesign. While the framework is applicable to arbitrary XML data, we utilize examples for bridging IP-XACT and UML in order to facilitate comparison.

However, the existing approaches for briding IP-XACT and UML still include varying degrees of manual effort, making them inherently tedious, error-prone, and time consuming, as well as hindering collaboration and communication among the users of XML and UML, increasing development time. These adverse effects are aggravated if the structure of the XML format changes, which can potentially happen frequently in the case of ad-hoc or in-house formats.

[1]Such as the Web Services Description Language (WSDL)

[2]Such as libvirt's Network XML Format, or the Network Configuration Protocol (NETCONF)

[3]Such as AUTOSAR system configuration descriptions

Kühlwein, A., Reiter, S., Rosenstiel, W. and Bringmann, O.
Bridging XML and UML - An Automated Framework.
DOI: 10.5220/0007470403780385
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 378-385
ISBN: 978-989-758-358-2

In addition, validation of the XML data is not considered, i.e. whether the data in the XML conforms to the constraints given in the corresponding XSD.

In this paper, we propose a framework for the automated integration of XML data of arbitrary XML schemas into UML models, creating a unified UML and XML data base. This single source of data can serve as a pivot for collaboration, enabling quick round-trip engineering and promoting codesign.

In particular, changes made to XML descriptions, such as IP-XACT, can quickly be reflected in the UML side, which opens up new possibilities with respect to model-based activities, such as analyses, model-to-model or model-to-text transformations, code generation in particular, and simulations.

By specifying mappings between XSD and UML elements, UML models can be created from complex XML data. Using our approach, UML can also be used as a frontend for editing arbitrary XML data with validation according to the constraints given in the corresponding XSD.

The remainder of this paper is structured as follows. In Section 2, we discuss related work. Section 3 presents our main contribution, the automated framework. In Section 4, we illustrate our contribution with a typical industrial use case based on IP-XACT and MARTE. Finally, we conclude our work in Section 5.

2 RELATED WORK

There exists a large body of research on the representation of XML schemas in UML. For instance, see (Bernauer et al., 2004) for an overview.

However, to the best of our knowledge, no prior work addressing the integration of arbitrary XML data into UML models exists. As mentioned in the introduction, there have been numerous approaches at integrating IP-XACT and UML in the field of model-driven embedded software and hardware development, since IP-XACT and UML, in particular MARTE, have a complementary relationship with respect to the information they provide. In the following, we will outline these approaches.

Initial efforts created dedicated UML profiles which enabled integrating IP-XACT data to some degree. Arpinen et al. (2008) presented a codesign framework based around their TUT-Profile. Their framework allows structural hardware platform designs created in UML to be exported to IP-XACT designs. A similar approach was employed by Revol et al. (2008), who developed the ESL profile, which creates a bridge between the hardware resource model of MARTE and IP-XACT. In their proposed methodology, the hardware platform is modeled by using MARTE together with their profile. The model can then be exported to IP-XACT designs and SystemC code. Khan et al. (2008a, 2009) proposed to use MARTE models to create IP-XACT descriptions, which they enrich with timing information provided by MARTE. Khan et al. (2008b) embedded IP-XACT descriptions in MARTE using a custom profile. Schattowksy et al. (2009) created a comprehensive UML profile for IP-XACT which effectively allows UML to be used as a frontend for editing and managing IP-XACT data. Weissnegger et al. (2017) also extended MARTE with a custom profile for IP-XACT data to create detailed models of hardware platforms, from which they generate virtual prototypes.

An approach similar to the one we propose was presented by Tallec et al. (2011). IP-XACT descriptions are extracted from SystemC code, which then are used to synthesize MARTE models. Transformation of MARTE models back to IP-XACT descriptions is proposed as well, in which additional information provided by the MARTE model is stored in IP-XACT *vendor extensions*. However, their transformations do not use the entire IP-XACT standard, abstracting away some information.

Herrera et al. (2012) introduced a framework for the automatic generation of executable models enabling early design space exploration. In their proposed framework, IP-XACT descriptions of the hardware platform are automatically generated from MARTE models. Their framework does not integrate IP-XACT data into the MARTE models.

Ochoa-Ruiz (2013) proposed a high-level methodology for IP-XACT and MARTE codesign that introduces transformation rules between both standards. However, his work does not cover all aspects of IP-XACT and MARTE and abstracts from certain concepts.

All these approaches have a number of drawbacks. The custom profiles that enable IP-XACT and UML codesign are hand-crafted and mostly based on older versions of the IP-XACT standard. This implies that refactoring these profiles to conform to the most recent version of IP-XACT has to be done manually in a tedious and error-prone process. Most approaches only integrate parts of the IP-XACT standard or abstract away from it, leaving out potentially important information. In addition, the codesign direction of a large portion of these approaches is unidirectional, prohibiting round-trip engineering.

With our framework, custom profiles for XML handling can be generated automatically and easily adapted to evolutions of the underlying XSDs. The

generated profile reflects the XSD in its entirety. By using our proposed XSD to UML mapping specification, abstractions of XML elements in the UML model can be created. Finally, our framework supports round-trip engineering by providing export of integrated XML data from UML models.

In principle, a large number of these approaches can at least in part be implemented or emulated using our framework, making it a meta-framework in a sense. We illustrate this point in our use case in Section 4.

3 AN AUTOMATED FRAMEWORK FOR INTEGRATING XML INTO UML

In this section, we present our main contribution, which is an automated framework for integrating XML data into UML models. Fig. 1 gives a conceptual overview of our framework.

XML files in specific formats, such as IP-XACT, are created and edited by their corresponding tools. Analogously, UML data, which may include elements from standard or custom profiles is created and edited by specific UML tools.

Both the XML and the UML data are assumed to be well-formed, i.e. they conform to metamodels, which can be either explicit or implicit. In the case of XML this metamodel is described by an XSD. UML models conform to the Meta Object Facility (MOF) OMG (2016) architecture, in which UML profiles can be considered a metamodel as well.

To forster a seamless usage between both formats, import and export capabilites are required. Since we regard UML as the core model, integration of XML data into the UML model requires an import.

Given an XML file, its data is imported into a UML model by creating a UML profile corresponding to the XSD of the XML. New stereotyped UML elements are then created in the model, corresponding to the XML entities in the XML file. The stereotype applications contain the data in the XML file.

We chose to apply the stereotypes to UML classes, because they can be nested. This allows us to reflect the XML element hierarchy in the UML model.

Optionally, an additional user-defined mapping from XML data to existing UML elements can be provided. This enables the usage of the XML data with existing metamodels and modeling methodologies. In the remainder of this paper, we always refer to this kind of user mapping if we use the term *mapping*, unless explicitly stated otherwise.

The UML elements containing the XML data can be exported into valid XML again.

With the exception of the specification of the user-defined mapping and the XML file, all of the aforementioned steps are automated. In the following, we will discuss the specific aspects of our framework in detail.

3.1 XML Import

The data contained in the XML files is imported into a UML model by means of an automated process. Central to this process is the core mapping of XSD to UML elements, which is used to generate a UML profile that reflects the structure of the XSD. This core mapping is outlined in Table 1.

XSD entities are mapped to stereotypes, attributes of mapped stereotypes, primitive data types, or enumerations.

XSD elements are mapped to stereotypes. If the element is contained by another element, i.e. not at the root level of the XSD, an additional attribute in the stereotype mapped to the containing element is created. This attribute is of the type of the stereotype mapped to the contained element.

XSD attributes are mapped to attributes of the stereotype mapped to the containing element.

If an XSD complex type is derived from a simple type, a primitive data type is created in the UML profile. Otherwise, a new stereotype is created.

XSD simple types with enumeration facets are mapped to UML enumerations. Otherwise, they are

Table 1: Core XSD to UML mapping.

XSD	Condition	UML
element	root-level	*stereotype*
	otherwise	*stereotype and attribute of containing mapped stereotype*
attribute		*attribute of containing mapped stereotype*
complexType	derived from simple type	*primitive data type*
	otherwise	*stereotype*
simpleType	has enumeration facet	*enumeration*
	otherwise	*primitive data type*
constraints		*OCL constraints*

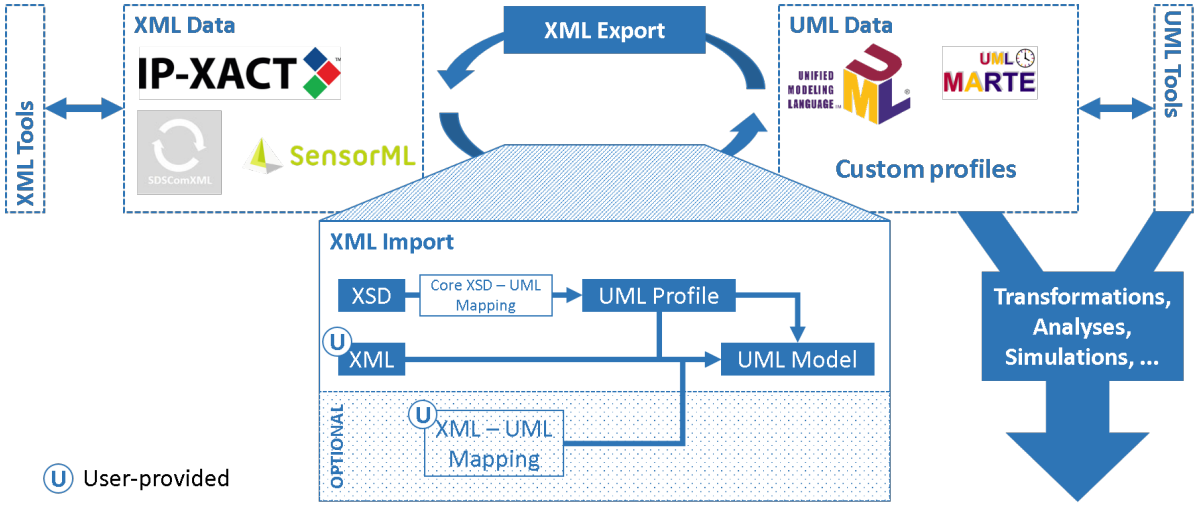

Figure 1: Overview of our proposed automated framework for integrating XML into UML.

mapped to UML primitive data types.

XSD allows the specification of constraints on XSD elements and their attributes, and any piece of XML data that conforms to an XSD must adhere to these constraints. In order to facilitate data validation, constraints in the XSD are mapped to corresponding Object Constraint Language (OCL) (OMG, 2014) constraints on the mapped elements in the generated UML profile.

Inheritance relationships between XSD elements are reflected via generalizations between the corresponding stereotypes in the profile.

For identification of the UML profile with the corresponding XSD, we assign the namespace Uniform Resource Identifier (URI) of the UML profile to the one targeted in the XSD. It is possible that a given XML file does not conform to a particular XSD, in which case the XSD can be automatically inferred from the XML data.

Using the UML profile, appropriately stereotyped classes in the model are created from the XML data, i.e. we map XML elements to stereotyped UML elements. For each XML element instance, we generate a UML class with the appropriate stereotype. The attributes and references of the stereotypes are set to the values contained in the XML. The stereotyped classes are nested according to the hierarchy of the corresponding XSD elements. This import fully integrates the XML data into the UML model, enabling access to all contained data from within the model.

If required, this integration can be refined by providing a user-defined mapping between XML and UML elements. Depending on the expressivity of the mapping language, models can be created from arbitrarily complex XML data. For instance, one could easily imagine UML network topology diagrams being created from a single XML network topology description, such as the one described by Balon (2008).

This mapping is strictly additive, meaning that the mapped model elements are created in addition to the model elements representing the imported XML data. In other words, the model contains two additional sets of model elements after the XML import: stereotyped classes representing the XML data and model elements created according to the provided mapping.

Of course, the mapping could directly transform the former elements, which would avoid data duplication and synchronization issues. However, this could result in data loss and would make the export of the integrated XML data more difficult, particularly when the imported data is aggregated. We thus argue that an additive mapping is better suited for our framework.

Once the XML data has been imported, the augmented UML model can be used to drive advanced model-based activities, such as code generation, analyses, simulations, or verifications. Unless a custom solution is implemented, these activities usually cannot work with the model elements of the UML profile generated from the XSD, but instead require model elements from another profile. In this case, the mapping can be used to create the appropriate elements from the XML data. For instance, we can map to stereotypes from the MARTE profile, enabling tools utilizing this profile to work with the XML data, as demonstrated in Section 4.

3.2 XML Export

Because the import integrates the XML data completely, the UML elements containing the XML data

can be exported into XML that conforms to the XSD. For this, the XML data is reconstructed from the stereotypes and the values contained in them. This enables round-trip engineering and codesign, and allows UML to be used as a frontend for editing and validating XML data.

In this case however, the original XML data and its imported UML representation may become misaligned when users perform a modification of one part without synchronizing the other part appropriately.

3.3 Automation

As mentioned earlier, our framework is almost completely automated. The only part requiring manual effort is the specification of the user-defined mapping between XML and UML elements.

The generation of a UML profile that corresponds to a given XSD is a relatively straightforward task. For instance, detailed mappings between XSD and the Ecore[4] metamodel have been described in (EMF, 2004). From this Ecore representation of the XSD, a UML profile can be easily generated.

In case an XML file does not conform to a particular XSD, many approaches exist for automatically generating an XSD from the structure of the XML. Such an approach is described in (Bex et al., 2007), for instance.

The generation of UML elements from XML data and the generation of XML data from its UML representation can be easily automated as well.

3.4 Proof-of-Concept Implementation

To demonstrate the technical feasibility of our automated framework, we have created a proof-of-concept implementation based on the Eclipse Modeling Framework (EMF[5]) and the Papyrus Modeling Environment[6].

Given an XML file, the XML importer engine first generates an Ecore model of the corresponding XSD using the XSD conversion integrated into EMF. From this Ecore model, a UML profile is built. If the XML file does not conform to an XSD, it is automatically generated using Wizools.org XSD Gen[7].

This profile is then loaded and applied to the UML model in the Papyrus Modeling Environment.

EMF automatically extracts value restrictions from the XSD and converts them to corresponding validation rules in the generated Ecore model. These validation rules have the same power as OCL constraints and are recognized and enforced by the Papyrus Modeling Environment in the UML model. Enforcing the validation rules guarantees that any modification of the XML data from within UML is conformant to the XSD.

Users can specify the import location within the UML model in our implementation. Once the profile is applied to the UML model, our implementation adds a stereotype application corresponding to the root-level element in the XML file to the selected model element, and stereotyped classes corresponding to the sub-elements in the XML file are created as children of the selected model element. The attributes of the applied stereotypes are then filled with the values from the XML data.

In our implementation, users have the option of specifiying the mapping between the XML elements and UML elements for the XML import. Our current implementation facilitates this mapping using the Epsilon Transformation Language (ETL) (Kovolos et al., 2008). The ETL transformation is executed after the XML data has been imported, because the ETL model-to-model transformation cannot work on the XML data directly, but instead requires the generated UML profile.

Export of the integrated XML data from UML to XML is implemented using the standard XML serialization of EMF.

4 USE CASE: IP-XACT AND MARTE

In this section, we present a use case which will illustrate two points. We show that we can import IP-XACT data into a MARTE model using the proof-of-concept implementation of our framework without loss of data, demonstrating its technical feasiblity. We also show how our framework is able to emulate some of the existing approaches for integrating IP-XACT and UML. In particular, we refer to (Schattowksy et al., 2009), (Khan et al., 2008a), and (Khan et al., 2008b), which propose manual approaches for integrating IP-XACT and UML. Our framework is able to create similar UML models.

Our use case represents a typical scenario in embedded software development. At a large semiconductor manufacturer, firmware for a new embedded device is being developed. This device will contain commercial off-the-shelf IPs from various external manufacturers. A basic MARTE model of the device has been created and IP-XACT descriptions of the IPs have been supplied, which now need to be integrated

[4]https://wiki.eclipse.org/Ecore

[5]https://www.eclipse.org/modeling/emf/

[6]https://www.eclipse.org/papyrus/

[7]https://github.com/wiztools/xsd-gen

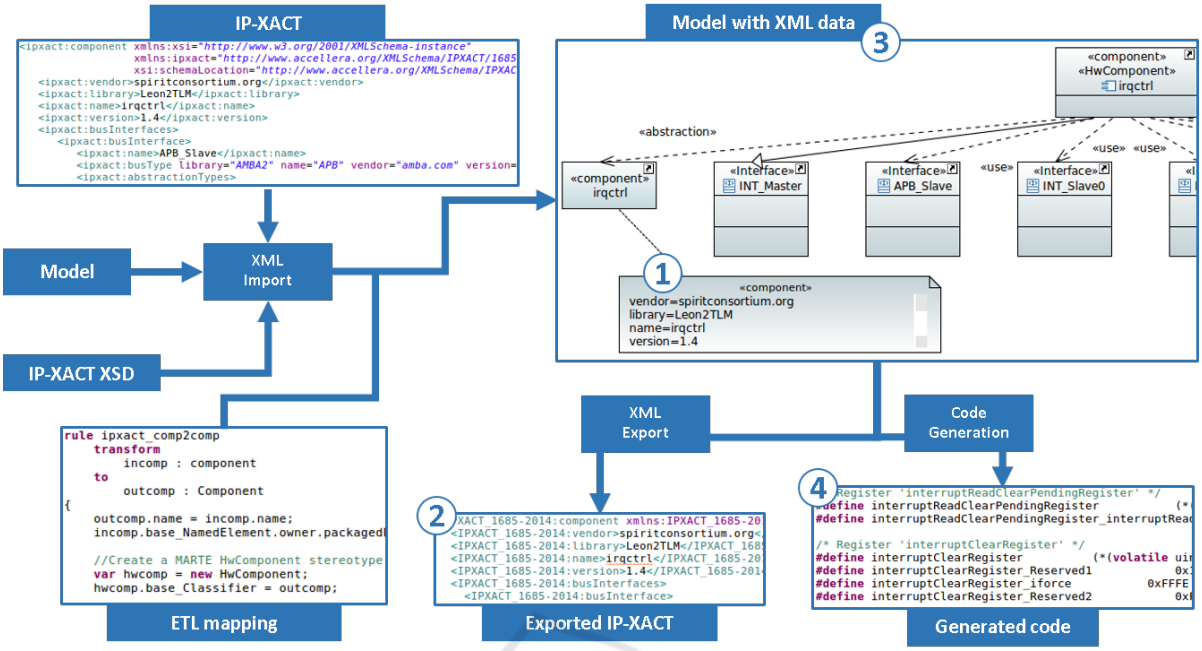

Figure 2: Augmenting our MARTE model with IP-XACT data.

into the model.

To integrate the external IPs into the UML model, their IP-XACT descriptions are imported using our framework. Fig. 2 illustrates this integration process.

For the XML import, users need to specify the XML file containing the data and the XSD it conforms to. The appropriate UML profile is generated accordingly. The profile is then applied to the UML model, so the stereotypes from the generated profile are available to the model. Finally, the corresponding stereotyped UML elements are created from the XML data.

As a result of the import in our example, a class with a *«component»* stereotype is created in the model (number (1) in Fig. 2), which holds the information of the IP-XACT description.

This direct import is similar to the approach described in (Schattowksy et al., 2009). For instance, the *«component»* stereotype of the generated profile is corresponding to the *«IP-XACT_Component»* stereotype of the profile proposed by the authors. The IP-XACT data can now be edited using the UML editor and exported into valid IP-XACT XML again (number (2) in Fig. 2). But in contrast to the aforementioned approach, data validation is provided by our framework due to the EMF validation rules extracted from the XSD. For instance, the well-formedness rules for IP-XACT port names are enforced by the data validation.

To fully integrate the IP into the MARTE model, an ETL mapping has to be specified.

Utilizing this mapping, appropriate MARTE model elements are created from the imported XML data.

In our example, the mapping creates a MARTE *«HwComponent»* for each IP-XACT *«component»*. The IP has a number of bus interfaces, which are also included in the model. For this, the mapping is extended to create required interfaces for slave ports and provided interfaces for master ports. This mapping emulates some aspects of the approaches proposed in (Khan et al., 2008a) and (Khan et al., 2008b).

The result of this mapping can be seen in number (3) in Fig. 2. To indicate that the *«HwComponent»* has been created from the IP-XACT *component*, an *abstraction* relationship is used.

The MARTE model now contains information about the register map of the imported IP. From this model, C headers for register access of the IP can be generated, as illustrated in number (4) in Fig. 2.

The integration of the XML data into the UML model is completely automated. The only part requiring manual effort is the specification of the additional mapping. In contrast, the approaches outlined in Section 2 would first require manual analysis of the XSD in order to handcraft an appropriate UML profile that is capable of reflecting the XML data. Then, the XML data would need to be integrated using this profile, either manually or with a bespoke importer that reads the XML and generates the corresponding stereotyped UML elements in the model.

This entire process is time-consuming, tedious,

and error-prone.

It is also extremely inflexible, because this entire process has to be started again if XML data in another format is going to be included or in case the XSD changes. While it can be argued that the frequency of such changes is usually in the order of years or even decades in case of heavily standardized formats such as IP-XACT, other formats may undergo changes more frequently.

Each time such a change occurs, existing approaches require the manual updating of the profile, any custom code that is responsible for handling the XML import, as well as code that works on the model, such as code generators.

In our approach, the parts requiring manual effort are now restricted to the specification of the mapping and any code that works on the imported model. The rest is handled automatically.

With respect to flexibility, our approach also allows the quick integration of any data expressed in a non-XML data format that can be automatically transformed into XML. For instance, consider the JavaScript Object Notation (JSON[8]), which is a prominent alternative to XML and the data format used by some electronic design automation tools. Using our framework, this data can be quickly integrated into a UML model. The data is first converted to XML[9].

The corresponding XSD is then inferred, and the data is imported into the model using our framework. All these steps are performed automatically.

5 SUMMARY AND CONCLUSIONS

We presented a framework for automatically integrating arbitrary XML data into UML models. XML data is integrated by creating a UML profile that corresponds to the XSD to which the XML conforms. Instances of the appropriately stereotyped classes are then created in the UML model and their values are set according to the XML data. Users have the option to refine this integration by specifying an XML to UML mapping, which is automatically executed at the end of the import. The integrated XML data can be exported to valid XML from the UML model. Validation of the XML data is ensured by corresponding OCL constraints in the UML profile.

The integration of XML data in UML models offers a number of benefits. For instance, advanced model-based analyses and simulations, as well as more comprehensive code generation can be performed from a single UML model augmented with XML data. Such a model can also serve as a pivot for collaboration between users of XML- and UML-based data, promoting communication and accelerating codesign. In addition, UML can be used as a frontend for editing and validating XML data.

We demonstrated the technical feasibility of our framework with a proof-of-concept implementation, which we based on the Eclipse Modeling Framework, the Papyrus Modeling Environment, and the Epsilon Transformation Language. We also presented an industrial use case in which we integrate IP-XACT descriptions into MARTE models using our framework, illustrating its potential usage and the benefits it provides compared to existing approaches. Mainly, our framework greatly reduces manual effort and provides a much larger degree of flexibility.

Since the imported XML data can be edited in UML, one major issue of our approach is data consistency. The original XML data and its imported UML representation may become misaligned in case of modifications. In addition, if a mapping between XML and UML is applied, the mapped data needs to be kept consistent with the UML representation of the XML data. We will investigate how the data consistency issue can be addressed, since our current approach does not provide any consistency enforcement mechanism.

In future work, we will also further explore potential mappings between XML and UML, as we believe these mappings can pave the way for new powerful model-driven methodologies that utilize UML models augmented with XML data. This will include mappings on the level of individual attributes and references, as our current mappings are restricted to the level of individual elements.

Our investigations will also address the question of how this mapping can be extended to the generation of the UML profile from the XSD, so in addition to the stereotypes reflecting the XSD elements, the profile may contain additional stereotypes that aggregate or abstract from the data of the XSD elements in some way. This kind of mapping would enable our framework to fully emulate some of the approaches mentioned in Section 2, making it a true meta-framework.

[8] https://www.json.org/

[9] A number of implementations exist, for example https://github.com/lukas-krecan/json2xml

ACKNOWLEDGEMENTS

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) in the ITEA3 project COMPACT under grant 01IS17028C. The authors are responsible for the content of this publication.

REFERENCES

Arpinen, T., Salminen, E., Hännikäinen, M., and Hämäläinen, T. D. (2008). Model-driven Approach for Automatic SPIRIT IP Integration. In *PProceedings of the Fifth International UML-SoC DAC Workshop*.

Balon, S. (2008). A standard XML Format for a Network Topology Representation. http://totem.run.montefiore.ulg.ac.be/doc/UserGuide/node4.html.

Bernauer, M., Kappel, G., and Kramler, G. (2004). Representing XML Schema in UML – A Comparison of Approaches. In *ICWE 2004: Web Engineering*, pages 440–444.

Bex, G. J., Neven, F., and Vansummeren, S. (2007). Inferring XML Schema Definitions from XML Data. In *Proceeedings of the 33rd International Conference on Very Large Data Bases*, pages 998–1009.

EMF (2004). XML Schema to Ecore Mapping. https://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf.

Herrera, F., Posadas, H., Villar, E., and Calvo, D. (2012). Enhanced IP-XACT Platform Descriptions for automatic Generation from UML/MARTE of fast Performance Models for DSE. In *15th Euromicro Conference on Digital System Design*, pages 692–699.

IEEE (2010). IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. Technical report.

Khan, A., Mallet, F., André, C., and Simone, R. (2008a). MARTE Timing Requirement and SPIRIT IP-XACT.

Khan, A., Mallet, F., André, C., and Simone, R. (2008b). Modeling SPIRIT IP-XACT with UML MARTE.

Khan, A., Mallet, F., André, C., and Simone, R. (2009). IP-XACT Components with abstract Time Characterization. In *2009 Forum on Specification & Design Languages*, pages 1–6.

Kovolos, D. S., Paige, R. F., and Polack, F. A. C. (2008). The Epsilon Transformation Language. In *ICMT 2008: Theory and Practice of Model Transformations*, pages 46–60.

Ochoa-Ruiz, G. (2013). *A high-level Methodology for automatically generating dynamically reconfigurable Systems using IP-XACT and the UML MARTE Profile*. PhD thesis, Université de Bourgogne.

OMG (2011). UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Technical report.

OMG (2014). Object Constraint Language. Technical report.

OMG (2015). OMG Unified Modeling Language^{TM}(OMG UML). Technical report.

OMG (2016). OMG Meta Object Facility (MOF) Core Specification. Technical report.

Revol, S., Taha, S., Terrier, F., Clouard, A., Gerard, S., Radermacher, A., and Dekeyser, J.-L. (2008). Unifying HW Analysis and SoC Design Flows by Bridging Two Key Standards: UML and IP-XACT. In *Distributed Embedded Systems: Design, Middleware and Resources*, pages 69–78.

Schattowksy, T., Xie, T., and Müller, W. (2009). A UML frontend for IP-XACT-based IP management. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 238–243.

Tallec, J.-F. L., DeAntoni, J., Simone, R., Ferrero, B., Mallet, F., and Maillet-Contoz, L. (2011). Combining SystemC, IP-XACT and UML/MARTE in Model-based SoC Design. In *Workshop on Model Based Engineering for Embedded Systems Design*.

W3C (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). Technical report.

W3C (2012a). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Technical report.

W3C (2012b). W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. Technical report.

Weissnegger, R., Pistauer, M., Schachner, M., Kreiner, C., Römer, K., and Steger, C. (2017). SaVeSoC - Safety aware Virtual Prototype Generation and Evaluation of a System on Chip. In *Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering*.