DOI: https://doi.org/10.52651/sam.a.2021.2.15-27

ANALYSIS OF CURRENT TRENDS IN THE DEVELOPMENT OF DSLS AND THE POSSIBILITY OF USING THEM IN THE FIELD OF INFORMATION SECURITY

Patrik HARNOŠ, Ľubomír DEDERA

Abstract: Use of an appropriate DSL can significantly reduce development time. This is due to the fact that DSLs are limited to the use of terms relating to the explicit domain, which makes them much easier for programmers to understand and learn. Despite these and other advantages of DSLs over GPLs, programmers will only exceptionally turn to DSLs in development process. Therefore, in this article, we will look closer on what DSLs are, when it is appropriate to use them in a project and when not. In the last part of this article, we will focus on the possibilities of using DSLs in the field of information security.

Keywords: Computer languages; DSL; Language workbenches; Alternative Computation Model; Information security.

1 INTRODUCTION

Domain-specific languages (DSLs) have been part of computer science for several decades, but like with other academic research in computer science, their potential is not yet used too much in practice. Nevertheless, many DSL experts are of the opinion that in these days when it is more relevant to focus on programming time than on processing time, the future of programming languages lies in the use of DSLs. The productivity of software development is just one of the advantages of using DSLs, although one of the most important ones. As for some DSLs, there are even efforts to completely omit the programmer from the software development process. However, in most cases of DSLs, one needs to have at least basic programming skills along with knowledge of a particular domain to create useful software. Therefore, in the process of developing a software project, the programmer's communication with domain experts is very important. Insufficient communication with domain experts or customers is the most common source of project failures during development [1]. Understandable code created by using a domain-appropriate DSL is an invaluable advantage, at this stage of project development, over general-purpose languages (GPLs).

The acronym DSL has gained popularity only with the advent of Domain-specific modeling [2]. You can find many papers written about DSLs, and probably every programmer has already worked with some DSLs. In spite of that, most programmers usually do not decide to implement their own DSL in the project anyway. However, GPLs do not always support the computational models of all projects. For example, when you want to change behaviour at runtime, or express aspects of behaviour in a more self-sense. This has led to an increasing use of XML configuration files, even in project where a custom syntax would be more readable and not harder to do. We may also encounter the use of parsers when a fluent interface in their regular language would be a lot less work. Martin Fowler, the author of "Domain-Specific Languages", came up with a hypothesis that this fact is caused by knowledge gap. This hypothesis speaks to the fact that skilled programmers have little awareness of the benefits of DSLs and how to implement them. Therefore, in this article, we will briefly summarize what DSLs are.

Domain Specific Language (DSL) is a computer programming language with limited expressiveness that is targeted to a particular kind of problem in focused domain [1]. So obviously DSLs are not Turing-complete. This is the main difference from a general-purpose language that is aimed at any kind of software problem. But this definition of DSL, like most things in software, is not sufficiently defining, and therefore there are a number of cases where determining whether it is DSL or not is the reason for disagreement.

For this reason, the experts have formed four main elements identifying whether it is DSL:

A. DSL is used by people, so it is as simple as possible to understand, but it is still executable by computer.
B. DSL should be as smooth as possible, that is, it consists not only of separate expressions but also of composite ones.
C. DSL is created for use in a particular aspect of the system in its domain and at the same time should be as simple as possible to understand. For these reasons, DSL should contain only the necessary minimum functions to cover domain requirements.
D. DSL is effective only when it is focused only on a small domain.

2 TYPES OF DSLs

DSLs are mostly divided into two main categories: external DSLs, internal DSLs, but DSLs created by language workbenches can be recognized as a third category [1].

An external DSL is a domain-specific language represented in a separate language. An external DSL has a custom syntax, but it may follow the syntax of another representation such as XML. Host application of external DSL uses text parsing techniques to parse script written in this language. External DSLs usually come with tools like IDEs that

are designed to support specific functions of the language. This can result in a much-improved user and developer experience. Static analyses, code completion, visualizations, debuggers, simulators and all kinds of other niceties can be provided. These features can improve the productivity of users exponentially and they also make it easier for new team members to become productive. Examples of external DSLs that you have probably come across include regular expressions, SQL, and XML configuration files, but there are plenty of other less known external DSLs. Good examples are textual DSLs, which verbally describe graphical output, like graph or diagram. This group includes, for example, DOT and PlantUML.

In contrast to external DSLs, internal DSLs do not include tool support like debuggers and testing tools in its language design. Internal DSLs just define specific way how to use host GPL in order to use it effectively in specific domain. So, a script in an internal DSL is valid also in its host GPL. The most common way to accomplish this is by implementing a library or framework and then manipulate this framework through command-query API calls, or alternatively by a DSL. In this view, a DSL is a more understandable front-end to a library. That is why internal DSLs are sometimes called fluent API. From language design perspective, API defines the vocabulary of the abstraction, whereas an internal DSL adds a grammar [1]. So, the goal of developers of an internal DSL is mostly to create a language with understandable syntax for a domain expert, however, without language tool support, this language cannot increase productivity exponentially and it will probably be unpopular among developers.

A language workbench is a term coined by Martin Fowler in 2005. This term describes an environment designed to help people create new DSLs. A language workbench will typically include high-quality tools to support the definition, reuse and composition of domain-specific languages. To define the DSLs, the workbench supports:

1. Defining the schema for a Semantic Model for the language
2. Defining one or more rich editing environments for the language
3. Defining the behavioural semantics for the language, through a mix of interpretation and code generation.

All of this is valuable, but the truly interesting aspect of language workbenches is that they allow a DSL designer to go beyond the traditional text-based source editing to different forms of language. Anyway, they did not become mainstream, but developers continue to be interested in them and many of them reckon that they can change programming landscape significantly.

To be completely precise, internal and external DSLs can also be divided into two groups, based on how they are combined with other languages: to fragmentary and stand-alone DSLs [1]. To put it simply, fragmentary DSLs are modules of code inside a GPL and such code is hardly understandable for nonprogrammers. An example of an external fragmentary DSL is SQL request to take action on database inside some GPL. A good example of an internal fragmentary DSL is unit testing. On the other hand, stand-alone DSLs contain files consisting exclusively of a DSL, and therefore there are understandable for domain experts.

3 WHEN IT IS APPROPRIATE TO USE DSL

After the previous sections, where we introduced you to various DSL types, you can see advantages of using DSLs in your project. However, this does not mean that it is suitable to use one in every project. The first obvious obstacle to the deployment of DSLs in a project is the cost of building. This obstacle is enlarged by the fact that developers are not used to building them and therefore the deployment of DSLs may require learning new techniques. And that can be time consuming. However, if we can spend some time on development of one or more DSLs in a project, we are likely to benefit from this investment of time later in project, in its easier testing and longer lifecycle. A good example is HTML, well known domain specific language. Technologies like web browsers have been evolving from the beginnings of HTML, but this DSL itself, representing domain logic, is still widely used. This is ensured by the characteristics of well-designed DSLs, and the resulting benefits. But you can create well-designed DSLs only if you can clearly define its scope. So, if we cannot exactly specify the domain of DSLs in a project, their use would not be effective and therefore we should not use them in such a project.

Now we have an idea when it is not appropriate to develop DSLs, so let's summarize why you should consider using DSLs in your project by naming their benefits. Firstly, a DSL helps knowledge workers doing their job, because it captures errors. Next, but maybe the most obvious advantage, is better communication with domain experts, but only some stand-alone DSLs fulfil this aspect (for example, regular expressions do not make communication easier). You can streamline communication with client and domain experts by adding some kind of visualization to your DSLs language support portfolio. Great examples of how it can be done are DOT and PlantUML.

DOT is a language for describing graphs, thanks to which we can verbally define a graph and then, using a program working with that language, generate the graph in a graphical form. Generating graphs using comprehensible commands is practical, but DSLs can bring more added value to the project. For example, PlantUML, which allows you to shape different types of UML diagrams by textual based commands, can provide a valuable tool to support

whole development process, e.g. for easier versioning. UML can be modified by anyone without special tools and PlantUML can be big help during discussions with client and domain experts by allowing immediate changes. For a better idea, we have prepared an example of the code in PlantUML:

```
@startuml
title <b>Use of a DSL</b>\n Creation of standardised documents
left to right direction
skinparam packageStyle rect
skinparam shadowing false

actor :Document creator: as user
actor :Template creator: as admin
note left of user: Inserts data into a template.
note left of admin: Inserts templates into the database.

rectangle semanticModel{
(Add Data) as addData
(Add Template) as addTemplate
(Generate Document) as generate
(View Database) as database

addData.> generate: includes
user --> addData
user --> database
admin --> addTemplate
admin --> database
}
@enduml
```

We can edit this code during a discussion and, by using a generator (for example, online at http://www.plantuml.com), we can interpret changes almost immediately. You can see output of our PlantUML code, interpreted by generator, in Figure 1:

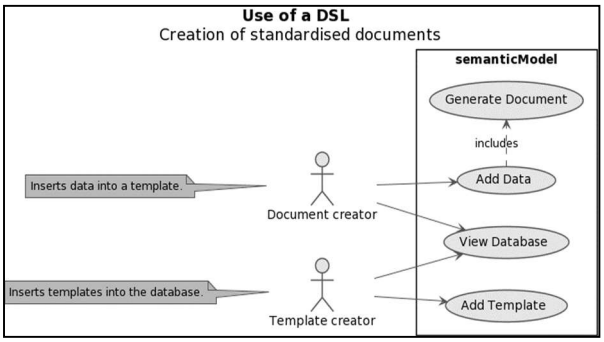


Fig. 1 Use case diagram
Source: author.

Another aspect of using a DSL in any project is the increase of productivity across whole project lifecycle, but mostly during specification (feedback cycles are much shorter), testing (verifying can be done directly by domain experts), using, modifying and maintaining. There are many well designed DSLs that can greatly facilitate work within their given scope. But in this article, as an example, we present only one, LaTeX.

Name LaTeX denotes a high-quality typesetting system; it includes unique DSL and other features designed for creating technical and scientific documentation. LaTeX runs on top of Donald E. Knuth's TeX typesetting system. LaTeX is available as free software [3]. LaTeX belongs to a group of languages that receive the specified sequence of commands on the input and interpret them as a separate artifact — a document on output. This is our example code in LaTeX:

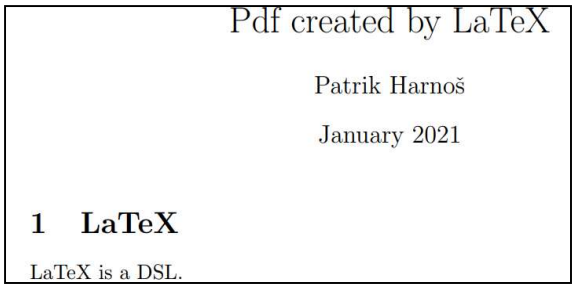
```
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{Pdf created by LaTeX}
\author{Patrik Harnoš}
\date{January 2021}

\begin{document}
\maketitle
\section{LaTeX}
LaTeX is a DSL.
\end{document}
```

We can easily convert this code to a Pdf document by the “pdflatex.exe” application, which is a part of LaTeX system.

```
D:\Phd\DSL\Clanky\LaTeX\instalacia\miktex\bin\x64>pdflatex LaTeX.txt
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (MiKTeX 20.12)
entering extended mode
(LaTeX.txt
LaTeX2e <2020-10-01> patch level 2
L3 programming layer <2020-12-07> xparse <2020-03-03>
(D:\Phd\DSL\Clanky\LaTeX\instalacia\tex/latex/base\article.cls
Document Class: article 2020/04/10 v1.4m Standard LaTeX document class
(D:\Phd\DSL\Clanky\LaTeX\instalacia\tex/latex/base\size10.clo))
(D:\Phd\DSL\Clanky\LaTeX\instalacia\tex/latex/base\inputenc.sty)
(D:\Phd\DSL\Clanky\LaTeX\instalacia\tex/latex/l3backend\l3backend-pdftex.def)
No file LaTeX.aux.
[1{C:/Users/Pato/AppData/Local/MiKTeX/pdftex/config/pdftex.map}] (LaTeX.aux) )
<D:/Phd/DSL/Clanky/LaTeX/instalacia/fonts/type1/public/amsfonts/cm/cmbx12.pfb><
D:/Phd/DSL/Clanky/LaTeX/instalacia/fonts/type1/public/amsfonts/cm/cmr10.pfb><D:
/Phd/DSL/Clanky/LaTeX/instalacia/fonts/type1/public/amsfonts/cm/cmr12.pfb><D:/P
hd/DSL/Clanky/LaTeX/instalacia/fonts/type1/public/amsfonts/cm/cmr17.pfb>
Output written on LaTeX.pdf (1 page, 42462 bytes).
Transcript written on LaTeX.log.
pdflatex: major issue: So far, you have not checked for MiKTeX updates.
```

Fig. 2 Example of using the "pdflatex.exe"
Source: author.

You can see a part of the Pdf document created by “pdflatex.exe” in Figure 3. LaTeX is based on the idea that the author of the document handles only the text of the article, while the formatting is taken care of by the document developers. Similar idea of DSLs helping users effectively check and add data to systems can be used in various domains including information security.


Pdf created by LaTeX

Patrik Harnoš

January 2021

1 LaTeX

LaTeX is a DSL.

Fig. 3 Pdf created from example code
Source: author.

4 DSLs IN THE FIELD OF INFORMATION SECURITY

The field of information security follows the trend of using DSLs in whole computer science. We can say that there are just a few of them, mostly XML-based, some of them are DSLs only in a narrower point of view, but new ones are being developed. For example, identity access management and security auditing, as subfields of the information security, have been enriched by several narrow-scoped languages. So, we picked some DSLs to describe in this chapter, and we divide them into these two information security subfields.

4.1 Identity and access management

Most organizations still use legacy systems with inbuilt authorization logic. Sometimes, one organization contains a large number of information systems and applications and each system or application uses their own process for authorization. Today, authorization has become more complex because users within organizations and outside need access to shared data and need to collaborate efficiently. Therefore, it is a challenging task to manage such legacy systems, custom authorization systems. However, XACML offers a solution to this problem, as a standard which is ratified by the OASIS standardisation organization [4]. Before we start with XACML, let's take a look at its domain and access control management. There are four main types of access control:

1. **Access Control Lists** – Oldest and most basic form of access control. This type is easy to implement due to the use of maps, but it was primarily adopted for use in operating systems. So, it can become difficult to manage larger user bases.
2. **Role-base Access Control (RBAC**) – This is a static permission model which provides access control to authorized users based on their role. This type of approach reduces management overhead and therefore it is used by the majority of enterprises with more than 500 users.
3. **Attribute-based Access Control (ABAC)** – This is a new, more flexible and fine-grained approach in comparison with RBAC. Access rights are granted to users through the use of policies that combine attributes together. Because of it there is no need to know the user prior to granting access.
4. **Policy-based Access Control** – This is the most complex form of access control and it addresses the requirement of larger enterprises to have more uniform access control mechanism for the large amount of organization units. This involves specifying policies unambiguously using XACML and using authorized attribute sources in the enterprise.

So, now we know that XACML is used in access control management, mostly for specifying policies in Policy-based access control approach. But there is much more behind the label XACML. XACML is very popular as a fine-grained authorization method among the community. XACML was introduced as a standard by OASIS in 2003 and it defines an access control policy language, request/response language (protocol), and reference architecture. The policy language is used to express access control policies (who can do what, when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses). The reference architecture proposes a standard for deployment of necessary software modules within an infrastructure to allow efficient enforcement of policies.

The XACML reference architecture illustrated below, in Figure 4, consists of the following four blocks:

1. **Policy Administration Point (PAP):** This entity allows the administrator to create, update and change a policy or policy set, by defining them in XACML policy language. These defined policies are stored in policy store. PAP makes policies and policy sets available to the PDP by Policy Retrieval Point (PRP). These policies or policy sets represent the complete policy for a specified target.
2. **Policy Enforcement Point (PEP):** The access requester sends a request for access to the PEP and this point performs access control by enforcing authorization decisions. This is the entity that sends the XACML request to the PDP and receives back an authorization decision (response).
3. **Policy Decision Point (PDP):** This entity evaluates policies against access requests sent by PEP. To provide the decisions, PDP may also need to query a PIP to gather descriptive attributes about the user or any other missing attribute in the request.
4. **Policy Information Point (PIP**): If there are missing attributes in the XACML request that is sent by PEP, PIP would find them for the PDP to evaluate the policy. PIP acts as a source of attribute values.

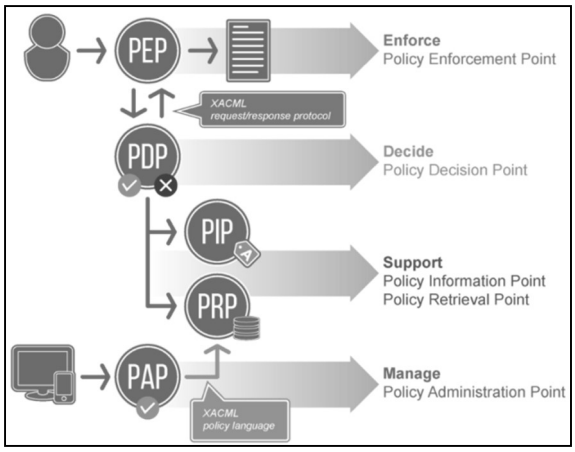


Fig. 4 Diagram of XACML Architecture
Source: [18].

The most important part of XACML, for this article, is XACML policy language structure and syntax. XACML policy language is XML-based because it is an industry standard and experts of this domain are familiar with it. XACML supports policy-based access control, based on attributes. XACML categorizes attributes in policies into four groups: subject (user, workstation, etc.), resource (server, database, etc.), action (read, write, etc.) and environment (SAML, J2SE, etc.). But from XACML 3.0, custom categories are also supported. The elementary unit of policy is a Rule.

A *Rule* is a single statement using Boolean logic, defined by *RuleId*, that specifies the individual rule in the policy. Each Rule has a particular effect on an access request (deny or permit the access). Each *Rule* is composed of a *Target* and a *Condition*.

A *Target* is an XACML component that defines categories applied on a policy. A *Target* consists of *AnyOf* and *AllOf* components and each *AllOf* consists of *Match*. Each *Match* contains only one particular category to be matched with the request.

A *Condition* specifies the applicability of the rule [5].

XACML policy language syntax, with components mentioned above, is demonstrated on the following example policy:

```
<Policy xmlns="urn:..." PolicyId="hr-admin-
access"...>
<Target>
<AnyOf>
<AllOf>
<Match MatchId="...function:string-equal">
<AttributeValue DataType="...#string">hradmin
</AttributeValue>
<AttributeDesignator AttributeId=".../role"
        Category="...:subject-category:access-
subject"
        DataType="...#string"
MustBePresent="true"/>
</Match>
</AllOf>
</AnyOf>
</Target>
<Rule Effect="Permit" RuleId="Rule-1">
<Target>
<AnyOf>
<AllOf>
<Match MatchId="...:function:string-equal">
<AttributeValue DataType="...#string">/Employee
</AttributeValue>
<AttributeDesignator
AttributeId="...:resource:resource-id"
        Category="...:attribute-category:resource"
        DataType="...#string"
MustBePresent="true"/>
</Match>
</AllOf>
</AnyOf>
<AnyOf>
<AllOf>
<Match MatchId="...:function:string-equal">
<AttributeValue DataType="...#string">GET
</AttributeValue>
<AttributeDesignator AttributeId="...:action:action-
id"
        Category="...:attribute-category:action"
        DataType="...#string"
MustBePresent="true"/>
</Match>
</AllOf>
</AnyOf>
</Target>
</Rule>
<Rule Effect="Deny" RuleId="Deny-Rule"/>
</Policy>
```

This reduced example of the policy permits only GET access to the resources /Employee by everyone with role hradmin. You can write XACML policies with any XML editor, but there are more friendly solutions how to create and publish similar, but mostly more complex, policies to Policy Administration Point (PAP). One of them is WSO2 Identity Server.

WSO2 Identity Server provides secure identity management for enterprise web applications, services, and APIs by managing identity and entitlements of the users by including the role-based access control (RBAC) convention, fine-grained policy-based access control, and Single-Sign-On (SSO) bridging. The Identity Server enables enterprise architects and developers to reduce identity provisioning time and guarantee secure online interactions. WSO2 Identity Server uses XACML as a tool for controlling access to applications. The Identity Server supports XACML 3.0, which is based on Balana XACML implementation. The source code, distribution and documentation are available for free and it is released under Apache Software License Version 2.0, one of the most business-

friendly licenses available today. The XACML engine of the WSO2 Identity Server has two major components Policy Administration Point (PAP) and Policy Decision Point (PDP) [4].

Usage of XACML in WSO2 is demonstrated in the figures below. The Identity Server enables you to create or edit policies in PAP, not only by using XML editor, but with five other creation methods. You can use Simple, Basic and Standard Policy Editors or Policy Set Editor. Importing Existing Policy is also available. As you can see in Figure 5, thanks to Simple Policy Editor, creating a policy is a matter of filling in its name, description, and defining which role can perform what actions on which resources.

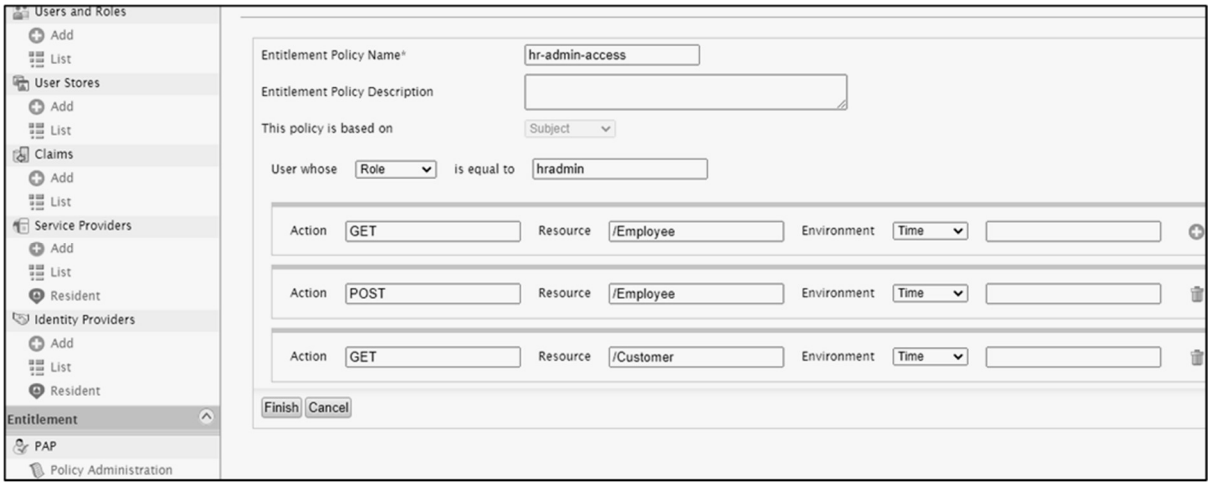


Fig. 5 Creation of example policy in the Simple Policy Editor
Source: author.

Fig. 6 Part of the XML code of example policy created in Simple Policy Editor
Source: author.

Fig. 6 shows a part of the example policy, created using Simple Policy Editor, displayed in XML. As you can see, this policy specifies rules for the role *hradmin*. Before we can publish and test this policy, we need to create user with this specific role.

As you see in Fig. 7, this is not a problem because the Identity Server enables you to create, maintain and terminate user accounts along with user identities across multiple systems including Cloud applications. Testing of a policy is also very intuitive by creating resource request in WSO2 request editor. You just need to fill XACML attributes in request. Then, WSO2 Carbon (componentized middleware platform) will evaluate request according to policy and informs you about the response (deny, permit or not applicable when attributes in request are not specified in the policy).

Fig. 7 User with *hradmin* rule
Source: author.

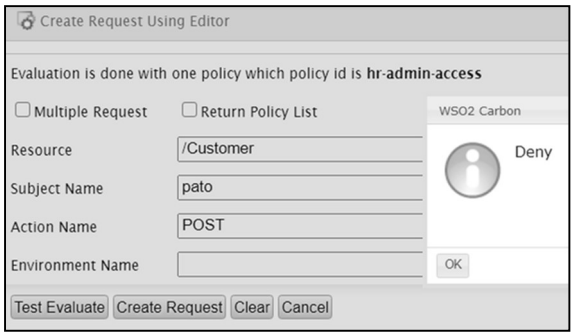


Fig. 8 Denied response to the request
Source: author.

Fig. 8 above shows a denied response sent by identity server as an answer to the request. Response is denied because user *pato* in *hradmin* role is not allowed to post in */Customer* resource based on our tested policy. On the other hand, request from user *pato* to post in */Employee* resource is in accordance with our example policy, so it is permitted as you can see in Fig. 9.

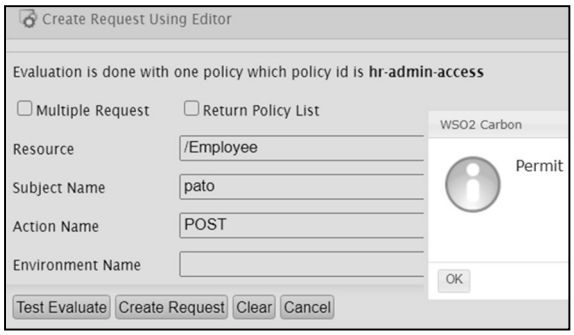


Fig. 9 Permitted response to the request
Source: author.

So, at the end we can say that XACML is very viable and successful DSL because of robustness of XACML and support from identity management environments like WSO2. The main advantage of XACML is its extensibility. A good example of that is in the GeoXACML. It defines a geo-specific extension standardised by Open Geospatial Consortium (OGC), which allows location-based authorisation where the access control policy can contain geographic primitives (points, lines, polygons etc.) as well as operations on these (contains, within, distance etc.) within the authorisation policy. This is quite a unique feature of XACML. XACML from version 3.0 and GeoXACML meet changing requirements from developers who prefer the JSON format over XML by enabling encoding based on JSON. GeoXACML standard defines geospatial data interchange format based on JSON, with name GeoJSON. There are more languages that can be included in the group of DSLs in the field of access control, but we are not describing them in detail here. It is because they are similar to XACML in terms of the syntax (they are all XML or JSON based) and also because they are DSLs only from a certain point of view.

SAML (Security Assertion Markup Language) is an open XML security standard for exchanging authentication and authorization data between an identity provider and a service provider and it is also an XML-based markup language for access-control decisions statements.

Privacy and Identity Management for Europe (PRIME) project has developed a privacy-aware access control policy language and a data handling policy language able to protect user's personal information and to provide a framework that can be smoothly integrated with current architectures and online services. PRIME defines three types of privacy policies, Access control, Release and Data handling policies, to fully achieve privacy-aware access control system [6].

A P3P Preference Exchange Language (APPEL) is a policy language which enables users to specify their privacy preferences. Language (APPEL), similarly to PRIME languages, proposes a XML-based language for regulating secondary use of data and provides restrictions on the recipients. Main difference with PRIME languages is that users in fact can only accept the server privacy practices or stop the transaction, APPEL does not support definition of policies based on attributes of the recipients and protection against releases to third parties.

Rei is a policy language based in OWL-Lite (OWL-Lite uses only some features of Web Ontology Language which is used to define how applications process content of information) that allows policies to be specified as constraints over allowable and obligated actions on resources in the environment [7].

Web Services Policy is policy layer standard and a simple language that has four elements - Policy, All, ExactlyOne and PolicyReference - and two attributes - wsp:Optional and wsp:Ignorable. It offers mechanisms to represent combinations of capabilities and requirements, of policies and to associate them with Web service metadata constructs [8].

ODRL (Open Digital Rights Language) is a licencing standard and a policy expression language that provides flexible and interoperable mechanisms to support transparent and innovative

use of digital content in publishing, distributing and consuming of digital media across all sectors and communities.

4.2 Security auditing and system monitoring

Computer security has become increasingly important and security auditing is one of the techniques to detect vulnerabilities and schedule a procedure to reduce them before possible attack. The goal of every company, using some information system, should be to properly implement security guidance, as target computers need to be hardened and continuously monitored during their lifecycle. Organizations can achieve this by implementing their security policy in the form of automation protocol SCAP with its XML based languages OVAL, OCIL, XCCDF and ARF.

Security Content Automation Protocol (SCAP) is a collection of standards managed by National Institute of Standards and Technology (NIST). It was created to enable users to perform the security audit on multiple remote systems from a single, centralized environment and also to provide a standardized approach for checking system security configuration settings, monitoring systems for signs of compromise and automatically verifying the presence of patches. Users can achieve it by using SCAP Workbench, GUI tool that serves as an SCAP scanner and provides functionality for SCAP content. SCAP includes languages [9]:

- **OVAL**: The Open Vulnerability and Assessment Language is an XML-based community standard and language for making logical assertions about the state of an endpoint system by a three-step assessment process: representing configuration for testing; analysing the system for the presence of vulnerability, configuration, patch state, etc.; and reporting the results which can be transferred across the entire spectrum of information security tools and services.
- **XCCDF**: A language to express, organize, and manage security guidance that references OVAL.
- **OCIL**: The Open Checklist Interactive Language defines a framework to provide a standard way of querying a human user. It represents a set of questions to a user and interprets responses to these questions. Although the OCIL specification was developed for use with IT security checklists, other possible use cases include research surveys, academic course exams, and instructional walkthroughs.
- **ARF**: Asset Reporting Format is a language to express the transport format of information about assets, and the relationships between assets and reports.

The Insider Threat Prediction and Specification Language (ITPSL) is an external XML based DSL created to provide a systemic way to describe insider threats and misuse incidents. This is an early language compiler prototype and its underlying insider threat monitoring framework are not fully released but they have huge potential because various information security surveys and case studies indicate the importance and manifestation of the insider threat problem. ITPSL can be described as a specialized language that is able to encode system level data made by legitimate user actions, in order to create the process of misuse threat prediction. ITPSL can be of help to domain experts like the security analyst, as well as the IT administrator in charge of system operation and security issues. Both of these types of domain experts should be able to examine insider misuse incidents and express insider misuse scenarios by using the language semantic. The process of doing so starts with the security analyst writing the description of a particular insider misuse scenario, using the ITPSL semantics. The signature is validated by a compiler that translates the signature directives to query commands and uses a logging infrastructure (ITPSL use LUARM audit engine, relational model and SGL interface to audit logs), in order to examine whether the specified criteria exist in the system. The Evaluated Potential Threat is score indicating the likelihood of threat occurring according to a given detected conditions [10].

Panoptis is an anomaly detection system for security administrators, based on Unix process-accounting records, which consists of a single program that reads accounting records and updates profile tables, optionally reporting cases that fall outside the existing profiles. Its arguments are a DSL-based configuration file that directs the program operation, the database to update, the interval to operate upon, and an optional list of process accounting files. Panoptis is quite unique for using DSL-based instead of traditional XML-based configuration files which are used more often for real time setting of a system. Panoptis was published in 2002, so it could be a milestone in the new trend of using custom made DLSs in configuration files.

PowerShell is still used in managing operation systems from Microsoft, so to be more complex we can mention a PowerShell module that was implemented as a DSLs or even Pester, testing and mocking framework for PowerShell, which allows to create a mini-DSL for writing your tests. Pester uses a simple set of functions: Describe, Context, It, Should and Mock. So, your test can be readable and fluent [11]:

```
It 'Earth is the third planet in our Solar System' {
    $allPlanets = Get-Planet
    $allPlanets[2].Name | Should -Be 'Earth'
}
```

As a conclusion of this chapter, we can see that in the field of information security there are mostly XML-based DSLs. These languages meet the DSL properties we mentioned in the introduction, but like the original XML, universal data exchange languages, they are more machine-readable rather than smooth languages. But nowadays we can create smoother DSLs thanks to language workbenches.

5 LANGUAGE WORKBENCH

The definition of language workbench is provided in the second chapter of this article, but without mentioning examples. There are many language workbenches under active development, both in industry and academia. Notable examples include Jetbrains MPS, MetaEdit+, Xtext, Rascal, Spoofax, SugarJ, Melange, Cedalion, Epsilon, EMFText, Intentional Software, Whole Platform, DrRacket, Eco, Ensō, MontiCore, and others. Anyway, there is a lot to choose from, but the choice should depend on the type of DSL that will be built. It is caused by the fact that some language workbenches are suitable for textual languages, others for graphical, but most exciting and powerful are projectional editors.

5.1 Textual DSLs

Xtext is an open-source framework and a solid solution for developing programming languages and DSLs. Unlike standard parser generators, Xtext generates full infrastructure, including parser, linker, typechecker, compiler as well as editing support for Eclipse, any editor that supports the Language Server Protocol (for example IntelliJ) or a web browser. But for advanced features you have to be familiar with Eclipse technology, for example Xtext produces parser using Eclipse Modeling Framework (EMF). You can see simple Hello World grammar definition from Eclipse examples in the code below [12].

```
    grammar org.example….
generate domainmodel
"http://www.example.org/..."
    Model:
    greetings+=Greeting*;
    Greeting:
    'Hello' name=ID '!';
```

textX is very similar alternative because it was inspired by Xtext. Major difference between textX and Xtext is that textX is a Python framework, it does not use EMF, it does not generate code and editor support like Xtext because it uses metaprogramming power of Python to define classes in memory [13].

Spoofax is an open-source language workbench for creating textual DSLs. It can generate parsers, type checkers, compilers and interpreters. But it is more academic than an industrial-grade language workbench aimed to separate definition and implementation of a language. Spoofax can be used inside Eclipse or IntelliJ. It is based on a set of hi-level declarative DSLs (meta-languages) aimed at various concerns of textual DSL creation process. With this approach, designers are not distracted by language implementation details. These meta-languages abstract language implementation and focus on the language design. For example, Spoofax users are addressing the issue 'what is the syntax of my language?', instead of 'how do I implement a parser for my language?' [14].

5.2 Graphical DSLs

In comparison with textual languages, graphical languages are more user friendly for domain experts in many cases. But on the other hand, they require building specific editors to be used and they are less flexible than textual languages. So, they are less frequent and even the tools to build graphical they are not fully tuned and easy to use. GMF is a good example of this statement.

Graphical Modeling Framework (GMF) can help you when you need extreme flexibility to build your very own graphical editor. The core of GMF is GMF Runtime, a Java based framework to run graphical editors inside Eclipse. It uses EMF, similarly to Xtext, to define the structure of your data and then GMF permits to specify how the different elements are represented, how their connections are displayed and so on. It provides a highly customizable way to render any model elements with several kinds of graphical shapes by filling out the form and editing the details of each element in a separate panel. An example of a GMF editor is displayed in Fig. 10 below.

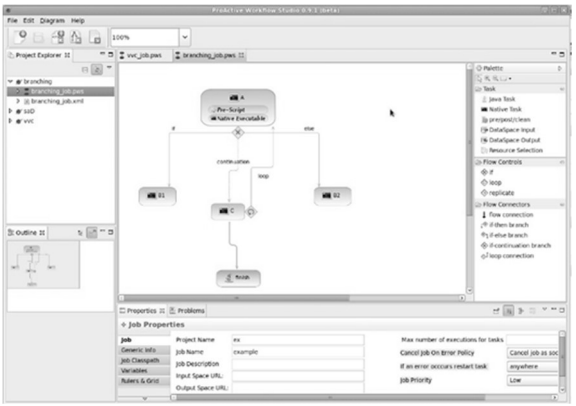


Fig. 10 Example usage of GMF
Source: [19].

GMF is flexible and powerful with potential but without solid documentation and supporting community, therefore it offers a painful experience for the language designer. For this reason, there are tools like Eclipse Eugenia and Eclipse Sirius, built on top of GMF to make better user experience. Sirius is more complex and it uses model introspection instead

of code generation approach. It is decently supported, and it offers reasonable usability.

MetaEdit+ is a commercial solution among graphical language workbenches. It provides a simple metamodeling language and tool for designing language concepts, their properties, associated rules, symbols, checking reports and generators. MetaEdit+ makes modeling tool development fast, intuitive, and cost-effective by allowing you to get started straight away with the reuse of language components from extensive library [15].

5.3 Projectional editors

Projectional editing is a superset of the graphical editing, so you can define graphical languages using a projectional editor like Jetbrains MPS, but projectional editors are much more flexible than typical graphical languages. Projectional editors offer combination of different notations and support all sorts of representation you need for your case. With projectional editing, the abstract representation is the core definition of the system and projectional editor shows a projection of the content stored on file, as you can see in Fig. 11.

It is a similar solution to graphical workbenches, but different in comparison with a text editor which stores user changes directly to a disk. When a user interacts with projection, in the projectional editor, he can edit text, tables or diagrams, but the editor translates those interactions and they will be stored in a different format like XML or even binary. As a result, while thinking about your editable representations you actively think about how an editor works with them. This leads to different ideas than you would get from a purely passive editable representation such as text. The point is that you can work with those files only inside their special editor [1].

JetBrains MPS (Meta Programming System developed by JetBrains) is an extremely powerful tool for designing domain-specific languages and it is the most mature projectional editor available now. It uses projectional editing which allows overcoming the limits of language parsers, and building DSL editors, such as ones with tables and diagrams. It implements language-oriented programming. MPS combines an environment for language definition, a language workbench, and an Integrated Development Environment (IDE) for such languages. MPS can be useful to build families of interoperable languages with advanced tooling to describe the logic of your problems, to define tests and documentation. This platform has built in all sorts of simulators, debuggers and tools to analyse code coverage. In Fig. 12 below there is a demonstration of MPSs sample named ChemMastery from chemistry knowledge worker perspective. Model situation is that a domain expert wants to create and maintain models expressed in the provided DSL.

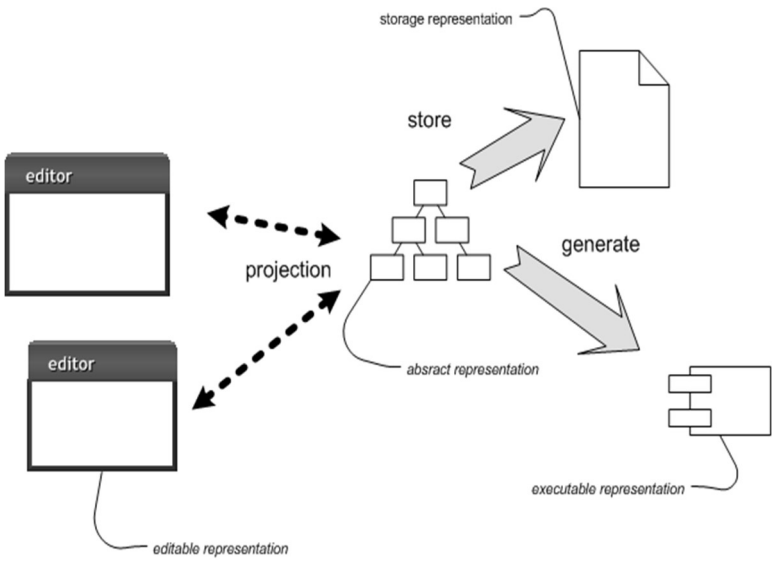


Fig. 11 Manipulating representations with a language workbench
Source: [20].

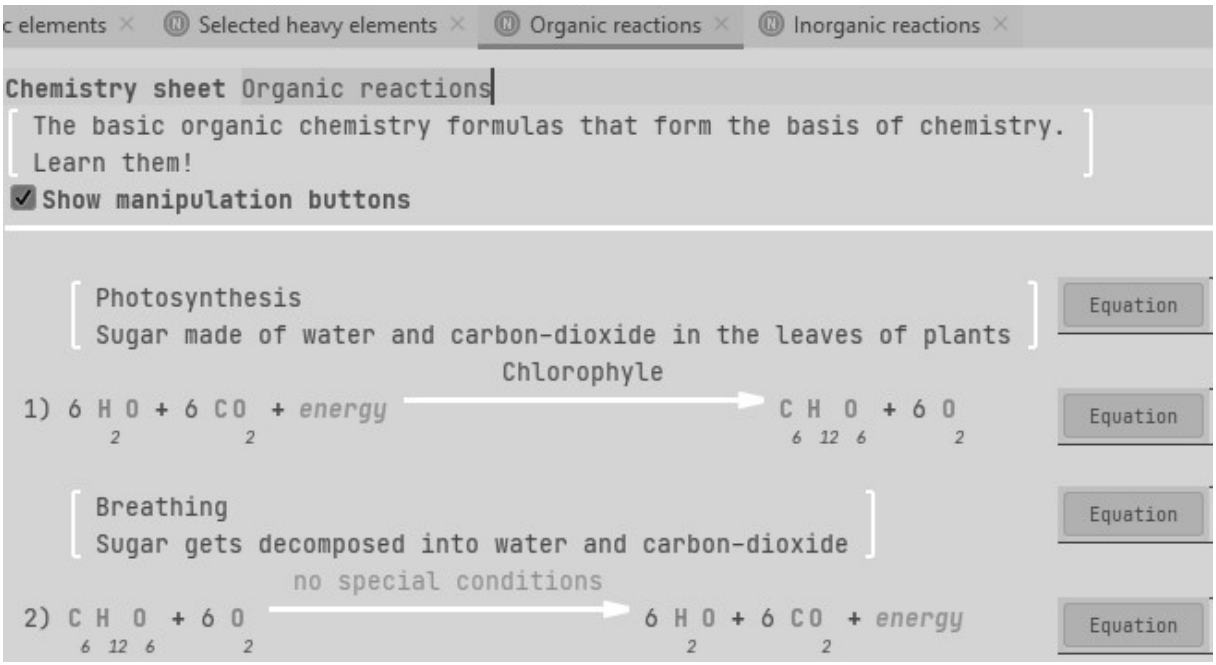


Fig. 12 MPSs sample named ChemMastery from user perspective
Source: author.

Whole Platform is another overlooked but good language workbench supporting existing formats and adapting advanced language-engineering approaches. This editor also offers the idea of Pattern Language, it means that you can take the model and define its variability points that could be filled with values from another model.

There are more projectional editors like Intentional Platform or Modeling SDK for Visual Studio, but International Platform is not publicly available yet. Modeling SDK offers valuable alternative to MPS for language developers who prefer working with Microsoft Visual Studio. DSL designed in this kit can be distributed as part of a Visual Studio Integration Extension (VSIX) package, so users can work with the DSL in Visual Studio [16]. The user interface of DSL Tools solution in Visual Studio will resemble the following Fig. 13:

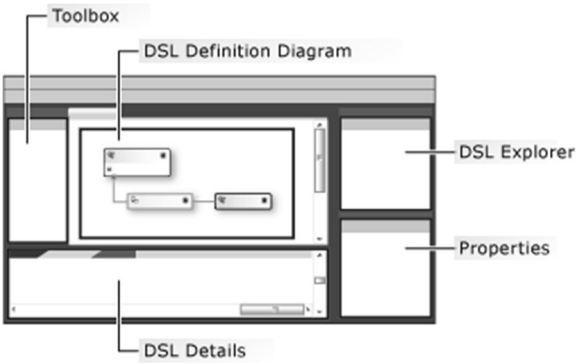


Fig. 13 Domain-Specific Language Tools User Interface
Source: [16].

You can also use Windows Forms to display the state of a DSL model, instead of using a DSL diagram. [17].

6 CONCLUSION

One of the goals of this work was to get ourselves familiar with the DSLs and their potential in information security, and with language workbenches. This is accomplished and we can focus on creating an experimental DSL for the field of information security. Based on the information written in this article to date, we know that new DSLs have been developed in this area of computer science in recent years, but the majority of them are XML-based. We also know that if we want to create an efficient and popular DSL, it should be as readable and simple as possible, but at the same time it must support all common tasks performed by domain experts.

Our challenge is to create a DSL that would allow knowledge workers to effectively control and add data to the system based on a template set up by the organization. Initial use case diagram of this DSL is in Figure 1. After writing this article, we have a solid idea of the capabilities of various types of language workbenches and we can conclude that we will probably use one of the projectional editors, such as JetBrains MPS, to create a DSL.

References

[1] FOWLER, M. and R. PARSONS. *Domain-Specific Languages.* Massachusetts: Addison-Wesley, 2010. ISBN-10: 0-321-71294-3.

[2] RAJA, A. and D. LAKSHMANAN. Domain Specific Languages. In: *International Journal of Computer Applications* (0975 - 8887), Vol. 1 – No. 21, 2010. Available at: https://doi.org/10.5120/37-640

[3] *LaTeX – A document preparation system.* [Online]. [accessed 20. January 2021]. Available at: https://www.latex-project.org/

[4] *WSO2 Identity Server Documentation - Access Control and Entitlement Management.* [Online]. [accessed 30. January 2021]. Available at: https://is.docs.wso2.com/en/latest/get-started/access-control-and-entitlement-management/#introducing-xacml

[5] RAMLI, C. D. P. K., H. R. NIELSON and F. NIELSON. *The Logic of XACML – Extended.* 2011, [Online]. [accessed 30. January 2021]. Available at: https://export.arxiv.org/pdf/1110.3706

[6] *PRIME, Privacy-aware Access Control Policies.* [Online]. [accessed 1. February 2021]. Available at: https://doi.org/10.1007/978-3-642-27739-9_827-2

[7] KAGAL, L. *Rei: A Policy Language for the Me-Centric Project.* Palo Alto: HP Laboratories, 2002, [Online]. [accessed 1. February 2021]. Available at: https://www.hpl.hp.com/techreports/2002/HPL-2002-270.pdf

[8] *Web Services Policy 1.5 – Primer.* W3C Working Group Note, 2007, [Online]. [accessed 1. February 2021]. Available at: https://www.w3.org/TR/ws-policy-primer/#introduction

[9] PRICE II, R. and M. PREISLER. *Practical OpenSCAP – Security Standard Compliance and Reporting.* Red Hat Summit, 2016. [Online]. [accessed 1. February 2021]. Available at: https://www.redhat.com/files/summit/session-assets/2016/SL45190-practical-openscap_security-standard-compliance-and-reporting.pdf

[10] MAGKLARAS, G. and S. FURNELL. *The Insider Threat Prediction and Specification Language.* INC, 2012. [Online]. [accessed 1. February 2021]. Available at: https://www.semanticscholar.org/paper/The-Insider-Threat-Prediction-and-Specification-Magklaras-Furnell/4588ae6687100750954592c9a7b59dd1d7df0cf1?p2df

[11] *Pester – Quick Start.* Pester Team, 2019. [Online]. [accessed 1. February 2021]. Available at: https://pester.dev/docs/quick-start/#what-is-pester

[12] *Xtext – documentation*. Eclipse, 2015. [Online]. [accessed 1. February 2021]. Available at: https://www.eclipse.org/Xtext/documentation/index.html

[13] DEJANOVIĆ, I., R. VADERNA, G. MILOSAVLJEVIĆ and Ž. VUKOVIĆ. *TextX: A Python tool for Domain-Specific Languages implementation. ScienceDirect, 2017.* [Online]. [accessed 1. February 2021]. Available at: https://doi.org/10.1016/j.knosys.2016.10.023

[14] *Documentation — Spoofax 2.5.13 documentation*. Spoofax, 2016. [Online]. [accessed 1. February 2021]. Available at: https://spoofax.readthedocs.io/en/latest/source/dev/doc.html

[15] *MetaEdit+ Domain-Specific Modeling tools.* MetaCase. [Online]. [accessed 1. February 2021]. Available at: https://www.metacase.com/products.html

[16] *Getting Started with Domain-Specific Languages - Visual Studio*. Microsoft Docs, 2016. [Online]. [accessed 1. February 2021]. Available at: https://doi.org/10.1007/978-1-4842-4382-4_1

[17] *Create a Windows Forms-Based Domain-Specific Language - Visual Studio.* Microsoft Docs, 2016. [Online]. [accessed 1. February 2021]. Available at: https://docs.microsoft.com/en-us/visualstudio/modeling/creating-a-windows-forms-based-domain-specific-language?view=vs-2019

[18] *100 % Pure XACML*. Axiomatics, 2013. [Online]. [accessed 1. February 2021]. Available at: https://www.axiomatics.com/100-pure-xacml/

[19] *Let's solve once for all the Eclipse GMF copy-paste problem and then forget about it.* Wordpress.com - esalagea, 2011. [Online]. [accessed 1. February 2021]. Available at: https://esalagea.files.wordpress.com/2011/04/workflowstudioview1.jpg

[20] FOWLER, M. *ProjectionalEditing*. ThoughtWorks, 2008. [Online]. [accessed 1. February 2021]. Available at: https://martinfowler.com/bliki/ProjectionalEditing.html

Dipl. Eng. Patrik **HARNOŠ**
Armed Forces Academy of General M. R. Štefánik
Department of Informatics
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: harnos.p@gmail.com

Assoc. Prof. RNDr. Ľubomír **DEDERA**, PhD.
Armed Forces Academy of General M. R. Štefánik
Department of Informatics
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: lubomir.dedera@aos.sk

Patrik Harnoš is a System engineer at the Ministry of Defence in Bratislava. In 2018 he graduated at the Armed Forces Academy in Liptovský Mikuláš with his thesis focused on Android application development. He is currently a PhD student researching DSLs in the field of Information Security.

Ľubomír Dedera works as an Associate Professor at the Department of Informatics, Armed Forces Academy in Liptovský Mikuláš. He graduated (RNDr.) from the Faculty of Mathematics and Physics, Comenius University in Bratislava in 1990. He received a PhD. degree in Artificial Intelligence from the Military Academy in Liptovský Mikuláš in 1997. His research interests include computer languages, computer security and artificial intelligence.