

Functional Programming Way to Interact with Software Attacks and Vulnerabilities

Violeta Damjanovic

Knowledge-based Information Systems
Salzburg Research, Jakob Haringer Strasse 5/III
5020 Salzburg, Austria
violeta.damjanovic@salzburgresearch.at

Dragan Djuric

Department of Software Engineering
University of Belgrade, Jove Ilica 154
11000 Belgrade, Serbia
dragan@dragandjuric.com

Abstract—This paper proposes using functional programming style in a way to respond to detection of and interaction with the software attacks and vulnerabilities. Additionally, our approach considers involving Description Logics, as a basis for the use of the Semantic Web and meta-programming to produce executable ontologies and to enable semantic reasoning over behavior and interaction with software attacks and vulnerabilities. Accordingly, we introduce Magic Potion, a recently defined Domain Specific meta-Language that uses Modeling Spaces framework to study heterogeneous modeling and meta-modeling problems inspired by Model Driven Architecture. As an example of formalism for modeling software attacks and vulnerabilities, we explore Attack Tree, which provides a formal methodology for analyzing the security of the system. Based on Attack Tree, which is herein specified for a particular problem of dealing with known attacks and vulnerabilities of the security layer of the Wireless Application Protocol, and which is particularly built on top of Magic Potion specification, we define our specific Domain Specific Language that we call Attack Tree Domain Specific Language. It is envisioned as a tool for modeling and interacting with software attacks and vulnerabilities.

Keywords—Software vulnerability; Functional Programming; Domain Specific Language; Description Logics; Model Driven Architecture; Attack Trees

I. INTRODUCTION

This paper explores how Functional Programming (FP) approach can be used together with Domain Specific Languages (DSL) and Description Logics (DLs) to support detection of, and interaction with software attacks and vulnerabilities. We additionally propose using Model Driven Architecture (MDA) approach to model compatibility of software attack and vulnerability according to the Semantic Web and meta-programming requests.

The central motivational problem that we address in this paper is multiple:

- to improve expressiveness of software attack and vulnerability models and base their detection on such a models;
- to semantically describe identified vulnerabilities and their interactiveness;

- to provide a specification framework for detecting and interacting with software attack and vulnerability;
- to provide a user-friendly language for building executable domain models dealing with attack and vulnerability.

To respond to the above problem, we ground our approach on Magic Potion, a recently proposed Domain Specific meta-Language that is based on DLs and inspired by MDA and FP paradigm [1]. In this paper, we explore how Magic Potion can support effective modeling, detection and interaction with software attack and vulnerability.

The paper is organized as follows. The introduction section gives a short overview of the paper and the motivational problem. Section 2 discusses related work in software attack and vulnerability, their detection and prevention method, continuing with the related work around DSL, DL, and FP that support our approach. Section 3 presents Magic Potion, a novel, still evolving user-friendly DSL that is based in DLs and implemented in a modern FP language, known as Clojure¹. Section 4 describes a motivating example problem for modeling attacks and vulnerabilities by using Magic Potion approach. At first, this section describes Attack Tree in general, and then reports on the Wireless Transport Layer Security (WTLS) Attack Tree that is followed by the correspondent attack scenario. This scenario is inspired by [2]. Section 5 presents the Attack Tree DSL, which is built on top of the Magic Potion specification framework. Section 6 validates the proposed approach at the domain level. Section 7 gives the conclusion remarks and shows current benefits and weaknesses in use of Magic Potion to model and interact with software attack and vulnerability.

II. RELATED WORK

This section discusses the related work around several diverse domains that we're bringing together through Magic Potion approach; these domains are as follows: software attack and vulnerability; FP, DSL, Semantic Web and DLs.

A. Software Attack and Vulnerability

A software vulnerability is malicious or non-malicious fault introduced during development phases (requirements,

¹ <http://clojure.org/>

analysis, design or configuration) of the system or in the way it is used, that could be exploited by an attacker to create intrusion and obtain some privileges in the system for own benefits [3]. Similar to vulnerability, an attack is a malicious external activity aimed to intentionally violate one or more security properties of the system [2].

In order to understand vulnerabilities, the creation of models that contain the set of conditions leading to attacks and vulnerabilities, is considered as very helpful for interaction with them, even for their prevention. Attacks and vulnerabilities can be modeled in many different ways, using Petri Nets, UML notations, or some of the following modeling formalisms [3]: Vulnerability Cause Graph (VCG) [4], Security Goal Indicator Trees (SGIT) [5], Vulnerability Inspection Diagram (VID), Security Activity Graph (SAG), and Attack Trees [6] that provide a formal methodology for describing and analyzing the security of systems, based on varying attacks. In this paper, we propose the use of FP style, which is influenced by DSL and DLs, to serve as a tool for detecting of and interacting with attacks and vulnerabilities.

B. Functional Programming

FP is programming paradigm with the following characteristics [7, 8]:

- treats computation as the evaluation of the mathematical functions, and avoids state and mutable data,
- functions have the same value regardless of their context,
- functions do not have any effects on memory,
- functions are not influenced too strongly by the architecture of the computer.

The well known FP languages are as follows: Erlang [9], Haskell, Scala [10]. Clojure² is a modern dialect of Lisp programming language. It insists on pure functional paradigm that is combined with a strong support for parallelization through immutable persistent data structures paired with software transaction memory. It is hosted on Java Virtual Machine (JVM); influenced by Erlang and Haskell languages. Below, we list the main features of the Clojure language [11]:

- Higher-order functions;
- Macros for meta-programming;
- Immutability: once created data structures can't change;
- Lazy evaluation and infinite sequences;
- Concurrency and the multi-core future;
- Software Transactional Memory (STM): mutating a mutable reference, that holds an immutable value, can only be done inside a transaction;
- Agent: it is a language construct that makes it easy to run code asynchronously on a separate thread.

C. Domain Specific Languages

DSL is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain [12]. Recently, they have become more popular due to the rise of

Model Driven Engineering (MDE) in which many examples of DSL may be found, e.g.:

- Object Constraint Language (OCL) that describes rules that apply to UML;
- Query/View/Transformation (QVT) that is today's standard for model transformation defined by the Object Management Group (OMG).

DSL can be implemented as a new language or by extending a given base language. It can be additionally used in aspect-oriented programming, functional programming, and more.

D. Semantic Web and Description Logics

One of the promises of ontologies is to establish semantic interoperability between agents and the applications on the Semantic Web [13], as well as to add a further representation and inference layer on top of the Web's current layers [14]. Tim Berners-Lee, the inventor of the WWW, defined three distinct levels of a *functional* architecture of Semantic Web [15]. DLs are formalisms for representing knowledge and one of the main theoretical cornerstones of ontologies and the Semantic Web. The main constructs in DLs theory are *atomic concepts*, as unary predicates, and *atomic roles* as binary predicates on the given domain. The complex concepts and roles are defined over atomic ones using logical constructors like negation, intersection, union, etc. A great advantage of DLs is availability of reasoning mechanisms usually based on tableau calculi.

III. MAGIC POTION: THE PROPOSED FUNCTIONAL PROGRAMMING APPROACH

This section discusses three main characteristics of Magic Potion approach such as: (a) it is inspired by MDA, (b) it is based on DLs, and (c) it is inspired by FP. The main motivation for combining FP and DSL as representatives of the *declarative programming* style, on one hand, and DLs on the other hand, have been seen in the following:

- expressiveness of knowledge,
- better support for reasoning about the behavior of knowledge that continuously transform from one state to the other,
- conciseness and reusability for various purposes,
- easiness of use, and
- cost reduction for education of the DSL users.

A. Magic Potion is Inspired by Model Driven Architecture

Magic Potion [1], as a DSL for creating more specific DSL, uses Modeling Spaces [16] that is a framework to study heterogeneous modeling and meta-modeling problems inspired by MDA [17]. MDA is based on four-layer meta-modeling architecture that encompasses the following layers:

- (M3) meta-metamodel layer,
- (M2) metamodel layer,
- (M1) model layer, and
- (M0) the real-world layer.

² <http://clojure.org/>

Magic Potion is additionally grounded on Clojure, a FP language that represents a modern dialect of the Lisp programming language. Hence, we consider Clojure at M3 layer, meta-metamodel layer, as Figure 1 shows. Clojure defines all metamodels, even Clojure itself. Thus, Magic Potion is created as a Clojure program at M2 layer, using Clojure’s built-in function, macros and data structures. Then, we create new models at M1 layer that represent the real-world models describing specific domains from M0 layer.

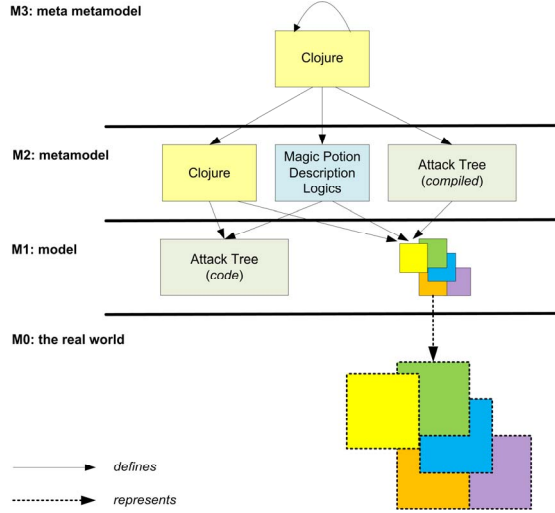


Figure 1. Magic Potion and Attack Tree in MDA layered architecture.

B. Magic Potion in Based on Description Logics

Magic Potion [1] is a Domain Specific meta-Language for modeling various domains that is theoretically grounded on DLs. As it is still highly evolving language, this paper discusses some of the basic correspondences between DLs concepts and Magic Potion’ DL [18]:

- A top concept in DLs, which is interpreted as the whole domain, corresponds to Magic Potion top concept thing that further subsumes all Magic Potion concepts;
- A concept-name in DLs corresponds to (concept name [...] [...]) in Magic Potion;
- A role-name in DLs corresponds to (property name [...] [...]) in Magic Potion;
- An individual-name in DLs is an instance of a given concept-name in Magic Potion.

According to the Magic Potion semantics, the interpretation function assigns a set Δ^A to every concept-name, and a binary relation $(\Delta^A \times \Delta^A)$ to every property name. Likewise, the interpretation function specifies membership of every DLs individual-name to the Magic Potion concept-name.

C. Magic Potion is Inspired by Functional Programming

Magic Potion is defined by using FP paradigm in mind; more specifically, it is grounded on Clojure FP language. Unlike ordinary DSL that are focused on creating descriptions of the data, Magic Potion compiles concepts into executable functions that create instances of the defined concepts as the groups of validated statements [18]. It follows DLs paradigm describing semantics of data, not their behavior.

D. The Goals of Magic Potion

We identify the following goals of Magic Potion:

- to offer flexible and semantically rich means for domain modeling; more specifically, to enable knowledge modeling through ontologies;
- to be practical and approachable for software developers with the minimal theoretical knowledge on the Semantic Web technologies;
- to be formally sound and theoretically based on DLs;
- to be established as FP language with the possibility to answer to the critical requirements of interactive knowledge.

IV. A MOTIVATING EXAMPLE PROBLEM FOR MODELING ATTACKS AND VULNERABILITIES

Our motivating example for modeling software attacks and vulnerabilities is inspired by the example given in [2]. This example shows the use of Attack Tree to derive scenarios for the specific attack injection purposes. The Attack Tree in [2] is constructed based on the available information about known attacks and vulnerabilities of the security layer of the Wireless Application Protocol (WAP) tack. The authors use the Wireless Transport Layer Security (WTLS) layer of the WAP gateway to provide security in the transactions between client and server in wireless network.

A. Attack Tree

Attack Tree [6, 19] is commonly used to model different ways in which a system can be attacked. It is a tree in which the nodes represent attacks: the root node is the global goal of an attacker; children of a node are refinements of this goal, while leafs represent attacks that can no longer be refined. A refinement can be conjunctive (aggregation; e.g., a case when all sub-goals have to be fulfilled) or disjunctive (choice). In other words, as defined in [6], Attack Tree has two types of nodes: AND-nodes and OR-nodes. The children of an AND-node should all be executed to reach the goal represented by the AND-node, while execution of any child of an OR-node suffices to reach the goal of the OR-node.

In order to calculate the cost or impact of an attack, an Attack Tree can be additionally described with attributes.

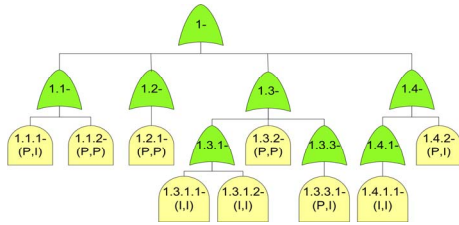
B. The WTLS Attack Tree

In order to create an Attack Tree, it is necessary to identify all possible attack goals [6]. Each goal forms a separate Attack Tree, but they can share subtrees and nodes. In this paper, we identify “Attack the WAP server” [2] as a generic goal of attacking the target protocol implementation,

while “Attack WTLS” is a root node of the WTLS Attack Tree.

The WTLS layer implements security functionalities offering *privacy*, data *integrity*, and *authentication* for WAP applications [2, 20]. Hence, the next level of the WSDL Attach Tree involves (i) privacy, (ii) integrity, and (iii) authentication, as the properties that the attacker attempts to violate. All the next levels of the WSDL Attack Tree decompose and more specifically describe known attacks and possible vulnerabilities of the WTLS.

Additionally, the attacker capabilities, such as “possible according to attacker capabilities” and “possible according to actual test bed”, have been identified in [2]. Figure 2 shows graphical notations of the WTLS Attack Tree. P denotes attacker capabilities which are “possible”; “I” denotes “impossible”.



C. The WTLS Attack Scenario

In order to show how Magic Potion, as a Domain Specific meta-Language, can be used to define more specific Attack Tree DSL that implements a newly proposed tool for modeling and interacting with software attacks and vulnerabilities, we choose the following attack scenario: “<1.1.2> Transaction attack in “*_close_notify” message”.

Table 1 presents actions from the WTLS “<1.1.2> Transaction attack in “*_close_notify” message” attack scenario in a more refined manner.

TABLE I. THE WTLS SCENARIO 1.1.2

Attack Number	Description of the Attack Action
1.1.2-	Truncation attack in “*_close_notify” message
1.1.2.1-	Check if message is “*_close_notify” type
1.1.2.2-	Drop “*_close_notify” message
1.1.2.3-	Drop last record of connection

In [2], the attack actions describing the WTLS scenario “<1.1.2> Transaction attack in “*_close_notify” message” are further converted into the <trigger, condition, action> notation. Such a notation serves as an active rule with the purpose to represent reactive behavior of a system. The semantics of the active rule <trigger, condition, action> is the following: if the <trigger> event occurs and the <condition> is verified, then the <action> is executed.

Tables 2-4 describe the active rules applied to the whole WTLS attack scenario “<1.1.2> Transaction attack in “*_close_notify” message”. These rules present an abstract fault scenario that is platform independent.

TABLE II. THE ACTIVE RULE FOR THE ACTION 1.1.2.1

1.1.2.1-	Check if message is “*_close_notify” type
trigger	an UDP packet is sent from the client (mobile) to the server (WAP gateway)
condition	(packet type byte is Alert Message) and (message type is ‘connection_close_notify’ or ‘session_close_notify’)
action	set state variable to indicate message type

TABLE III. THE ACTIVE RULE FOR THE ACTION 1.1.2.2

1.1.2.2-	Drop “*_close_notify” message
trigger	an UDP packet is sent from the client (mobile) to the server (WAP gateway)
condition	(packet type byte is Alert Message) and (message type is ‘connection_close_notify’ or ‘session_close_notify’)
action	drop message and set state variable (true value) to indicate next record truncation

TABLE IV. THE ACTIVE RULE FOR THE ACTION 1.1.2.3

1.1.2.3-	Drop last record of connection
trigger	an UDP packet is sent from the client (mobile) to the server (WAP gateway)
condition	state variable is true
action	drop record

V. ATTACK TREE DOMAIN SPECIFIC LANGUAGE

The aim of this paper is based on use of Attack Tree together with Magic Potion Domain Specific meta-Language as a way to create a new Attack Tree DSL. We claim that Attack Tree DSL is, at the same time, as follows:

- a functional programming tool that improve expressiveness of attack and vulnerability models and provide a new way of interacting with them;
- a user-friendly Semantic Web enabler that is built for a specific domain of software attacks and vulnerabilities.

In order to determine components of the Attack Tree DSL, we firstly build an Attack Tree metamodel (Figure 5). Following a UML graphical notation, the main components of Attack Tree, such as: AttackGoal, AttackProperty and ActionRule are shown in Figure 5. The ActionRule further consists of Trigger, Condition, and Action components.

An AttackGoal is composed of AttackProperty. This means that AttackProperty is an integral part of AttackGoal and cannot exist independently of it.

An AttackProperty is associated with AttackerCapability, for which we can specify either of two exclusive values: possible or impossible. An

AttackerCapability may be further associated with any number of AttackProperty.

AttackProperty is described by an ActionRule, which is composed of a Trigger that aggregates a Condition and an Action.

Analogous to Figure 1, which is a general representation of Magic Potion placed in four-layer MDA, we provide a more specific description showing the Attack Tree DSL aligned to the rest of Magic Potion general architecture. The Attack Tree DSL in four-layer MDA is shown in Figure 1.

VI. ATTACK TREE SCENARIO SPECIFICATION

This section examines the Attack Tree DSL validation scenario at the domain level, such as, e.g. the active rule of WTLS attack scenario that describes the attack action “<1.1.2.1> Check if message is “*_close_notify” type”.

For the sake of simplicity and brevity, we focus on a part of the Attack Tree’ metamodel (Figure 4) that considers only the following three components of the Attack Tree’s action rule: a Trigger, a Condition, and an Action.

Now, we use the set of values given in Table 5, which are assigned to the target action rule “<1.1.2.1> Check if message is “*_close_notify” type”, and put these values in the context of the Attack Tree ActionRule. By using a pseudo-code, we can describe a selected part of the target action rule in the following way:

```
CheckIfMessageIs "*" _Close_Notify" Type MobilePhone
UDPpacket Send WAPGateway AlertMessage
Session_Close_Notify StateVariable Set MessageType
```

By combining Figure 7 with the values defined in Table 5, we create the following scenario:

First, ActionRule known as “<1.1.2.1> Check if message is “*_close_notify” type”, initiates the Trigger component of the action rule. The client concept, which is defined in the Trigger component and holds the value “Mobile phone” initiates the triggerObject with the value “UDP packet”. Then, the triggerObject has to be executed via a triggerAction such as “Send” to the server that is known as “WAP Gateway”.

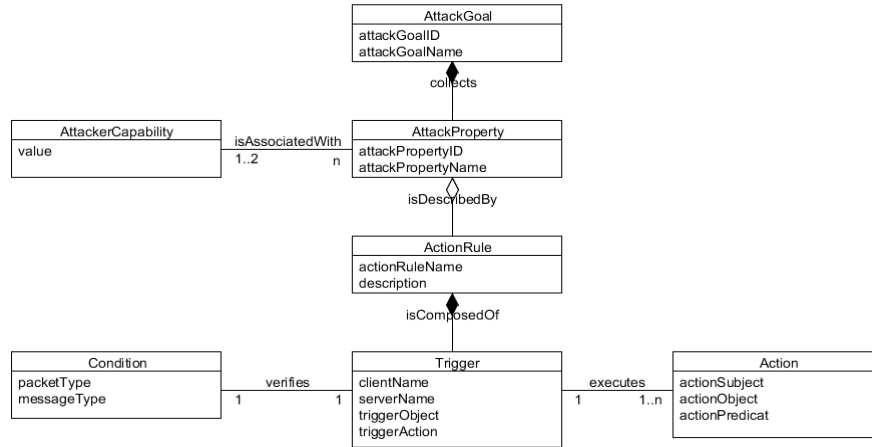


Figure 2. A UML diagram of Attack Tree metamodel.

Second, the Condition component of the action rule is launched initializing the packetType with the value “AlertMessage”, and the messageType that can be whether “connection_close_notify” or “session_close_notify”. Finally, the Action component of the action rule initiates ActionSubject with the value “StateVariable” that is followed by an ActionObject “Set” to indicate an ActionPredicat such as “MessageType”.

	triggerObject	UDP packet
	triggerAction	Send
Condition	packetType	Alert Message
	messageType	‘connection_close_notify’ / ‘session_close_notify’)
Action	actionSubject	state variable
	actionObject	set
	actionPredicat	message type

The Active Rule for the Action 1.1.2.1

Action Rule component	Action Rule subcomponent	Value
ActionRule	actionRuleName	Check if message is ‘*_close_notify’ type
Trigger	clientName	Mobile phone
	serverName	WAP gateway

Based on the above description of the WTLS attack scenario, the rest of this paper considers creation of an executable Attack Tree DSL that is grounded on the Attack Tree’ metamodel, refined to Magic Potion in MDA four-layered architecture, and uses the vocabulary that consists of the Attack Tree’s related components, such as properties, concepts, relationships and individuals.

Thus, to describe the Attack Tree DSL, we firstly define the following set of properties, concepts and relationships:

- Properties, such as `actionRuleName`, `client`, `server`, `triggerObject`, `triggerAction`, `packetType`, `messageType`, `actionSubject`, `actionObject`, `actionPredicat`
- Concepts, such as `ActionRule`, `Trigger`, `Condition`, `Action`, and
- Object properties, effectively relationships such as `isComposedOf`, `verifies`, `executes`.

Written in Magic Potion manner, we define our specific Attack Tree DSL by describing the above set of properties, concepts and relationships.

VII. CONCLUSIONS

In this paper, we define Attack Tree DSL which is based on Attack Tree formalism that provides a formal methodology for analyzing the security of the system, and is implemented on top of Magic Potion. Magic Potion is a Domain Specific meta-Language that is grounded in FP and DLs.

Central to our work is idea of using Magic Potion as a novel approach for modeling and interacting with particular domains and their knowledge. Such a modeling and interaction is fully inspired by the MDA and FP paradigm, and enables using ontologies without the need for having the complex ontology tools and/or huge Semantic Web infrastructure.

As one of the great inhibitors to the widely use of the Semantic Web languages and their fully acceptance outside of the Semantic Web community, the complexity of the ontology languages and the need for using complex ontology tools and difficult-to-manage infrastructure, have been noticed. Hence, in this paper we explore Magic Potion as an approach to help in the direction to the wide adoption of the Semantic Web technologies, especially from the side of software developers that possess the minimum of the theoretical knowledge on the Semantic Web.

Magic Potion is not intended to be a replacement for the Semantic Web technologies and standards such as RDF or OWL, neither has the same goals. As it is still highly evolving, in this section we emphasize the only currently known benefits and weaknesses, and steps to be done:

The main benefits of Magic Potion is related to the support in knowledge modeling and reasoning about the common knowledge, as well as the behavior of knowledge that transform from one state to the other, even to the users with the minimal theoretical background on the Semantic Web. Also, Magic Potion is a purely functional DSL language with a possibility to answer to the critical requirements of interacting with knowledge. By building on Clojure's support for parallel programming, it is a good fit for modeling problems that have to be executed on multicore processors.

REFERENCES

- [1] D. Djuric, J. Jovanovic, V. Devedzic, R. Sendelj, "Modeling Ontologies as Executable Domain Specific Languages," In Proceedings of the 3rd Indian Software Engineering Conference, Mysore, India, 2010.
- [2] E. Martins, A. Morais, and A. Cavalli, "Generating Attack Scenarios for the Validation of Security Protocol Implementations," Proc. of the 2nd Brazilian Workshop on Systematic and Automated Software Testing (SBES 2008 - SAST), Brazil, October 2008, pp.21—33.
- [3] W. Jimenez, A. Mammam, and A. Cavalli, "Software Vulnerabilities, Prevention and Detection Methods: A Reviw," Proc. European Workshop on Security in Model Driven Architecture 2009 (SECMDA 2009), Enschede (The Netherlands), June 24, 2009, pp.6—13.
- [4] D. Byers, S. Ardi, N. Shahmehri, and C. Duma, "Modeling Software Vulnerabilities with Vulnerability Cause Graphs," Proc. of the International Conference on Software Maintenance, Philadelphia, PA, USA, 2006.
- [5] H. Peine, M. Jawurek, and S. Mandel, "Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection," Proc. 11th IEEE High Assurance Systems Engineering Symposium (HASE 2008), pp. 9—18, 2008.
- [6] B. Schneier, "Attack Trees: Modleing Security Threats," Dr Dobb's Journal, Vol.24, No.12. Online available at: <http://www.schneier.com/paper-attacktrees-ddj-ft.html>. Retrieved January 2010.
- [7] P. Hudak, "Conception, Evolution and Application of Functional Programming Languages," ACM Computing Surveys 21 (3), pp. 359—411, 1989.
- [8] J. Fokker, "Functional Programming," Utrecht University, 1995. Online available: <http://people.cs.uu.nl/jeroen/courses/fp-eng.pdf>
- [9] J. Armstrong, "Making Reliable Distributed Systems in the Presence of Software Errors," PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, 2003.
- [10] M. Odersky, L. Spoon, B. Venners, "Programming in Scala," Artima, 2008.
- [11] A. Rathore, "Clojure in Action," Manning Early Access Program, Manning Publications, 2009.
- [12] A.V. Deursen, J. Visser, "Domain Specific Languages: An Annotated Bibliography," ACM Sigplan Notices, 35(6), pp: 26—36, 2000.
- [13] T.B. Lee, J. Hendler, O. Lassila, "The Semantic Web," Scientific American, 284(5), pp. 28—37, 2001.
- [14] S. Decker, S. Melnik, F. van Harmelen, "The Semantic Web: The Roles of XML and RDF." IEEE Internet Computing, pp. 63—74, 2000.
- [15] J.Z. Pan, I. Horrocks, "Metamodeling Architecture of Web Ontology Languages," In The Emerging Semantic Web, I.F.Cruz at al., (Eds.), IOS Press, pp. 21—45, 2002.
- [16] D. Gasevic, D. Djuric, V. Devedzic, "Model Driven Engineering and Ontology Development," 2nd Ed. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2009.
- [17] D.C. Schmidt, "Guest Editor's Intorduction: Model Driven Engineering," Computer, pp. 25—31, 2006.
- [18] D. Djuric, N. Krdzavac, "Software Transaction Memory Powered Domain Modeling," Submitted to the Journal of Parallel and Distributed Computing (year of submitting 2009).
- [19] S. Mauw, M. Oostdijk, "Foundations of Attack Tree," in LNCS, ICISC 2005, D.Won, S. Kim (Eds.), Springer, Heidelberg, Vol. 3935, pp. 186—198, 2006.
- [20] WAP Forum, "WAP Architecture, Version 12-July-2001." Online avaialble. Last accessed in May 2008: <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>