A Modular Logic Approach for Expressing Web Services in XML Applying Dynamic Rules in XML

Theodoros MITAKOS, Ioannis ALMALIOTIS
Technological Educational Institute of Chalkida, Chalkida, Greece
mitakos@teihal.gr, alma@teihal.gr

RuleML is considered to be a markup language for the semantic web. It allows the enrichment of web ontologies by adding definitions of derived concepts and it enhances interoperability among different systems and tools by publishing rules in an XML format. Moreover the increasing demand for interfaces that enhance information sharing has given rise to XML documents that include embedded calls to web services. In this paper we propose a variation of RuleML that is based on modular logic programming. Our approach is based in a two level architecture. In the first level a modular logic language, called M-log, is presented. This language encompasses several mechanisms for invoking web services. In the second level we exploit the semantics of M-log to present a variation of RuleML with rich modeling capabilities. Formal foundations for this variation are given through direct translation to M-log semantics.

Keywords: *Knowledge Management, XML, Modular Logic Programming, E-Services*

1 Introduction

XML [4] has succeeded to be established both in the research community and also in software development industry as a self-describing, semi structured data model. It has become a standard for data exchange between applications over the Web. On the other hand exchange of static data is not enough for contemporary web applications. Modern web applications need not only mere data exchange but also powerful programmatic interfaces that are able to support communication and interoperability in highly diversified environments. Web service is the commonly used term to describe such programmatic interfaces in the web.

Consider as a simple example the case where a library borrows books to readers. We want to model in XML the possible readers, the library with the available books and the business rules that allow the borrowing of books. This scenario presents a number of difficulties that traditional XML modeling capabilities fall short to satisfy. It is needed to model business logic that expresses rules such as to whom books are borrowed. It is also needed to adopt a flexible scheme for expressing the dynamic nature of stored information e.g. the possibility of enriching the library with new books or some books to become unavailable to readers because they are already borrowed to some other readers. Moreover XML documents have to model calls to other XML documents e.g. information about available books may exist in one document and registered readers may exist in another document but we need to know if a reader can be borrowed a book.

Web services need to invoke programs over the web so that dynamic, up-to-date data to be extracted from various sources of information. Several interesting attempts have been made towards this direction. SOAP [2] and WSDL [3] are standards, based on XML, that specify the exchange of structured information in the implementation of web services. Additionally AXML [1] is an interesting declarative framework that harnesses Web services for distributed data management, and is put to work in a peer-to-peer architecture.

Our approach also specifies a declarative framework that allows both the exchange of intentional and extensional information. It is based on modular logic programming and on the language RuleML [7]. We propose a two level architecture that it based on well established standards. It also has firm theoretical foundations, concise and elegant syntax.

We focus on modeling calls to web services as Datalog queries among distributed Ru-

leML documents. The main aspects that are considered are operational semantics that form the foundations of calls to web services and appropriate syntax that expresses different ways of information sharing among web interfaces. The first objective is accomplished with the introduction of a modular logic language [5,6] and the second objective with the introduction of a RuleML variation, that is built upon the first language.
This paper is organized as follows. First we present briefly the language Datalog and then a variation of Datalog called M-log. Additionally we present XML and RuleML in brief. Next we present the syntax and semantics of RuleML-U. We conclude giving directions for future research.

2 Datalog

In this section we give a brief presentation of a "toy" deductive database language called Datalog [8]. Datalog is a rule based language. A rule has the form:

h:-b1,b2,…bn

where h,b1,b2,….,bn are atoms, that is predicates or relation names with variables or constants as arguments. The symbol :- means implies. The left hand side of :- is called head and is always consisted of one and only one atom. The right hand side of :- is consisted by one or more atoms. The atoms in the body are coma separated. In the context of a Datalog rule coma means logical AND. A variable appearing in the head is called distinguished. A variable that appears in the body is called non distinguished. Distinguished variables are universally quantified. Non-distinguished variables are existentially quantified.
The following sentence describes the semantics of a rule. If there exist values of the non distinguished variables that make all sub goals of the body true then the head of the rule is also true. That is the head of a rule is concluded true if all literals in its body can be concluded.
A collection of rules is called Datalog theory. Moreover a Datalog program consists of a collection of rules. A rule that has only head and nobody is called fact. The set of facts in a program is called EDB that is extensional database. The set of rules with body and head is called IDB that is intentional database. In the following we consider that no negative literals exist in the body of rules. The semantics of a Datalog program without negation is given by the least Herbrand model.
The expressive power of Datalog without recursive rules is the same as SQL without aggregation and grouping. Moreover if recursive rules are allowed the expressive power of Datalog is greater than the expressive power of relational algebra.
It can thus be considered as the subset of logic programming needed for representing the information of null-value-free relational databases, including (recursive) views. That is, in Datalog we can define facts corresponding to explicit rows of relational tables and rules corresponding to tables defined implicitly by views.

3 The language M-Log

In this section we are going to present a language that is based on Datalog and modular logic programming. It is also capable to express message passing mechanism with various ways.

3.1 Defnitions and Syntax of M(essage)-LOG

In classical logic programming there are no tools for structuring and modularizing programs. In this section we present the syntax of the language m(essage)-log which extends the traditional notion of unit by allowing clauses to exchange messages. A unit can be considered as a named, finite collection of rules. The mechanism of message exchanging is well known as message passing in object oriented languages. Actually, in object oriented terms it means a method invocation. After the execution of the invoked method the results are passed back to the calling program. In the following we will show four mechanisms of message passing that can be adopted by M-LOG. The first can be considered as static or early execution mechanism. The other three can be considered as dynamic or late execution mechanisms. The syntax of Datalog is enhanced with units and with symbols $\supset$ and >. If u is a unit and A is an atom then $u \supset A$ is atom and u>A is also an atom. If p is an atom then $u \supset p$ is called external atom and u<p is called internal atom. The alphabet of M-LOG consists of a finite set of unit names U, a finite set of predicate names P, a countable set of variables V and a finite set of constants C. To every $u \in U$ we assign a set of clauses. The clauses that correspond to unit u are called unit implementation. The set of ground instances of the clauses of a unit implementation is denoted as |u|. Moreover, if p is a predicate name, the set of clauses that have p as their head is called implementation of p. If S is a set of clauses then we denote with $\pi(S)$ the set of predicate names defined by the clauses of S. Is A is an atom then pred (A) is the predicate of A.

Datalog is a database query language based on logic programming and consists of function free Horn clauses. The clauses in our proposal follow the syntax of positive Datalog clauses extended with operators >, $\supseteq$, $\supset$ and $\oplus$. E.g. p:-q, u>r or p:-q, $u \supset r$.

Operator > accepts two operands. The first operand is a unit identifier u and the second is a goal r. Let say that a rule ru' = pu':-q_1u', q_2u', .., u>r , .., q_nu' exists in unit u'. It means that the goal r must be evaluated using the predicates, $\pi(u)$, of unit u. That is the evaluation of r will take place in unit u. If r is evaluated true in unit u then u>r is also true in unit u'.

Operator $\supseteq$ accepts two operands. The first operand is a unit identifier u and the second is a goal r. Let say that a rule ru' = pu':-q_1u', q_2u', .., $u \supseteq r$, .., q_nu' exists in unit u'. This means that all EDBs of unit u that have predicate r must be imported from unit u to unit u'. Then the evaluation of goal r will take place in unit u'. After the evaluation of r the imported EDBs will be rejected.

Operator $\supset$ accepts two operands. The first operand is a unit identifier u and the second is a goal r. Let say that a rule ru' = pu':-q_1u', q_2u', .., $u \supset r$, .., q_nu' exists in unit u'. This means that all IDBs of unit u that have predicate r must be imported from unit u to unit u'. Then the evaluation of goal r will take place in unit u'. After the evaluation of r the imported IDBs will be rejected.

Operator $\oplus$ accepts two operands. The first operand is a unit identifier u and the second is a goal r. Let say that a rule ru' = pu':-q_1u', q_2u', .., $u \oplus r$, .., q_nu' exists in unit u'. This means that all EDBs and IDBs of unit u that have predicate r must be imported from unit u to unit u'. Then the evaluation of goal r will take place in unit u'. After the evaluation of r the imported EDBs and IDBs will be rejected.

If A is a Datalog atom then A is an M-LOG atom too. A is called simple atom.

If A is a Datalog atom then u>A is an M-LOG atom too.

If A is a Datalog atom then $u \supseteq A$ is an M-LOG atom too.

If A is a Datalog atom then $u \supset A$ is an M-LOG atom too.

If A is a Datalog atom then $u \oplus A$ is an M-LOG atom too.

Any M-LOG atom that is not simple atom is called composite atom. Composite atoms can appear only in the body of a rule.

The set of EDBs for a unit u is denoted as EDB(u) and the set of IDBs for a unit u is denoted as IDB(u). Let Su set of clauses that belong to unit u, then Su=EDB(u) $\cup$IDB(u).

If p is a predicate name that belongs to a unit u we denote as EDB(u,p) the set of EDBs that belong to unit u and have predicate name p. Moreover If p is a predicate name that belongs to a unit u we denote as IDB(u,p) the set of EDBs that belong to unit u and have predicate name p.

In our approach a unit consists of two sets of clauses, namely its EDBs and its IDBs. Operators $\amalg$ EDB, $\amalg$ IDB and $\amalg$ are used to handle composition of units. They are defined as follows:

$u_p \amalg_{EDB} u_q = Su_p \cup EDB(u_q) = u_z$

$u_p \amalg_{IDB} u_q = Su_p \cup IDB(u_q) = u_z$

$u_p \amalg u_q = Su_p \cup Su_q = u_z$

Moreover

$u_p \amalg_{EDB,r} u_q = Su_p \cup EDB(u_q, r) = u_z$

$u_p \amalg_{IDB,r} u_q = Su_p \cup IDB(u_q, r) = u_z$

$u_p \amalg_{r} u_q = Su_p \cup EDB(u_q, r) \cup IDB(u_q, r) = u_z$

$u_{zpriv} = \{A\text{:-}G \in u_{ppriv}\}$

3.2 Operational Semantics of M-Log

The operational semantics of M-LOG is given as follows. The proof predicate ├ is defined by the following inference rules, where ε is the identity substitution and ỉ is the empty formula.

Empty formula		
	――――――	
	u ├$_\varepsilon$ ỉ	
Atomic formula	u ├$_\sigma$G θ	h:-G∈u θ= mgu(g,h)
	――――――	
	u ├ $_{\theta\sigma}$g	
Conjunction	u ├$_\theta$G$_1$ ∧ u ├$_\sigma$G$_2$ θ	
	――――――	
	u ├ $_{\theta\sigma}$ G$_1$, G$_2$	
Message passing	ữ ├$_\theta$G	pred(G) ∈π(ữ)
	――――――	
	u ├$_\theta$ ữ>G	ữ≠u
EDB facts insertion	u' ├$_\theta$g	u’=u ∐ $_{EDB,g}$ ữ ữ≠u
	――――――	
	u ├$_\theta$ ữ⊒g	
IDB rules insertion	u' ├$_\theta$g	u’=u ∐ $_{IDB,g}$ ữ ữ≠u
	――――――	
	u ├$_\theta$ ữ⊃g	
EDB facts and IDB rules insertion	u' ├$_\theta$g	u’=u ∐ $_g$ ữ ữ≠u
	――――――	
	u ├$_\theta$ ữ⊕g	

4 XML

XML (Extensible Markup Language) is a flexible tag based language that is used for data exchange on the World Wide Web, intranets, and elsewhere. Unlike HTML makes it possible to define the content of a document separately from its formatting. The basic object in XML is the XML document. Elements and attributes are the main structuring concepts that are used to construct an XML document. We do not consider in this paper additional concepts such as identifiers or references.

XML documents are structured following the tree data model. They consist of an element on the top level that contains all other elements. This element is called root. An element can contain other elements. In case of an element that does not contain any sub element then it is called leave. Attributes in XML provide additional information that describes element properties. Attributes are used together with elements to represent the pieces of information that compose an XML document. A document that conforms to the XML syntax rules is called well formed. Moreover constraints on the structure and content of an XML document can be described by some form of an XML schema e.g. a DTD. An XML document that conforms to a specific XML schema is called valid.

The number of tags that can be used in a document is not predefined by the language itself. On the contrary the composer of an XML document is free to use the tags that describe better the meaning of the data that are included in the document. The markup symbols that can be used are unlimited and self-defining. One of the reasons that XML is designed that way is because web applications need not only to exchange documents but also to interpret their contents automatically. XML began as a simplified subset of the Standard Generalized Markup Language (SGML) and today it is a formal recommendation from the World Wide Web Consortium. Its success gave birth to application languages implemented in XML. These include but are not limited to RuleML, MathML, GraphML and MusicXML. Nowadays XML is not used only as vehicle for information sharing in a consistent way but also as specification language for such application languages.

5 The language RuleML

In this section we will briefly describe the language RuleML. RuleML is a markup language for publishing and sharing rules on the World Wide Web. It uses Datalog as the kernel of its family of sublanguages. Its semantics is defined via Herbrand models. RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL. In the following we give an example that shows how Datalog rules can be expressed in RuleML. Let's consider the sentence “John is son of Mary”. This natural language sentence can be modeled as the following fact in Datalog: son(john,mary). This simple atom can be modeled in RuleML as follows:

```
<Atom>
  <Rel>son</Rel>
  <Ind>john</Ind>
  <Ind>mary</Ind>
</Atom>
```

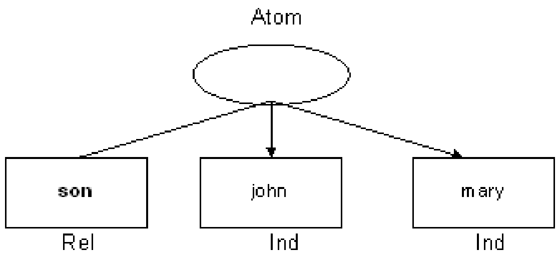

Fig. 1. A simple RuleML atom modeled as a parse tree

Since this markup follows the XML syntax it can be represented as a kind of parse tree. Oval drawing corresponds to anonymous, non-terminal, inner node labeled ‘Atom’. Figure 1 shows the modeling of a simple RuleML atom as a parse tree.
Rectangular drawings correspond to leaf nodes with RDF-like literals containing PCDATA. These terminal nodes are labeled ‘Rel’ and ‘Ind’.
We examine the nodes of the tree left to right and in a bottom up fashion. Notice that "son" is marked up as the relation name (table name) for the fact: <Rel>son</Rel>. On the same level, the two names *"John"* and *"Mary"* are marked up as individual constants that are the two arguments (table columns) of the relation, in the given sequence: `<Ind> john </Ind>` and `<Ind> mary </Ind>`. The entire relation application constitutes an atomic formula, marked up by the tags <Atom> ... </Atom>.
In the following we give an example of a rule. Consider the following English sentence:
"X is parent of John if John is son of X."
This natural language sentence is an implication. It can be modeled as the following rule in Datalog:

parent:-son(john,X)

It can be marked up as the following RuleML Datalog rule:

```
<Implies>
  <head>
    <Atom>
      <Rel>parent</Rel>
      <Var>X</Var>
      <Ind> john</Ind>
    </Atom>
  </head>
  <body>
    <Atom>
      <Rel>son</Rel>
      <Ind> john</Ind>
      <Var>X</Var>
    </Atom>
  </body>
</Implies>
```

This rule is represented as a parse tree in figure 2.

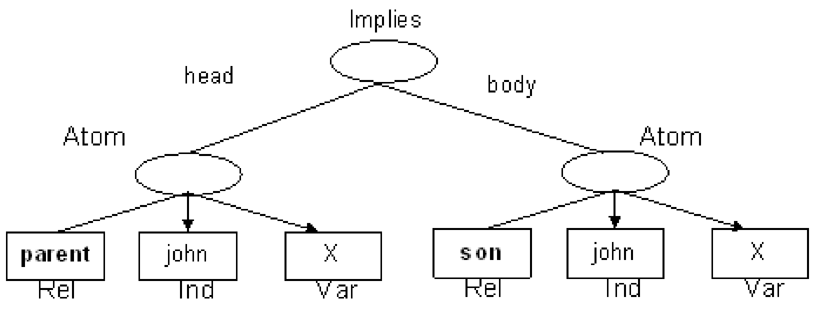

Fig. 2. A RuleML rule modeled as a parse tree

6 RuleML-U

Various RuleML Datalog implementations exist. We propose a variation of RuleML that facilitates implementation and calling of web services. This variation is based on M-log. We propose new tags to represent operators, a new operator to express units and a new tag to express composite atoms of M-log. In the following we give part of a DTD that specifies operators, units and composite atoms.

```
<!ELEMENT Operator (#PCDATA)>
<!ELEMENT Unit (#PCDATA)>
<!ELEMENT C-atom (Unit, Operator,
Atom)>
```

Accepted parsed character data for element operator are only the abbreviations "ms", "ie", "ii" and "iei". Abbreviation "ms" stands for message passing operator and abbreviations "ie", "ii" and "iei" stand for EDBs insertion, IDBs insertion, EDBs and IDBs insertion operator accordingly.
We also give an equivalent XML schema.

```
<?xml version="1.0" encoding="UTF-8"
?>
<xs:schema targetNames-
pace="http://www.ruleml.org/0.9/xsd"
xmlns="http://www.ruleml.org/0.9/xsd
" xmlns:xs =
"http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:annotation>
<xs:documentation xml:lang="en">
  </xs:documentation>
</xs:annotation>
<xs:element name="C-atom">
  <xs:complexType>
    <xs:sequence>
<xs:element name="Unit"
type="xs:string"/>
<xs:element name="Operator">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="ms"/>
<xs:enumeration value="ie"/>
<xs:enumeration value="ii"/>
<xs:enumeration value="iei"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Atom"
type="Atom.type" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The previous code shows an XML Schema module for M-Log related RuleML elements.
In the following we demonstrate the modeling capabilities our new RuleML variation. We refer to this variation as RuleML-U. We consider the following simplified scenario. Three units that contain EDBs and IDBs are presented:

Unit Library

```
borrows(Title):- book(Title), avail-
able(Title)
book(database systems)
book(programming languages)
book(operating sys-
tems)available(programming languag-
es)
available(operating systems)
```

Unit University

```
lent(Person,Title):- reader(Person),
Library>borrows(Title)
reader(Person):-
Student⊒classC(Person)
classC(Person):-faculty(Person) fa-
culty(john)faculty(samantha)
```

Unit Student

```
classA(nikos)   classB(george)
classC(mary)    classC(jim)
```

These three given units correspond to the following RuleML-U documents:

```
<document name="Library">
<Implies>
<head>
  <Atom>
    <Rel>borrows</Rel>
    <Var>Title</Var>
   </Atom>
</head>
<body>
  <Atom>
    <Rel>book</Rel>
    <Var>Title</Var>
  </Atom>
  <Atom>
  <Rel>available</Rel>
  <Var>Title</Var>
</Atom>
</body>
</Implies>
<Atom>
  <Rel>book</Rel>
  <Ind> database systems </Ind>
</Atom>
<Atom>
  <Rel>book</Rel>
  <Ind> programming languages
</Ind></Atom>
<Atom>
  <Rel>book</Rel>
  <Ind> operating systems </Ind>
</Atom>
<Atom>
```

```
  <Rel> available </Rel>
  <Ind> programming languages </Ind>
</Atom>
<Atom>
  <Rel> available </Rel>
  <Ind> operating systems </Ind>
</Atom>
</document>

<document name="University">
<Implies>
<head>
  <Atom>
    <Rel>lents</Rel>
    <Var>Person</Var>
     <Var>Title</Var>
  </Atom> </head>
<body>
  <Atom>
  <Rel>reader</Rel>
  <Var>Person</Var> </Atom>
  <C-atom>
<Unit> Student </Unit>
<Operator> ie </Operator>
<Atom>
  <Rel>classC</Rel>
  <Var>Person</Var>
</Atom> </body> </Implies>
<Implies>
<head>
  <Atom>
    <Rel>classC</Rel>
    <Var>Person</Var>
  </Atom> </head>
<body>
  <Atom>
      <Rel>faculty</Rel>
      <Var>Person</Var>
  </Atom>
  <Atom>
     <Rel>faculty</Rel>
     <Ind> john </Ind>
  </Atom>
<Atom>
     <Rel>faculty</Rel>
     <Ind> samantha </Ind>
</Atom> </body> </Implies>
</document>

<document name="Student">
<Atom> <Rel>classA< /Rel>
    <Ind> nikos </Ind>
</Atom>
<Atom> <Rel>classB </Rel>
    <Ind> george </Ind>
</Atom>
<Atom> <Rel>classC< /Rel>
    <Ind> mary </Ind>
</Atom>
<Atom> <Rel>classC </Rel>
    <Ind> jim </Ind>
</Atom>
</document>
```

A unit called "Library" contains two types of EDBs, that is `book` and `available`. EDB `book` denotes the books that library possesses. EDB `available` denotes the books that are currently available for borrowing. The IDB `borrows` denotes a business rule for borrowing books. Similarly a unit called "Student" contains three types of EDBs, that is `classA`, `classB` and `classC`. This means that three different categories of students exist in unit "Student". Lastly a unit called "University" contains an EDB called `faculty` and three IDBs, that is `lent`, `reader` and `classC`. Notice that in the body of IDB lent exists a message passing atom, that is `Library>borrows(Title)`. Moreover in the body of IDB `reader` there exists an EDB insertion atom, that is `Student⊇classC(Person)`. In the first case the computation takes place in unit "Library" and the atom borrows is concluded in unit "Library". If it is evaluated true in unit "Library" then atom `Library>borrows(Title)` is also true in unit "University". In the second case the EDBs with predicate name `classC` that belong to unit "Student" will be imported to unit "University" and they will take part in the evaluation of atom `classC(Person)` in unit "University". If `classC(Person)` is evaluated true in unit "University" then `Student⊇classC(Person)` is also true in unit "University".

7 Conclusions

In this paper we presented a variation of language RuleML that is called RuleML-U. This variation is influenced by modular logic programming. It provides a flexible and rich interface for implementing calls to web services. We intend to extend our research on RuleML-U and enrich it with features such as object identity, inheritance and information hiding giving an object oriented flavor to this deductive language. We also intend to build a compiler for RuleML-U documents.

References

[1] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu and T. Milo, "Dynamic XML Documents with Distribution and Replication," *In Proc. of ACM SIGMOD*, 2003.

[2] Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP.

[3] Web Services Definition Language (WSDL). http://www.w3.org/TR/wsdl.

[4] Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/REC-xml.

[5] F. Afrati, I. Karali and T. Mitakos, "Datalog, Units and Information Hiding," *Proceedings of the 1997 International Conference in Languages and Models with Objects (LMO 97) - in cooperation with ECOOP'97*, pp. 147-159, October 1997, Brest.

[6] T. Mitakos and I. K. Almaliotis, "A modular logic approach for P2P networks, A deductive object oriented framework for P2P networks," *16th International shop on Systems, Signals and Image Processing*, June 18-20, 2009 Chalkida, Greece.

[7] The Rule Markup Initiative, Available at: http://ruleml.org/

[8] S. Ceri, G. Gottlob and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 1, March 1989.

Theodoros MITAKOS graduated the Faculty of Computer Engineering and Informatics at University of Patras in 1992. He achieved the PhD in Informatics in 1998 with thesis on Deductive and Object Oriented Databases at National Technical University of Athens. He has published as author and co-author over 15 articles in national and international conferences and journal. He is coauthor in one book (Introduction to databases and Spreadsheets). He collaborates with Technological Educational Institute of Chalkida, with Hellenic Open University and he is the director of software company InfoWise. He teaches courses, seminars and laboratories on software engineering, databases, operating systems and computer architecture. His current research areas are internet and multimedia databases, mobile databases, logic programming and distance learning.

Ioannis ALMALIOTIS received his B.Sc. in Mathematics from the Department of Mathematics at the University of Crete, Greece in 1983 and his M.Sc. in Computer Science from the Department of Computer Science at the Greek Open University in 2008. Since 1996 he serves as the head of the Network Operation Center at the Technological and Educational Institution of Chalkis, Greece. He also works as a computer science lecturer in the General Department of Sciences at the same place. He has participated in many seminars and workshops and his main research interests are in the area of wireless sensor networks, and algorithmic analysis and Java programming.