Cureus Journal of Computer Science

Open Access Original Article

Towards the Systems Requirement Modeling With a Single Language

Corina Abdelahad [1] , Daniel Riesco [1] , Carlos Kavka [2]

1. Computer Science, National University of San Luis, San Luis, ARG 2. Science and Technology, Luxembourg Institute of Science and Technology, Esch-sur-Alzette, LUX

Corresponding author: Corina Abdelahad, cabdelah@unsl.edu.ar

Received 06/29/2025
Review began 07/03/2025
Review ended 10/10/2025
Published 10/16/2025

DOI:
https://doi.org/10.7759/s44389-025-08526-x

Abstract

In the field of engineering, the design process frequently involves numerous graphical formats, lacking a unified standard. The Object Management Group (OMG) introduced SysML to model system requirements and their interrelationships with other modeling elements; however, SysML primarily addresses functional and non-functional requirements, disregarding the modeling of decision requirements prevalent in numerous systems. Conversely, the OMG also proposes the Decision Model and Notation (DMN) standard for modeling decision requirements. This paper proposes the use of SysML as a unified language for modeling various types of requirements, with a particular focus on decision requirements. To achieve this, we present a model-to-model transformation using Query/View/Transformation to transform a DMN requirement model to the SysML requirement model.

Categories: Software Development, Modeling
Keywords: dmn, decision requirement, sysml, transformation, qvt

Introduction

In recent years, system development has focused on model creation. These models are abstractions and serve as the tool for engineers to deal with the complexity of systems, as abstraction allows them to focus only on information that is considered significant or relevant. A good design of all requirements, including the decision requirements, is the key to successful system development. Knowing the modeling language enables the creation of well-structured designs, and, in addition, using a single modeling language for the system development significantly helps in fast and effective communication between stakeholders.

Model-Driven Architecture (MDA) [1], the Object Management Group's (OMG) [2] implementation of Model-Driven Engineering (MDE) [3-4], proposes a software development process where models and their transformations are the central elements. In this process, software is developed by building one or more models and transforming them into other models or into executable code [5]. With this objective in mind, the OMG introduced a standard language to express queries and define model transformations: Query/View/Transformation (QVT) [6]. This specification relies on two other OMG standards: Meta-Object Facility (MOF) [7] and Object Constraint Language (OCL) [8]. Using QVT for specifying transformations enhances the reusability of technologies that adhere to OMG standards. The concept of QVT is primarily based on defining a language for queries on MOF models, establishing a standard for generating views that display specific aspects of modeled systems, and developing a language for describing transformations of MOF models [9].

On one hand, the OMG defined systems modeling language (SysML) [10], a standardized general-purpose modeling language based on unified modeling language (UML). It is built around four key pillars - Requirements, Structure, Behavior, and Parametrics - each represented through dedicated diagrams that offer distinct perspectives on the system. SysML enables the relation of requirements to other requirements and modeling elements using its defined relationships [11-12]. However, while it supports the modeling of functional and non-functional requirements, SysML does not explicitly address the modeling of decision requirements, which are prevalent in many systems [13].

On the other hand, the OMG introduced a standard specifically for decision modeling known as Decision Model and Notation (DMN) [14]. DMN enables the representation of decision models by separating decision logic from control flow logic in business processes [15].

This paper presents a QVT transformation of a DMN requirements diagram to a SysML requirements diagram in order to unify the modeling the requirements, uniting all the requirements of a system in a single SysML diagram.

Related work

SysML is highly intuitive and widely adopted for modeling complex systems [16]. Several works are related

to the concept of transformation between models, and many of them use SysML. For example, Wang et al. [17] introduce an automated tool for transforming SysML State Machine diagrams into the Uppaal-timed automata, thereby integrating formal methods into Model-Driven Development. The authors present semantic mapping rules implemented using ATLAS Transformation Language (ATL). Additionally, they argue that this proposal offers several benefits, including the promotion of formal methods in SysML modeling and the improvement of system robustness in the early stages of development. The tool's effectiveness is demonstrated through a case study.

Miao et al. [18] propose a prototype tool that converts SysML state machine diagrams into C++ code using the XML Metadata Interchange (XMI) standard. The proposed approach consists of exporting SysML models in XMI format, parsing the XMI file, establishing mapping rules, and generating C++ code. This approach facilitates model simulation in open-source software environments, verifying the correctness and feasibility of the system design.

Li et al. [19] present a method for transforming SysML models between different software tools using the XMI standard, ensuring interoperability in Model-Based Systems Engineering. The structure of XMI files is analyzed, focusing on key elements such as SysML metadata, UML elements, stereotypes, and extensions. The method transforms complex SysML diagrams, including internal block diagrams and state machine diagrams, while preserving structural integrity, properties, and their relationships. This work highlights the importance of standardization in Model-Based Systems Engineering and offers a practical solution to enhance tool interoperability through the XMI standard.

However, to the best of our knowledge, no existing research has incorporated SysML requirements within model transformation approaches.

On the other hand, while DMN was developed to work alongside Business Process Model and Notation, as highlighted by various studies including Thanh et al. [20], Tavantzis et al. [21], and Shen et al. [22], other research, such as Weske [23] and Maleki and Gailly [24], explores its integration with different languages and notations. Nevertheless, existing research has not explicitly addressed the modeling of decision requirements using SysML.

Materials And Methods

Transformation architecture

Our proposal seeks to model decision requirements using SysML, with the aim of unifying the modeling language and showing the connection they have among these requirements, the connection with other system requirements and in turn, other modeling elements. Additionally, in this way, it is possible to follow the traceability of these requirements.

Both DMN and SysML requirement models have a graphical and an XML representation, enabling the use of transformation languages like QVT to convert one model into another. The QVT transformations describe relations between the source metamodel and the target metamodel while allowing the precise definition of generic transformations between metamodels, which are specified in MOF. An important feature is that the transformations can be bidirectional. Since the execution of transformations can occur in either direction, the direction is determined by the set of domains associated with the target model type. Each domain defines a candidate model.

As illustrated in Figure *1*, the defined transformation is applied to a source model - an instance of the DMN metamodel - to produce a resulting model that conforms to the SysML metamodel. The specific models to which the transformation defined in the metamodel level is applied in order to obtain DMN requirement models are shown at the middle level. The lower level represents the instances of the models. More details will be provided in the following subsections.

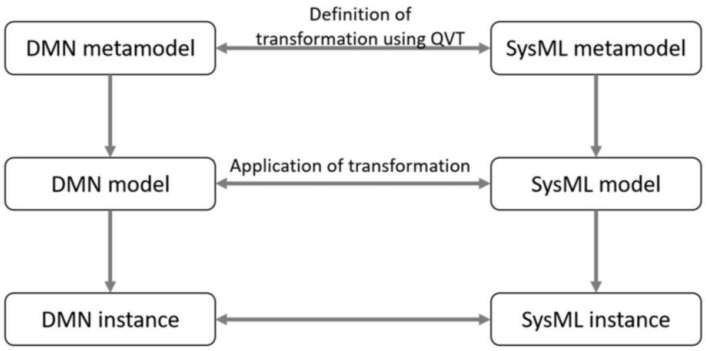

FIGURE 1: Transformation architecture.

DMN, Decision Model and Notation; SysML, Systems Modeling Language; QVT, Query/View/Transformation

Having a bidirectional transformation enables engineers to visualize DMN models in SysML and vice versa. This allows engineers to work in the modeling language they are most comfortable with, promoting greater flexibility. The DMN decision requirements diagram, DMN model, illustrates how decision requirements relate to each other, how they connect with the input data involved in these requirements, their relationship with Business Knowledge (BK) Models, and with Knowledge Sources (KS) [25].

Each element of the DMN diagram [14] is detailed below:

- A Decision requirement (*Decision* class in Figure *2*) determines an output from inputs by applying some decision logic that can refer to one or more BK Models. Decisions can be decomposed into sub-decisions.

- A BK Model (*BusinessKnowledgeModel* class in Figure *2*) represents a function that encapsulates BK.

- An Input Data (*InputData* class in Figure *2*) represents information used as input for one or more Decisions and KS.

- A KS (*KnowledgeSource* class in Figure *2*) denotes an authority for a Decision or for a BK Model. KS represents the source of technical knowledge for making a decision, such as regulations or policies on how a decision should be made.

The relationships in the decision requirements diagram are described below:

- An Information Requirement (*InformationRequirement* class in Figure *2*) denotes the Input Data to a Decision, or Decision to a Decision.

- A Knowledge Requirement (*KnowledgeRequirement* class in Figure *2*) denotes the invocation of a BK Model, i.e. Decisions are related to Knowledge Models through it.

- An Authority Requirement (*AuthorityRequirement* class in Figure *2*) denotes the dependency of one element on another that will act as a KS. It allows linking a *KnowledgeSource* with a *Decision* or with an *InputData*, see Figure *2*.

A partial view of the DMN metamodel with the metaclasses involved in the relations described in this work is shown in the UML class model presented in Figure *2* [14]. It illustrates what was mentioned above, i.e. how Decisions are related to other modeling elements. *Decisions* are connected between themselves and with *InputData* through *InformationRequirement*. Through *AuthorityRequirement*, *Decisions* are related to *KnowledgeSource* and the latter to *InputData*. Through *KnowledgeRequirement*, *Decisions* are related to the *BusinessKnowledgeModel* [14]. It is important to mention that *Definitions* is the main label in a DMN model, i.e. the *Definitions* class is the outermost object for all DMN elements, i.e. all these elements are contained within it.

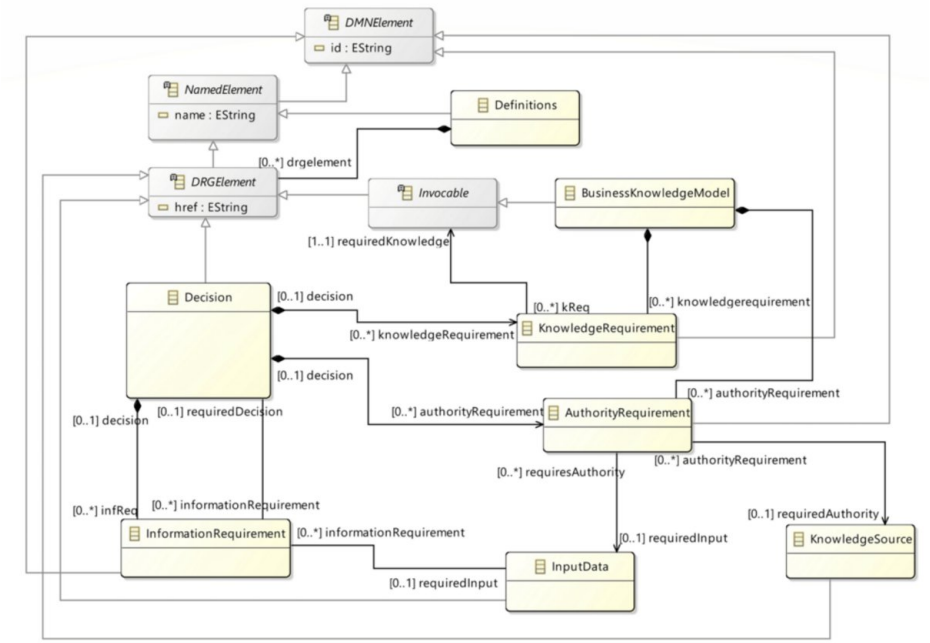

FIGURE 2: Simplified DMN metamodel.

DMN, Decision Model and Notation.

As mentioned before, SysML has been defined as an extension of UML, and therefore some SysML classes are instances or generalizations of UML metaclasses.

The simplified SysML metamodel, depicting the pertinent classes for this article, is illustrated in Figure *3* [10]. This metamodel contains the classes along with their relationships, extracted from the SysML specification [10] and the UML specification [26]. SysML, like UML, has stereotypes, an extension mechanism to extend the standard. Modelers and modeling tools can use this extension mechanism to add new building blocks with additional properties and constraints without breaking compliance with the standard. By utilizing stereotypes, non-functional requirements can be explicitly denoted using the «nonFunctionalRequirement» stereotype, while decision requirements can be marked with the «decisionRequirement» stereotype. As illustrated in the figure, the extension of a class is represented by a filled arrowhead pointing from the stereotype to the class being extended.

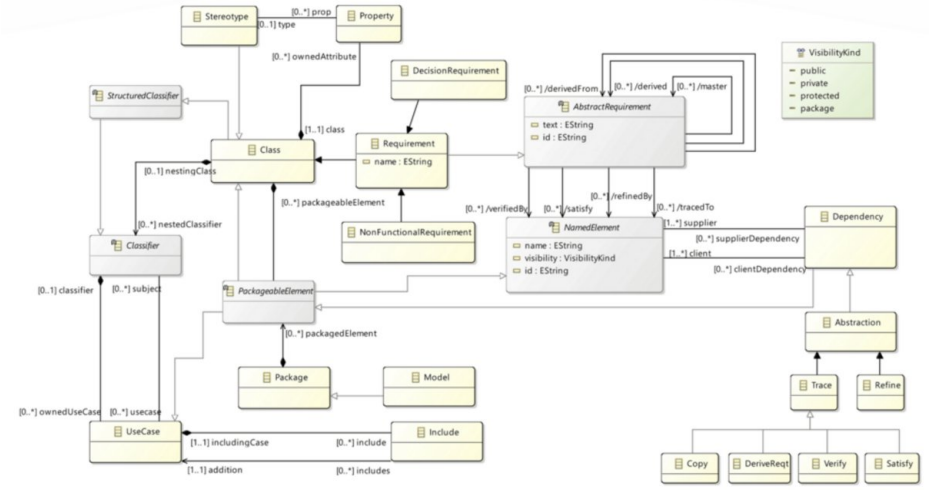

FIGURE 3: Simplified SysML metamodel.

SysML, Systems Modeling Language.

Graphical example of the transformation

As mentioned earlier, the DMN and SysML language feature both graphical and XML-based representations. This subsection presents an example of a transformation from the point of view of the designer, who expects to get a SysML requirements diagram to be obtained from a previously defined DMN diagram as a result of the transformation process.

Figure *4* presents a generic DMN requirements diagram alongside its corresponding SysML requirements diagram representation. The DMN requirements diagram consists of decisions, input data, BK, and KS.

Decisions can be decomposed into sub-decisions, and according to the DMN specification, these decisions are considered requirements [14]. KS can represent company policies, regulations, etc.; therefore, based on Somerville's classification of non-functional requirements [27], KS can be interpreted as non-functional requirements. BK represents a function, meaning it corresponds to functional requirements and, therefore, can be represented as use cases.

Input data refers to the information required for decisions and KS, which can be captured through the text property that each requirement has. When a KS is related to a decision, it means that the decision needs to have knowledge of its content, i.e. both have a relationship with a purpose. When a BK is related to a decision, this BK refines the decision. In SysML, purpose and refinement relationships are represented by trace and refine, respectively.

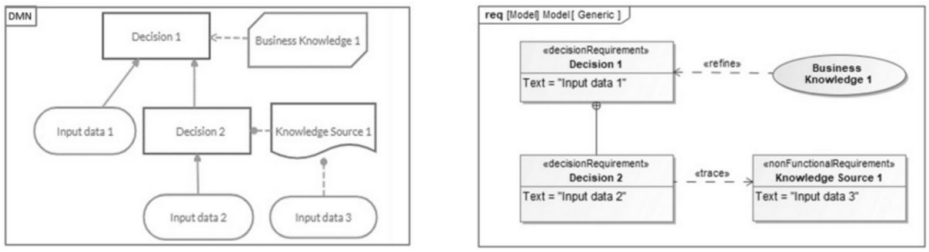

FIGURE 4: DMN to SysML.

DMN, Decision Model and Notation; SysML, Systems Modeling Language.

An important point to note is that, although the overall graphical structure may appear similar in both diagrams, their XML representations differ significantly due to distinct metamodels. There is no one-to-one correspondence between the elements of DMN and SysML.

This transformation is enabled by the fact that both diagrams are formally defined using an XML schema, which establishes a foundation for a rigorous transformation process. The details of this transformation process, along with the selected implementation code in QVT, are presented in the following sub-section.

QVT transformation

As noted earlier, a QVT transformation defines a set of relations that the elements of the models involved must satisfy. A transformation has input and output parameters, known as domains. For each output parameter, a new instance of the model is created according to its metamodel (in this case, the SysML metamodel). For each domain, there is an associated set of elements defined via patterns. A pattern can be viewed as a set of variables and a set of constraints that the model elements must satisfy. Within QVT, a relation formally defines the transformation rules. For more details on QVT, the reader is invited to visit the OMG links [6].

The transformation between the DMN metamodel and the SysML metamodel is defined as follows:

```
transformation DMNToSysML(source: dmn_m, target: sysml_m){
```

Note that this transformation takes as input a DMN model and produces a SysML model, as explained in Figure *1*.

Below, the createModel relation is presented; it is responsible for creating the main tag in a SysML model. In the when clause, the invocation to createStereotype relation, responsible for creating the «decisionRequirement» and «nonFunctionalRequirement» stereotypes is necessary in this work as expressed in the previous sections. The transformation from a decision to a requirement is not straightforward. In the where clause, the invocations of the relations for creating the classes

corresponding to the decisions and the KS can be observed since the requirements are instances of Class. In other words, a requirement is a class stereotyped with «requirement».

```
top relation createModel {
 name_model:String;
 id_model:String;
 checkonly domain source d:dmn_m::Definitions {
 name=name_model,
 id=id_model
 };
 enforce domain target s: sysml_m::Model{
 type= 'uml:Model',
 id=id_model,
 name=name_model
 };
 when {
 createStereotype(d,s);
 }
 where {
 decisionToClass(d,s);
 kSToClass(d,s);
 decisionToDecisionRequirement(d,s);
 ksToNonDecisionRequirement(d,s);
 }
} //realtion
```

Below, a partial view of the createStereotype relation is presented, illustrating how the decisionRequirement stereotype is created. In other words, it shows how the necessary XML code is generated to extend SysML, thereby enabling the modeling of decision requirements in SysML. Similarly, the nonFunctionalRequirement stereotype is created.

As mentioned earlier, decisions and KS are stereotyped classes. Therefore, it is necessary to first create the classes and then assign the stereotype. This is handled by the decisionToClass and ksToClass relation, respectively. Note that in the where clause of decisionToClass relation, the getNested relation is invoked because when a decision is related to other decisions, this relationship is represented as a composition in the requirements. Consequently, this composition must also be reflected in the classes.

```
top relation createStereotype {
 checkonly domain source d:dmn_m::Definitions {};
 enforce domain target s: sysml_m::Model{
 packagedElement = pe1:sysml_m::Stereotype {
 id='decisionRequirement_id',
 name='decisionRequirement',
 ownedAttribute=oa1:sysml_m::Property {
 id='ppt_id_123',
 name = 'base_Element',
 visibility='private',
 type= tp1:sysml_m::Stereotype{
 href='http://www.omg.org/spec/UML/20131001/UML.xmi#Element' }
 }
 },
 stereotype=st1:sysml_m::Stereotype{
 name= 'decisionRequirement',
 stereotypeHREF= 'decisionRequirement_id' },
};
}//relation

relation decisionToClass {
 id_dec:String;
 name_dec:String;
 checkonly domain source d:dmn_m::Definitions {
 decision= dec:dmn_m::Decision{
 id=id_dec,
 name=name_dec }
 };
 enforce domain target s: sysml_m::Model{
 packagedElement = pe:sysml_m::Class {
 id=id_dec,
```

```
 name=name_dec
 }
 };
where {
 getNested(d,id_dec,pe);
}
}//relation

relation kSToClass{
 id_ks:String;
 name_ks:String;
 checkonly domain source d:dmn_m::Definitions {
 knowledgeSource=ks:dmn_m::KnowledgeSource{
 id=id_ks,
 name=name_ks
 }
 };
 enforce domain target s:sysml_m::Model{
 packagedElement = pe:sysml_m::Class {
 id=id_ks,
 name=name_ks
 }
 };
}//relation
```

The relations mapping decisions to decision requirements and KS to non-functional requirements are shown below.

```
relation decisionToDecisionRequirement {
 id_dec:String;
 checkonly domain source d:dmn_m::Definitions {
 decision= dec:dmn_m::Decision{
 id=id_dec
 }
};
 enforce domain target s:sysml_m::Model{
 requirement=re:sysml_m::Requirement{
 base_Class=id_dec
 }
 decisionRequirement=dr:dmn_m::DecisionRequirement{
 base_Element=id_dec
 }
 };
 where {
 generateAbstractionD(dec, id_dec, s);
 getRelationWithKS(dec,id_dec,re);
 hasIDataDec(d,id_dec,re);
 }
}//relation

relation ksToNonDecisionRequirement {
 id_ks:String;
 checkonly domain source d:dmn_m::Definitions {
 knowledgeSource=ks:dmn_m::KnowledgeSource{
 id=id_ks
 }
 };
 enforce domain target s:sysml_m::Model{
 requirement = re:sysml_m::Requirement {
 base_Class=id_ks
 }
nonFunctionalRequirement=nfr:sysml_m::NonFunctionalRequirement{
 base_Element=id_ks
 }
 };
where {
 generateAbstractionKs(ks, id_ks, s);
 tracedToKsToKs(ks, id_ks, re);
```

```
 hasIDataKs(d,id_ks,re);
 }
}//relation
```

Observe that in the where clause of the decisionToDecisionRequirement relation, three relations are invoked. This is because a decision can be associated with a KS. In such a case, a trace dependency must be generated, as explained in subsection Transformation Architecture. This is handled by the generateAbstractionD and getRelationWithKS relations, respectively. The hasIDataDec relation is responsible for adding to the text field of a requirement the input data that may be involved in the decision being transformed into a decision requirement. Similarly, this happens in the ksToNonDecisionRequirement relation.

Results And Discussion

Experimental case applied to the approach

To demonstrate our approach, we conducted a case study that includes the partial modeling of a biodiesel distiller. This case study illustrates how to carry out the transformation of the requirements.

Figure *5* shows the DMN decision requirements diagrams, which represent the decisions to carry out the biodiesel requirement.

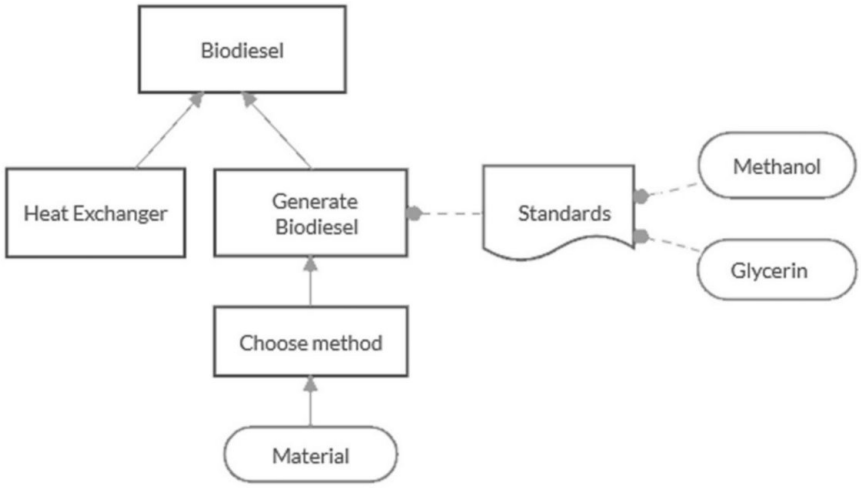

FIGURE 5: DMN model for biodiesel distiller.

DMN, Decision Model and Notation.

The SysML model generated by the transformation is shown in Figure *6*.

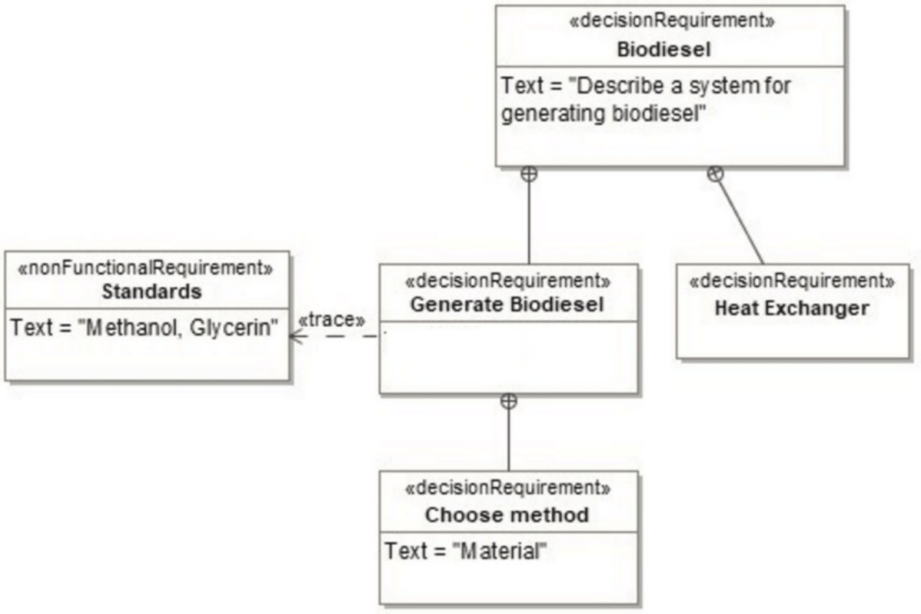

FIGURE 6: SysML model obtained from the DMN model through the transformation.

DMN, Decision Notation and Modeling; SysML, Systems Modeling Language.

The XML code corresponding to the previously shown DMN diagram is as follows:

```
<definitions id="definitions_191paqw" name="Model">
 <decision id="Decision_0luhpfj" name="Biodiesel">
 <informationRequirement id="InformationRequirement_1uu6hlz">
 <requiredDecision href="#Decision_1jnngv1" />
 </informationRequirement>
 <informationRequirement id="InformationRequirement_0vj3wzv">
 <requiredDecision href="#Decision_1esh2tr" />
 </informationRequirement>
 </decision>
 <decision id="Decision_1jnngv1" name="Heat Exchanger" />
 <decision id="Decision_1esh2tr" name="Generate Biodiesel">
 <informationRequirement id="InformationRequirement_0apuont">
 <requiredDecision href="#Decision_1yczoef" />
 </informationRequirement>
 <authorityRequirement id="AuthorityRequirement_145p8di">
 <requiredAuthority href="#KnowledgeSource_1p8fpbg" />
 </authorityRequirement>
 </decision>
 <decision id="Decision_1yczoef" name="Choose method">
 <informationRequirement id="InformationRequirement_0x87yb4">
 <requiredInput href="#InputData_1gsgvc6" />
 </informationRequirement>
 </decision>
 <inputData id="InputData_1gsgvc6" name="Material" />
 <knowledgeSource id="KnowledgeSource_1p8fpbg" name="Standards">
 <authorityRequirement id="AuthorityRequirement_0ckkq6g">
 <requiredInput href="#InputData_1hzn0nr" />
 </authorityRequirement>
 <authorityRequirement id="AuthorityRequirement_1tlmewz">
 <requiredInput href="#InputData_0yfgssr" />
 </authorityRequirement>
 </knowledgeSource>
 <inputData id="InputData_1hzn0nr" name="Methanol" />
 <inputData id="InputData_0yfgssr" name="Glycerin" />
</definitions>
```

It is worth noting that the XML used complies with the OMG standard, ensuring that this transformation is compatible with any tool following this standard.

After executing the *createModel* and *createStereotype* relationships, the following XML code corresponding to the SysML metamodel is obtained.

```
<uml:Model xmi:type='uml:Model' xmi:id='definitions_191paqw' name='Model'>
 <packagedElement xmi:type='uml:Stereotype' xmi:id='decisionRequirement_id'
name='decisionRequirement'>
 <ownedAttribute xmi:type='uml:Property' xmi:id='ppt_id_123' name='base_Element' visibility='private'>
 <type href='http://www.omg.org/spec/UML/20131001/UML.xmi#Element'/>
 </ownedAttribute>
 </packagedElement>
<packagedElement xmi:type='uml:Stereotype' xmi:id='nonFunctionalRequirement_id'
name='nonFunctionalRequirement'>
 <ownedAttribute xmi:type='uml:Property' xmi:id='ppt_id_345' name='base_Element' visibility='private' >
 <type href='http://www.omg.org/spec/UML/20131001/UML.xmi#Element'/>
 </ownedAttribute>
</packagedElement>

<stereotype name='decisionRequirement' stereotypeHREF='decisionRequirement_id'/>
<stereotype name='nonFunctionalRequirement' stereotypeHREF='nonFunctionalRequirement_id'/>
.....
```

Before generating the requirements using stereotypes, it is necessary to create the classes for the decisions and knowledge sources (KS), see *decisionToClass* and *ksToClass* relationships.

```
<packagedElement xmi:type='uml:Class' xmi:id='Decision_0luhpfj' name='Biodiesel'>
 <nestedClassifier xmi:type='uml:Class' xmi:id='Decision_1jnngv1' name='Heat Exchanger'/>
 <nestedClassifier xmi:type='uml:Class' xmi:id='Decision_1esh2tr' name='Generate Biodiesel'>
 <nestedClassifier xmi:type='uml:Class' xmi:id='Decision_1yczoef' name='Choose method'/>
 </nestedClassifier>
</packagedElement>
<packagedElement xmi:type='uml:Class' xmi:id='KnowledgeSource_1p8fpbg' name='Standards'/>
```

Once the classes are generated, they can be stereotyped as appropriate. The *decisionToDecisionRequirement* and *ksToNonDecisionRequirement* relationships are responsible for this.

```
<requirement base_Class='Decision_0luhpfj' Text='' Id=''/>
<decisionRequirement base_Class='Decision_0luhpfj'/>
<decisionRequirement base_Class='Decision_1jnngv1'/>
<requirement base_Class='Decision_1jnngv1' Text='' Id=''/>
<decisionRequirement base_Class='Decision_1esh2tr'/>
<requirement base_Class='Decision_1esh2tr' Text='' Id='' TracedTo='KnowledgeSource_1p8fpbg'/>
<decisionRequirement base_Class='Decision_1yczoef'/>
<requirement base_Class='Decision_1yczoef' Text='Material' Id=''/>
<nonFunctionalRequirement base_Element='KnowledgeSource_1p8fpbg'/>
<requirement base_Class='KnowledgeSource_1p8fpbg' Text='Methanol, Glycerin' Id=''/>

<packagedElement xmi:type='uml:Abstraction'>
 <client xmi:idref='Decision_1esh2tr'/>
 <supplier xmi:idref='KnowledgeSource_1p8fpbg'/>
 </packagedElement>
```

As observed in the generated code, there is a *packagedElement* tag with the type *Abstraction*. This tag is generated by the *generateAbstractionD* relationship, which creates the tag needed to represent the *trace* dependency. This dependency reflects the relationship between the decision and the knowledge sources, corresponding to the *AuthorityRequirement* relationships in the DMN diagram.

Discussion

As mentioned earlier, the overall graphical structure may appear similar in both diagrams; however, their XML representations differ significantly, as shown in the example presented. These differences make the construction of this transformation complex.

Each XML tag must be explicitly created through well-defined relationships, which requires a deep understanding of the underlying metamodels. For instance, creating a dependency relationship in SysML may involve generating multiple XML tags that represent elements such as the client, the supplier, and the dependency link itself. This level of detail increases the implementation effort.

The main objective of this work was to employ the use of industry standards, avoiding the creation of new notation that would imply an additional learning curve for engineers. In this sense, it seeks to avoid the development of a SysML profile with its own semantics, either through graphic representations or through specific stereotypes. Instead, it is proposed that engineers with DMN experience develop their models directly in this notation, which can then be transformed into SysML in order to facilitate the traceability of requirements. It should be noted that DMN is already adopted in the industry, and many tools were developed for its modeling, such as Cardanit, Camunda, BPMN.iO, Signavio, among others. Consequently, it is not admissible to discard existing DMN models and build or reconstruct SysML requirements models.

Conclusions

The paper has proposed the use of QVT-based bidirectional transformation technology in order to transform a decision requirements model defined in DMN standard to a SysML requirements model. An example involving the main elements has been presented. This transformation was built by previously analyzing the correspondences at the metamodel level of both languages, to use a single modeling language, and then be able to visualize the traceability of these requirements.

Establishing a correspondence between SysML and DMN simplifies the process of building models. If the engineer is familiar with SysML, they can use this language exclusively to model all requirements, including decision requirements, allowing for not only a unified requirements model but also showing the relationships between these requirements and other system requirements and modeling elements. If the engineer is knowledgeable about DMN, they can combine both notations while retaining the ability to convert diagrams from one format to another as needed.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Corina Abdelahad, Daniel Riesco, Carlos Kavka

Acquisition, analysis, or interpretation of data: Corina Abdelahad

Drafting of the manuscript: Corina Abdelahad

Critical review of the manuscript for important intellectual content: Corina Abdelahad, Daniel Riesco, Carlos Kavka

Supervision: Corina Abdelahad, Daniel Riesco, Carlos Kavka

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Model Driven Architecture (MDA). Accessed: October 10, 2025: https://www.omg.org/mda/.
2. Object Management Group (OMG). Accessed: October 10, 2025: https://www.omg.org/.
3. Bucchiarone A, Ciccozzi, Lambers L, et al.: What is the future of modeling?. IEEE Software. 2021. 38:119-127. 10.1109/ms.2020.3041522
4. Kent S: Model driven engineering. Integrated Formal Methods. IFM 2002. Lecture Notes in Computer Science. Butler M, Petre L, Sere K (ed): Springer, Berlin, Heidelberg; 2002. 286-298. 10.1007/3-540-47884-1_16
5. Kahani N, Bagherzadeh M, Cordy JR, Dingel J, Varró D: Survey and classification of model transformation tools. Software & Systems Modeling. 2018, 18:2361-2397. 10.1007/s10270-018-0665-6
6. Query/View/Transformation (QVT). Accessed: October 10, 2025: https://www.omg.org/spec/QVT/1.3/PDF.
7. MetaObject Facility (MOF). Accessed: October 10, 2025: http://www.omg.org/mof/.
8. Object Constraint Language (OCL). Accessed: October 10, 2025: https://www.omg.org/spec/OCL/2.4/PDF.
9. Buchmann T: Prompting bidirectional model transformations-the good, the bad and the ugly. Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems. 2024, 550-555. 10.1145/3652620.3687802
10. Systems Modeling Language (SysML). Accessed: October 10, 2025: https://www.omg.org/spec/SysML/1.6/PDF.

11. Hecht M, Chen J: Verification and validation of SysML models. INCOSE International Symposium. 2021, 31:599-613. 10.1002/j.2334-5837.2021.00857.x
12. Friedenthal S, Moore A, Steiner R: A Practical Guide to SysML: The Systems Modeling Language. Elsevier/Morgan Kaufmann, Waltham, MA; 2014.
13. Herber DR, Narsinghani JB, Eftekhari-Shahroudi K: Model-based structured requirements in SysML. 2022 IEEE International Systems Conference (SysCon). 2022, 1-8. 10.1109/SysCon53536.2022.9773813
14. Decision Model and Notation (DMN). Accessed: October 10, 2025: https://www.omg.org/spec/DMN/1.5.
15. Hasić F, Serral E, Snoeck M: Comparing BPMN to BPMN+ DMN for IoT process modelling: a case-based inquiry. Proceedings of the 35th Annual ACM Symposium on Applied Computing. 2020, 53-60. 10.1145/3341105.3373881
16. Balmelli L: The systems modeling language for products and systems development. The Journal of Object Technology. 2007, 6:149-149. 10.5381/jot.2007.6.6.a5
17. Wang S, Shi J, Huang Y, Yang Y: A Tool for transforming SysML state machine into Uppaal automatically. 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2023, 2471-2476. 10.1109/SMC53992.2023.10394302
18. Miao W, Zhou Y, Li B, Feng B, Du H: XMI-based conversion of SysML-STM models to C++ code. Proceedings of the 2024 16th International Conference on Computer Modeling and Simulation. 2024, 15-19. 10.1145/3686812.3686815
19. Li Y, Zhang H, Lv J, Liu Y: XMI-based SysML model transformation. 7th International Conference on Control, Robotics and Cybernetics (CRC). 2022, 74-77. 10.1109/CRC55853.2022.10041202
20. Thanh TN, Thanh NL, Thanh HHT, Thi TH: VeBPRu: A toolchain for formally verifying business processes and business rules. Proceedings of the 2023 8th International Conference on Intelligent Information Technology. 2023, 15-19. 10.1145/3591569.3591572
21. Tavantzis T, Promikyridis R, Tambouris E: Towards exploiting BPMN and DMN in public service modeling. Proceedings of the 27th Pan-Hellenic Conference on Progress in Computing and Informatics. 2023, 211-216. 10.1145/3635059.3635092
22. Shen X, Luo J, Wang H, Liu M, Dustdar S, Wang Z: A hybrid BPMN-DMN framework for secure inter-organizational processes and decisions collaboration on permissioned blockchain. arXiv. 2024, 10.48550/arXiv.2412.01196
23. Weske M: Data and decisions. Business Process Management. Weske M (ed): Springer, Berlin, Heidelberg; 2024. 267-293. 10.1007/978-3-662-69518-0_5
24. Maleki C, Gailly F: Exploring the power of triple crown process modeling in healthcare: sepsis case. Proceedings of the 17th International Joint Conference on BIOSTEC. 2024, 2:516-528. 10.5220/0012394000003657
25. Etikala V, Van Veldhoven Z, Vanthienen J: Text2Dec: Extracting decision dependencies from natural language text for automated DMN decision modelling. Business Process Management Workshops. BPM 2020. Lecture Notes in Business Information Processing. Del Río Ortega A, Leopold H, Santoro FM (ed): Springer, Cham; 2021. 397:367-379. 10.1007/978-3-030-66498-5_27
26. Unified Modeling Language. Accessed: October 10, 2025: https://www.omg.org/spec/UML/2.5.1/PDF.
27. Sommerville I: Software Engineering, Seventh Edition. Pearson Education, New Delhi, India; 2005.