Data Validation Constraints in MDE

Alessandro Rossini[1], Adrian Rutle[2], Federico Mancini[1], Dag Hovland[1],
Khalid A. Mughal[1], Yngve Lamo[2] and Uwe Wolter[1]
[1]Department of Informatics, University of Bergen, Norway
[2]Faculty of Engineering, Bergen University College, Norway

ABSTRACT

Software security encompasses all the measures taken to ensure confidentiality, integrity and availability in software systems. In present-day software development, security is often just an afterthought rather than part of the software development life-cycle. In order to reveal potential security flaws before a software system is actually implemented, security aspects should be taken into account starting from the early phases of the development. With Model-Driven Engineering gaining momentum in both academia and industry, an interesting challenge is the specification of security constraints within software models. In this paper we focus on data validation – the process of ensuring that a system operates on correct and meaningful data – in the MDE context. Our contribution is a formal approach to the specification of data validation constraints which involve multiple structural features. In addition, we map constraints specified at the model level to Java annotations which are then transformed to executable tests by an existing data validation framework.

1. INTRODUCTION

Software systems have made in-roads into all walks of society and government. Violating the confidentiality, integrity and availability of these systems can therefore lead to negative impact on economy, politics and health. Software security aims at ensuring that these properties are not compromised. In present-day software development, security is often neglected because of budget constraints, time to market values and lack of skills. Typically security concerns are considered far too late when the system is already nearing deployment. This is clearly insufficient since security aspects should be taken into account starting from the early phases of the development [2, 3] in order to reveal potential security flaws before a software system is actually implemented.

Model-Driven Engineering (MDE) is a new trend in software engineering which aims at improving productivity, quality, and cost-effectiveness of software by considering models as first-class entities of the software development process. In this regard, an interesting challenge is the specification of security constraints within software models. In this paper we focus on data validation – the process of ensuring that a system operates on correct and meaningful data – in the MDE context. The lack of proper data validation is listed as the most prevalent cause of software vulnerabilities by the OWASP [4].

Our contribution is a formal approach to the specification of data validation constraints which can involve not only single structural features, but also the relationship between multiple, interdependent structural features. Such relationships cannot be modelled with traditional modelling languages such as UML. It is instead possible to do this in the Diagram Predicate Framework (DPF) [6, 5, 7], which, for this reason, constitutes the formal underpinning of our approach to modelling.

Our work is related to the SHIP Validator [1] – a Java based framework which enables the validation of multiple interdependent properties of Java objects. Since the SHIP Validator uses Java annotations to associate validation tests and properties, it is possible to map constraints specified at the model level to the corresponding annotations. These annotations are in turn transformed to executable tests at run-time by the SHIP Validator.

The remainder of the paper is structured as follows. Section 2 provides an introduction to DPF and the theoretical concepts used in the paper. Section 3 outlines a running example and provides the formalisation of the concepts of data validation. In Section 4 the current research in security within MDE is summarised. Finally, in Section 5 some concluding remarks and ideas for future work are outlined.

2. DPF

In this section, a brief introduction to DPF is provided and the theoretical concepts used in the paper are defined. In DPF, models are represented by *diagrammatic specifications*. A diagrammatic specification is a structure which consist of a graph *decorated* by a set of *constraints*. The graph represents the structure of the model. Predicates from a predefined *diagrammatic signature* are used to impose constraints to the graph. In the remainder of the paper, we use the terms "model" and "diagrammatic specification" interchangeably. The formal definitions are as follows:

Definition 1. A (diagrammatic predicate) signature $\Sigma := (\Pi, \alpha)$ consists of a collection of predicate symbols Π with a mapping α that assigns a graph to each predicate symbol $p \in \Pi$. Accordingly, $\alpha(p)$ is called the *arity* of the predicate symbol p.

Definition 2. Given a signature $\Sigma = (\Pi, \alpha)$, a constraint (p, δ) on a graph G is given by a predicate symbol p and a graph homomorphism $\delta : \alpha(p) \to G$.

Definition 3. Given a signature $\Sigma = (\Pi, \alpha)$, a Σ-specification $S := (G^S, C^S)$ is given by a graph G^S and a set C^S of constraints (p, δ) on G^S with $p \in \Pi$.

3. DPF AND DATA VALIDATION

In this section we introduce a running example taken from [1] and show how the concepts introduced in the previous section are applied to the problem of data validation.

Example 1. Consider international money transfers. *IBAN* (International Bank Account Number) is the standard for identifying bank accounts internationally. Some countries have not adopted this standard and, for money transfer to these countries, a special *clearing code* is needed in combination with the plain *account number*. *BIC* (Bank Identifier Code) is the standard for identifying banks globally. Therefore, a form for international money transfers of a hypothetical Internet bank will contain (at least) the fields `bic`, `iban`, `account` and `clearingCode`. Moreover, the transfers systems will have to satisfy the following data validation constraints:

1. the BIC code of the beneficiary's bank is required
2. either the IBAN or both clearing code and the account number are required

Table 1 shows a signature $\Sigma_{sec} = (\Pi, \alpha)$ containing predicates used to specify the data validation constraints above in diagrammatic specifications. The first column of the table shows the names of the predicates. The second and the third columns show the arities of predicates and a possible visualisation of the corresponding constraints, respectively. In the fourth column, the intended semantics of each constraint is specified.

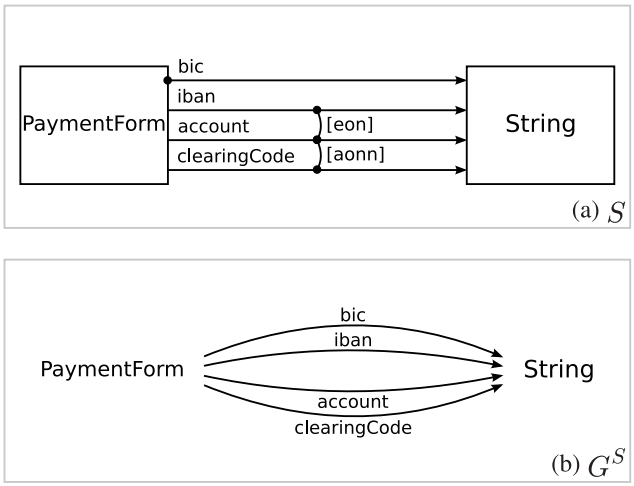


Figure 1: A Σ_{sec}-specification $S = (G^S, C^S)$

Fig. 1a shows a Σ_{sec}-specification $S = (G^S, C^S)$ using predicates from Σ_{sec}. S specifies the structural model of the form for international money transfers. The payment form is represented by the node `PaymentForm` while the input fields are represented by the arrows `bic`, `iban`, `account` and `clearingCode`. Fig. 1b presents the underlying graph G^S of S; i.e. the graph of S without any constraints.

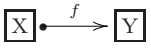
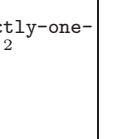
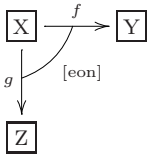
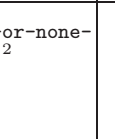
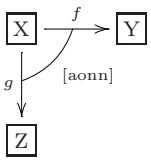
In S, the first data validation constraint is forced by the constraint $(\texttt{[required]}, \delta_1)$ on the arrow `bic`, i.e. $\delta_1 : (1 \xrightarrow{x} 2) \mapsto (PaymentForm \xrightarrow{bic} String)$. Moreover, the second data validation is forced in S by two constraints: $(\texttt{[exactly-one-null]}, \delta_2)$ on the arrows `iban` and `account` together with $(\texttt{[all-or-none-null]}, \delta_3)$ on the arrows `account` and `clearingCode`.

The model of the form for international money transfers can be transformed to the following Java class tagged by Java annotations compatible with the SHIP Validator.

```
1  public class PaymentForm {
2
3    String bic;
4    String iban;
5    String account;
6    String clearingCode;
7
8    @Required
9    public String getBic() {
10     return bic;
11   }
12
13   @ExactlyOneNull
14   @NotRequired
15   public String getIban() {
16     return iban;
17   }
18
19   @ExactlyOneNull
20   @AllOrNoneNull
21   @NotRequired
22   public String getAccount() {
23     return account;
24   }
25
26   @AllOrNoneNull
27   @NotRequired
28   public String getClearingCode() {
29     return clearingCode;
30   }
31
32 }
```

These Java annotations are in turn transformed into executable tests by the SHIP Validator. Note that the idea of using annotations to hide the actual validation code and, at the same time, tag the properties to be tested, allow the constraints to be easily integrated into existing code. Besides, the validation aspects of the system remain well separated from the application aspects. This separation of concerns facilitates the transformation of the diagrammatic constraints into actual working code.

Table 1: The signature Σ_{sec}

Π	α	Proposed visualisation	Intended semantics
`[required]`			$\forall x \in X : \lvert f(x)\rvert \geq 1$
`[exactly-one-null]`[2]			$\forall x \in X : (\lvert f(x)\rvert \geq 1$ and $\lvert g(x)\rvert = 0)$ or $(\lvert f(x)\rvert = 0$ and $\lvert g(x)\rvert \geq 1)$
`[all-or-none-null]`[2]			$\forall x \in X : (\lvert f(x)\rvert \geq 1$ and $\lvert g(x)\rvert \geq 1)$ or $(\lvert f(x)\rvert = 0$ and $\lvert g(x)\rvert = 0)$

4. RELATED WORK

In [2], the author describes how to use the UML extension mechanisms to include safety-requirements in UML models which are then analysed for satisfaction of the requirements. The approach can thus be used to encapsulate safety engineering knowledge. This approach is very general and covers several aspects of software security. However, despite the usage of UML extension mechanisms, the approach is not sufficient to express data validation constraints involving multiple structural features at the model level. In this respect, an advantage of DPF is that it allows such constraints to be expressed in a diagrammatic fashion.

5. CONCLUSION AND FUTURE WORK

In this paper, some of the key aspects of data validation in MDE have been illustrated. We have adopted DPF as a formal approach to the specification of data validation constraints in software models. Moreover, we have shown how these constraints can be mapped to Java annotations which are transformed to executable tests.

In the future, we plan to introduce a reasoning system for the analysis of predicate dependencies and to define a logic for this analysis.

6. REFERENCES

[1] D. Hovland, F. Mancini, and K. Mughal. The SHIP Validator: An Annotation-Based Content-Validation Framework for Java Applications. Technical Report 389, Department of Informatics, University of Bergen, Norway, September 2009.

[2] J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.

[3] G. McGraw. *Software Security: Building Security in*. Addison-Wesley Professional, 2006.

[4] OWASP. *Top Ten Project*. http://www.owasp.org.

[5] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In *FASE 2009*, volume 5503 of *LNCS*, pages 64–78. Springer, 2009.

[6] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Diagrammatic Formalisation of MOF-Based Modelling Languages. In *TOOLS 2009*, volume 33 of *LNBIP*, pages 37–56. Springer, 2009.

[7] A. Rutle, U. Wolter, and Y. Lamo. A Diagrammatic Approach to Model Transformations. In *EATIS 2008*, pages 1–8. ACM, 2008.