Model-driven Engineering – a promising approach for developing critical software applications

Abstract: Many different approaches and frameworks exist to use Model-driven Engineering (MDE). Some of these support the integration of non-functional requirements (NFRs). This report describes these two areas separately and relates them in a detailed analysis where the advantages and disadvantages get assigned to the properties and process characteristics of critical software applications. At the end this report states four key factors that should be considered when to decide if MDE is used to develop critical software applications.

1. Introduction
This report describes the use of Model-driven Engineering (MDE) to develop critical software applications. A systematic literature review after [17] has been performed for this report. This literature review has shown, that many different approaches and frameworks exist, that support MDE. Many of these frameworks cover specific properties of critical software (explained in chapter 2) (see [1, 2, 3, 4, 7, 8, 10, 15]). Other frameworks focus on the support of non-functional requirements (NFRs) in general, which are related to the properties of critical software (explained in chapter 2) (see [5, 9, 16]). Together with these different approaches, tools have been developed to support integrating the properties of critical systems and NFRs in general into MDE (see [6, 11, 12, 13, 14, 15, 22, 23]). However the literature has shown that no standard exists to develop critical software with MDE. Much more has the review shown that a differentiated analysis is necessary to answer the question, if MDE is a promising approach for developing critical software applications? This report will try to do that.

In section 2 will be explained what critical systems are, what properties are important and how critical systems are usually developed. Section 3 will explain MDE in general, especially how the framework from the Object Management Group (OMG) looks like, what the general goals of MDE are, together with the advantages and disadvantages of MDE. The last section will relate the properties of critical software applications to the advantages and disadvantages of MDE and provides a detailed discussion to answer the question above.
2. Critical software applications

To describe the properties of critical software applications the term critical software application has to be defined. After Sommerville a critical software application is a ‘technical or socio technical system that people or businesses depend on’ [18, p. 44]. Sommerville defines further, that a system failure can result in serious problems and significant losses. Depending on the type of critical software application (safety-critical, mission-critical or business-critical) the loss can be the loss of life, environmental damage, failure of goal-directed activity or high costs for a business [18].

The most important property of critical software applications is dependability [18]. Dependability covers a range of properties which will be explained in the following paragraph. The first property is availability, which describes the probability of a system to be up and running and to deliver the requested services [18, 19]. The second property is reliability, which describes the probability of a system to function correctly and deliver the services as expected [18, 20]. Reliability includes correctness (ensure the system services cover the functional and non-functional requirements), precision (ensure information in detail as necessary is delivered) and timeliness (ensure that information is delivered when needed) [18, 20]. The third property, safety, describes the probability of a system to cause damage to people or its environment [18, 21]. Security, fourth property, judges on how likely it is, that a system is assassinated to accidental or deliberate intrusions [19]. Security includes integrity (ensure that the software and data it contains are not damaged) and confidentiality (ensure that information is only delivered to authorised users) [18, 19]. The fifth property is repairability, which considers how fast a system can be repaired after a failure has been discovered [18, 23]. Maintainability, which is the sixth property, describes how fast and easy software can be adopted to satisfy new requirements [18]. Another property, number seventh, is survivability and describes the ability of a system to deliver services while under attack [18]. All these properties are NFRs and are important to prevent system failure and information loss [18]. Usually critical systems are developed using well-tried techniques rather than newer techniques [18]. In the following paragraph one process will be explained to develop critical software applications.

Critical software applications are usually developed using dependable processes [18]. Dependable processes consist of five characteristics: documentable, standardised, auditable, diverse, robust [18]. These characteristics have to be considered within the process activities. The process of developing critical software applications consists of seven activities [18]:

All content is copyright their rightful owner Marc Werfs. Republication of think creative 3.0 content should be properly credited (and linked to) as the source, including URL: thinkcreative30.wordpress.com.
• Requirements inspection: discover problems with the system specification
• Requirements management: keeping track of changes and trace these
• Model checking: automatically analysing system models to ensure consistency
• Design and code inspections: based on checklist of common faults
• Static analysis: program is analysed in detail to find erroneous conditions
• Test planning and management: ensure complete test coverage and traceability
• Configuration management: keep track of versions of software and its components

With these activities the majority of mistakes that occur during software development (e. g. misunderstanding of specification, over-complex programs and use of error prone constructs) can be avoided [18]. However the use of MDE changes the way software is developed and is needs to be checked if the properties and activities explained above are existent in MDE.

3. Model-driven Engineering

Similar to the concept of critical software applications, the term Model-driven Engineering has to be defined. The literature review (explained in section 1) has revealed that a lot of different definitions and terms exist. Most common are the use of the terms Model-driven Engineering and Model-driven Development (MDD)\(^1\). Within MDE models are used to specify, simulate, verify, test and generate the system that has to be built [25]. This definition is consistent with the one from the OMG, which is an organisation who sets standards [31]. However MDE is only a general definition of the topic. To use MDE for developing software a more concrete view has to be chosen. As well, a lot of different views and approaches exist [26]. This report will concentrate on the Model-driven approach (MDA) because it is the most prevalent [26]. MDA separates the development of software into three different stages, therefore it uses different models [25]. First of all a Platform Independent Model (PIM) has to be developed [25, 26]. This kind of model concentrates on the functionality based on the business-level. It is independent from the underlying technology [26]. After that, the PIM is transformed into a Platform Specific Model (PSM) through a Model-to-Model transformation [25]. Multiple PSM can share on PIM [25]. After that the PSM is transformed into executable code through Model-to-Text transformation [25]. Thereafter a complete MDA application

---

\(^1\)This essay will use MDE and MDD as synonyms.
consists of one PIM, one or more PSMs and one concrete implementation for each platform the developer supports [26]. Figure 1 shows the process.

The major goal of the use of models for developing software is to raise the level of abstraction [26, 27]. Thereafter the model aims to improve the communication between different participants (no matter if customer or developer) and makes the whole developing process more understandable [26, 29]. In general it makes software development more systematic and formal and aims to achieve adaptability (because models are less bound to the underlying technology), portability and reuseability (because models are less sensitive to the chosen technology and are closer to the problem domain) [27, 28, 29]. In addition it simplifies the development process, reduces the amount of developer effort (because it is an automated process) and the complexity of software [26, 29]. As explained above, the use of MDE aims to construct systems automatically. This helps to prevent failures made by developers (e.g. typos) and makes it possible to run simulations easily before the software deployment [30].

However MDE has not only advantages but also disadvantages. First of all it creates redundancy because in MDE multiple representations of artefacts are used [26]. As explained above, different models are used for different stages with a different level of abstraction. This means, if one model changes, all the other models have to be changed as well. This can create a huge developer effort [26]. The next problem is closely connected to the previous one. Between different models and abstractions for a software application exist relationships and many of these relationships are interrelated [26]. The round-trip problem occur when an interrelated attribute changes and that change affects some or all of its related attributes [26]. Above, the reduction of complexity was stated as an advantage for MDE; however not all the researchers agree. [26] has the opinion that models move the complexity elsewhere and not
always reduce it. Thereafter spurious complexity can be created which is difficult to solve [26]. For MDE different models are used (see above) with eventually different model technologies. Especially for bigger development teams this can become a problem. The different development teams have to be an expert in all the different model technologies used and not only in the one they used [26]. Thereafter more expertise is required at each construction site [26]. In addition they have to know the different transformation notations (see above) [26]. A last disadvantage is that with a higher abstraction more choices exist on how the executable code will be realised [26]. However this doesn’t have to be a disadvantage in call cases, as we will see in the next chapter, where the theory of MDE will be applied in terms of developing critical software applications.

4. Discussion
The second section of this report stated, that for critical software applications especially non-functional requirements are important. This means for a connection between MDE and critical software applications the support of MDE for NFRs has to be analysed. Furthermore the transformations of the different models are important because the priority of NFRs have to be chosen [25]. To justify if MDE is a promising approach for developing critical software applications, the advantages and disadvantages (section 3) have to be connected to the properties of critical systems (section 2) and related to NFRs. Table 1 and 2 show the result of this connection. The following paragraph will explain how this connection was achieved.

The first column shows the properties of critical systems (Table 1) respectively the characteristics of dependable processes (Table 2). The facts from this column are based on the research explained in section 2. The second column shows the advantages from section 3 assigned to the rows in column 1. The third column shows tools and frameworks (from the area of MDE) that help to satisfy the needs of the rows in column 1. The fourth column shows the disadvantages from section 3 as assigned to the rows in column 1. The sources at the end of every cell reveal the academic paper where the advantages and disadvantages were assigned to the properties of critical software applications.

Table 1 - Properties of critical software applications

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependability</td>
<td>advantages</td>
<td>Tools</td>
</tr>
<tr>
<td>availability</td>
<td>simplifying [26], abstract</td>
<td>Generalized Stochastic Petri Nets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>moving complexity [26], depending on the approach</td>
</tr>
</tbody>
</table>

precision and formal [27], accurate [27], predictive [27] LTSA [6] depending on the approach very high or low [25]
timeliness reusability [29], amount of developer effort [26] dependency on the approach very high till low [25]
security

Table 2 - Characteristics of dependable processes

<table>
<thead>
<tr>
<th>characteristics</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentable</td>
<td>simplifying [26]</td>
<td>round-trip problem [26]</td>
</tr>
<tr>
<td>standardised</td>
<td>more systematic and formal [27]</td>
<td>the higher the abstraction, the more choices for code [26]</td>
</tr>
<tr>
<td>auditable</td>
<td>understandable [6]</td>
<td>more expertise required [26]</td>
</tr>
<tr>
<td>diverse</td>
<td>the higher the abstraction, the more choices for code [26]</td>
<td>more expertise required [26]</td>
</tr>
<tr>
<td>robust</td>
<td>simulation [30]</td>
<td>the higher the abstraction, the more choices for code [26]</td>
</tr>
</tbody>
</table>

Table 1 and 2 show that for nearly every advantage a disadvantage exists, as well. In the majority the advantages and disadvantages are connected to the underlying MDE approach (including the transformations and modelling software) (e.g. cells C-2, C-5, C-11, E14). In some cases it is not even clear if the advantages of MDE can be clearly assigned to the properties of critical software applications because researchers do not always agree (e.g. cells A-1 (for abstraction), A-8, A-9, C-1, D-16, E14). In one case a disadvantage of MDE can be seen as an advantage in another circumstance (cells E-14, D-16). Based on the results shown in these tables it is not possible to give a clear answer to the proposed question in section 1. Based on the argumentation in this report the use of MDE within the context of critical software application development is connected to the following key factors:
• The use of MDE approach, because different models have different properties and therefore different advantages and disadvantages (Table 1 shows the advantages and disadvantages for a wide range of approaches)

• The use of MDE framework (especially transformations), because it depends on the framework if it is possible to choose the priority of NFRs

• The use of software tools that support MDE, because the available tools don’t support all properties equally

• The priority of properties of critical software applications, because not all properties are well supported by MDE (e.g. safety and repairability)

Thereafter I come to the conclusion that MDE can be considered for the development of critical software applications but a human being should always check the transformations between the different models and the executable code. Last but not least a human being should perform the last tests and validations because the life of other human beings depends on the software. This means that the process activities outlined in section 2 should persist and be performed by human beings.

5. Conclusion
This report has related the properties and processes of critical software application development to the advantages and disadvantages of different MDE approaches, including frameworks and tools. The results of this report show that the question formulated in section can’t be answered with yes or no. It highly depends on the four factors outlined at the end of section 4.

Word Count: 2100
Bibliography


