

Specifying Dynamic Security Properties of Web Service Based Systems

Artem Vorobiev and Jun Han

Faculty of ICT, Swinburne University of Technology, Melbourne, Australia
{avorobiev, jhan}@ict.swin.edu.au

Abstract

The security characteristics of web service based systems depend on those of the individual web services (WS) involved and the way in which they are related to each other. In principle, the security characteristics of WS or systems can be expressed in security properties that are published and available to external parties. Only by knowing the security properties of the individual WS another WS can invoke it (if it satisfies certain security requirements and capabilities) and the overall system's security properties can be analysed and deduced. In our earlier work, we have developed the security characterisation language, SCL, to specify the static security properties of software components and systems. In this paper, we use SCL for describing security properties of WS and further enhance this language with the capability of specifying the dynamic security characteristics. The extended version of SCL (E-SCL) incorporates such features as time, time intervals, time sequence, probability, runtime conditions, and alternative security properties. Furthermore, we have developed the WS security ontology and applied it together with E-SCL to publish the dynamic security properties of WS using OWL-S and analyse them dynamically. Our approach is illustrated with an example email system.

1. Introduction

Web Services (WS) will play a significant role in the next generation Web (Web 2) [19,29]. However, the attractive features of WS such as platform independence, XML and SOAP dependence, and simplicity to use, make them vulnerable to many security threats including new unexplored and inherited old problems [24]. Things become even worse when WS and Semantic Web approaches merge to create Semantic WS that are able to publish their semantic characteristics about functional and non-functional properties. Now, it is easier for intruders to generate attacks because semantic information of WS is publicly available. To protect themselves, WS need to utilize various security techniques that can be

changed dynamically in order to deal with the dynamic nature of WS based system (WSBS). The security characteristics of such WSBS depend on the individual constituent WS and the way in which these WS relate to each other. In principle, the security characteristics of WS or WSBS can be expressed in security properties [12] that are published and available to third parties.

Currently, there are several approaches, which allow publishing semantic data of WS, including OWL-S [22], Semantic Web Services Language (SWSL) [26], Web Service Modeling Ontology (WSMO) [28], WSDL-S [27], KAoS [10], and METEOR-S [17]. However, none of them are specially designed to express security requirements and capabilities of WS. Hence, there is the need for new approaches that allow publishing security properties and verifying them automatically.

In component context, we have developed in our earlier work the security characterisation language (SCL) for specifying the security properties of software components and systems [12,13]. Based on the Common Criteria [3], BAN logic [2], and logic programming [5], SCL is executable, and therefore allows automatic compatibility checking of security properties between interacting components by utilizing smodels [18].

In a component based system (CBS), a software component may require that certain security requirements are met by neighbouring components and provide a set of security guarantees to these neighbouring components. We call these security requirements and guarantees *required* and *ensured* security properties of the component. Using SCL, one can specify in a logic program style the required and ensured security properties of a component as part of its interface description. In general, the required and ensured properties are parts of the tails and heads of the logic program clauses respectively. When two components interact in a CBS, the relevant security properties of these components need to form a *compatible* compositional security contract, which shows that their ensured security properties satisfy

each others' required security properties. The exact description of SCL can be found in [13].

SCL assumes that the security provisions of software components do not change over their lifetime, and consequently describes their security properties in a static manner. In dynamic software systems, the components' behaviour and the system's structure and behaviour can change, responding to user requirements changes and environment perturbations. This includes the security properties of the components and the system. In particular, the specific security properties of the component or the system may depend on time or system events, have a particular probability, or even change in nature.

In this paper, we use SCL to describe security properties of WS and extend it with the capability of specifying the dynamic security characteristics of WS by incorporating such features as time, time intervals, time sequence, runtime conditions, probability, and alternative security properties. More specifically,

- Security properties with time, time intervals and time sequences state the period(s) of time when specific properties are valid.
- Security properties with probability describe in a flexible way the security characteristics of WS, where it cannot be predicted exactly which security provisions will apply.
- Alternative security properties allow the specification of WS with different sets of security properties at different situations.
- Runtime conditions allow expressing dynamic security properties.

The paper is organised as follows. In section 2, we introduce an example email system and identify the need to consider dynamic security properties. In section 3, we present the key features of the Extended SCL (E-SCL) and discuss the implications on security compatibility reasoning. In section 4, we introduce WS security ontology and E-SCL functions. In section 5, we illustrate the use of E-SCL, WS security ontology and OWL-S through the example. In section 6, we present related works. Finally, we draw some conclusions and identify future work in section 7.

2. Background and motivation

2.1. An example

We use the email system as a case study. The email system, as illustrated in Figure 1, may consist of an email WS, a key distribution (KD) WS and client WS installed on client PCs, laptops or PDAs. To establish secure connections, clients need cryptographic keys distributed by the KD WS. For example, Client1 (C1)

wants to send an encrypted message with several attached files to Client2 (C2). In order to securely communicate with C2, C1 needs a shared key, which should only be known to C2 and C1. For this purpose, there is the need for the security functionality `get_shared_key(KEY, ID1, ID2)` provided by KD, which distributes keys among different clients of the system.

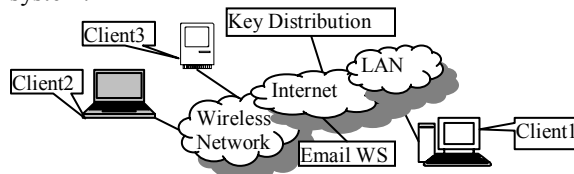


Figure 1. Email system

We assume that C1 and C2 send a lot of messages to each other, hence, they need a shared key instead of using their public/private keys, and we assume that public/private keys are already have been distributed.

The various WS of the system need to have compatible security provisions to ensure that the communications between them are carried out without compromising their functionality. This requires the explicit specification of their security properties (security requirements and capabilities) relative to their functionalities.

2.2 Motivation

SCL has provided a way for us to specify the security properties of individual components and to reason about the security compatibility between interacting components, leading to inter-component compositional security contracts. In a dynamic adaptive Web service system context, however, further challenges remain.

Firstly, SCL expresses the security properties of components while we want to apply it for describing security requirements and capabilities of WS. Secondly, SCL expresses only security properties of *static* systems without considering the *dynamic* changing nature of adaptive systems. Thirdly, there are cases where it is not possible to predict exactly what will happen with WSBS, when and what types of WS will be needed, etc, in future. For such purposes, probabilistic approximations can be useful. Finally, WS may utilize different sets of security techniques for different situations. Consequently, WS may need to specify alternative security properties.

3. E-SCL

In this section, we describe E-SCL including the constructs for specifying security properties that are

time and event dependent and probabilistic, and alternative security properties.

3.1. Time

In traditional security, time is used to check expiration date of security certificates or licenses, security sessions, etc. In E-SCL, time is applied to express dynamic security properties and qualify their validity. We adapt coordinated universal time (UTC) as a reference point, which is measured from 1 January 1970. Normal day and time are used implying their translation to UTC.

Time points. The time point $\langle T \rangle$ describes exactly when security properties of web services (WS) are valid. Time points are also used to express time intervals and time sequences. For example, Alice and Bob know that the shared key is valid at 9am but there are no guarantees that it is still valid at other time periods. In E-SCL it can be expressed in the following way: *shared(key,Alice,Bob,<9am>)*.

Time intervals. Time intervals are utilized to describe at which period of time certain security properties are valid. There are 14 types of time intervals, as shown in Figure 2.

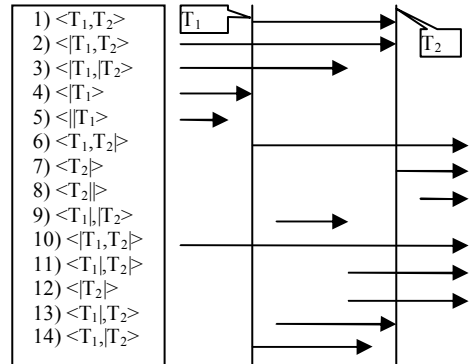


Figure 2. Time intervals

For example, $\langle T1,T2 \rangle$ shows that the security properties are valid exactly for the period of time from $T1$ to $T2$. The fact that the shared key between Alice and Bob is valid from 9am to 5pm can be expressed in E-SCL as: *shared(key,Alice,Bob,<9am,5pm>)*.

Other time intervals have different starting and ending times but their structures are quite similar, hence, they are not discussed here due to the space limitation.

Time sequences. Time sequences are used describe periods of time when security properties are valid. Members of a time sequence are ordered in the following sense: $\langle T_0 \rangle, \langle T_1 \rangle, \langle T_2 \rangle, \langle T_3, T_4 \rangle, \langle |T_5 \rangle, \dots, \langle T_{n-1}, |T_n \rangle$ where $0 \leq T_0 \leq T_1 \leq T_2 \leq \dots \leq T_n$ and $n \geq 0$. For

example, Alice and Bob can use the shared key from 9am to 10am and from 5pm till 7pm: *shared(key,Alice,Bob,<9am,10am>,<5pm,10pm>)*.

3.2. Probability

Probability is utilized if it is impossible to predict exactly what will happen with security properties of WS, when WS will demand certain services from other WS, etc. To describe probability, we employ an interval from 0 to 1 ($[0,1]$). A SCL expression $E_i \leftarrow R_i$ can be written with probabilistic properties as: $E_i([l_m, l_{m+1}]) \leftarrow R_i([p_k, p_{k+1}])$, where $p_k, p_{k+1}, l_m, l_{m+1} \in [0,1]$ and $p_k \leq p_{k+1}$ and $l_m \leq l_{m+1}$. The interval $[0, 0]$ means the security properties are not valid. The interval $[1,1]$ shows that the security properties are valid with 100% probability. For example, the shared key between Alice and Bob is valid from 9am to 5pm with the probability from 80% to 90%. In E-SCL it can be expressed as:

shared(key,Alice,Bob,<9am,5pm>,[0.8,0.9]).

3.3. Alternative security properties

Alternative security properties (in contrast to individual properties) allow the WS requestor to select an alternative match from a set of possible properties if it fails to satisfy a required property.

$\{E_i \leftarrow R_i, E_j \leftarrow R_j, E_l \leftarrow R_l\}, \{E_m \leftarrow R_m, E_n \leftarrow R_n\}, \dots$, where $i \geq 0, j \geq 0, l \geq 0, m \geq 0, n \geq 0$. Security property sets are shown as $\{\dots\}$. Only one of the individual security properties can be chosen from each set. A set of properties with one property inside $\{E_i \leftarrow R_i\}$ can be written as $E_i \leftarrow R_i$.

3.4. Compatibility reasoning

The required and ensured security properties of a service requestor (F) and responder (C) need to be compatible relative to time and probability before F can invoke C.

Time reasoning. When the security properties of F and C are valid with probability 100%, F can invoke C if their time properties concur exactly. For example, if F requires a service from 9am to 5pm and C provides it from 8am to 5pm then F can invoke C because time periods concur. If F requests a service from 9am to 10am and C provides this service from 11am till 4pm, then time periods do not intersect and F cannot invoke C. If F requests a service from 9am to 11am and C provides it from 10am to 4pm then F cannot invoke C because there is only a partial intersection of time periods from 10am to 11am and the requirements are not fully met.

Probability reasoning. Let us assume that time periods in the security properties of **F** and **C** concur exactly. When **F** requires a service from **C** with probability $[p_1, p_2]$ and **C** provides this service with probability $[p_3, p_4]$, **F** can invoke **C** if $p_3 \geq p_1 \geq 0$ and $p_4 \geq p_2 \geq 0$ where $p_2 \geq p_1$ and $p_4 \geq p_3$. For example, **F** requests a service with probability 80%-90% and **C** provides it with probability 85%-95%. In this case, **F** invokes **C** because **C** guarantees better probability.

3.5. Runtime conditions

Runtime conditions express when and how security properties are valid. They allow describing flexible scenarios:

```
if (<conditions>) {<security properties>}
else {<security properties>}
```

4. Web service security ontology

Our WS security ontology is partially based on the security ontologies from [15,7] and is used to express the security properties of WS. It is mapped on the E-SCL security functions and the Common Criteria (CC). There are six classes of the security functions that are derived from merging 11 CC classes: CryptographicSupport, IdentificationAuthorisation, Privacy, UserDataProtection, TrustedChannel, and SecurityAuditResourceUtilisation. Each of these classes has subclasses with the property hasSecurityObjective and which are mapped on the E-SCL security functions with parameters from the classes SecurityAlgorithm and SecurityConcept. There are other security classes including SecurityProperties, SecurityCredential, and SecurityAssurance that are not discussed here due to the page limit.

Security classes. The main security class is ServiceParameter with subclasses SecurityObjective (Fig. 3), SecurityAlgorithm (Fig. 4), SecurityConcept, SecurityAssurance, and SecurityCredentials. The OWL-S profile has the property ServiceParameter that can have the ServiceParameter class as a value. The property ServiceParameter further has the subproperty SecurityProperties.

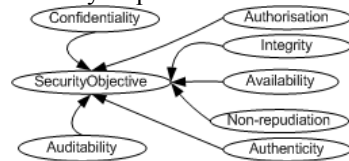


Figure 3. Class SecurityObjective

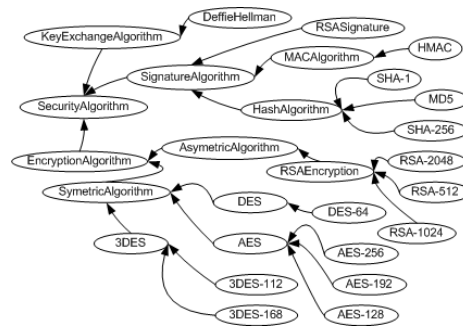


Figure 4. Class SecurityAlgorithm

Some security functions with security objectives. The table below lists some security functions and security goals that are used in our approach.

shared(K,P,Q) - Key K is shared between P & Q.	Confidentiality
key_algorithm(K,A,S) - K's algorithm A and standard S.	Confidentiality
key_size(K,L) - Key K has a length L.	Confidentiality
key_generated(K) - Key K is generated.	Confidentiality
key_distribution(K,A,S) - K's distribution algorithm is A and standard is S.	Confidentiality, Integrity
believes_shared(P,Q,X) - P believes that it shares X with Q.	Confidentiality
key_exchanged(K,P,Q) - K is exchanged between P and Q.	Confidentiality

5. An example

In this section, we focus on the WS based email system, discuss and evaluate the required and ensured security properties of WS associated with the security functionality get_shared_key(KEY, ID1, ID2) and explain the conditions when WS can invoke other WS. Using OWL-S and E-SCL, we describe the mechanisms of getting keys and key distribution between three WS: Client1 (C1), Client2 (C2), Key Distribution (KD), as illustrated in Figure 5. To invoke C2 and send a message, C1 has to get a shared key from KD. However, KD has the security requirements (required security properties) and the capabilities (ensured security properties) that should satisfy the security requirements and capabilities of C1 and C2.



Figure 5. The part of the email system

KD generates a symmetric key and ensures that a key is distributed securely to both C1 and C2. A composition between any of two WS is shown with a solid-line connecting them. For a composition with KD for the functionality get_shared_key(KEY, C1, C2), the required and ensured security properties of C1 are stated below. To save space, we use both SCL and OWL-S annotations.

```

<properties:Secproperties rdf:ID="Properties1">
key_exchanged( $K^{C2,C1}$ ,KD,C1,<9am,5pm>,[0.85,1])←
key_distribution( $K^{C2,C1}$ ,IKE,Diffie-Hellman,<9am,5pm>,[0.8,0.9]).
believes_shared(C1,C2, $K^{C2,C1}$ ,<9am,5pm>,[0.9,1])←
shared( $K^{C2,C1}$ ,C1,C2,<9am,5pm>,[0.8,0.9]),
key_exchanged( $K^{C2,C1}$ ,KD,C1,<9am,5pm>,[0.8,0.9]).
</properties:Secproperties>
<profile:Profile rdf:about="#Client1">
<profile:serviceName>Client1</profile:serviceName>
<profile:textDescription>WS Client1</profile:textDescription>
<securityProperties rdf:resource="#Properties1"/>
</profile:Profile>

```

IKE and Diffie-Hellman [23] are applied to key distribution and are utilized in SSL, Cisco encrypting routers etc. The security properties of C1 specify that the key is securely exchanged between KD and C1 from 9am to 5pm with probability 85%-100% if the key is distributed between C1 and KD during working hours from 9am to 5pm with probability 80%-90% using the stated method and algorithm. In the next rule, C1 believes the key is shared between C1 and C2 during working hours with the probability 90%-100% if the key is shared between C1 and C2 and it is exchanged according to the specified key distribution method and protocol from 9am to 5pm with probabilities 80%-90%. The security properties of KD regarding the functionality get_shared_key are described as:

```

<properties:Secproperties rdf:ID="Properties1">
key_algorithm( $K^{C2,C1}$ ,DES,S/MIME).
key_size( $K^{C2,C1}$ ,56).
key_generated( $K^{C2,C1}$ )←
key_algorithm( $K^{C2,C1}$ ,DES,S/MIME),key_size( $K^{C2,C1}$ ,56).
shared( $K^{C2,C1}$ ,C1,C2,[0.9,1])←key_generated( $K^{C2,C1}$ ).
if (bandwidth == low) {key_distribution( $K^{C2,C1}$ ,IKE,Diffie-
Hellman,<9am,5pm>,[0.9,1])}
else {key_distribution( $K^{C2,C1}$ ,DES,IPSec,<9am,5pm>,[0.9,1])}.
</properties:Secproperties>
<profile:Profile rdf:about="#KeyDistributor">
<profile:serviceName>KeyDistributor</profile:serviceName>
<profile:textDescription>Key distributor</profile:textDescription>
<securityProperties rdf:resource="#Properties1"/>
</profile:Profile>

```

The property key_algorithm($K^{C2,C1}$,DES,S/MIME) states the cryptographic key is generated with the data encryption algorithm DES and S/MIME (Secure / Multipurpose Internet Mail Extensions) [4,23]. It is ensured with the functions key_size and key_generated that a satisfactory key is generated for C1 and C2 if the algorithm DES is used, and the key size is 56 bit. KD ensures these required properties by itself. The property shared($K^{C2,C1}$,C1,C2,[0.9,1]) is ensured if the required property key_generated is satisfied. Probabilities 75%-100% and 90%-100% show that keys may be not delivered to a requestor WS during key sharing because it is known from traffic patterns that the network may be overloaded from 9am till 5pm.

The security properties of C2 for the functionality get_shared_key(KEY,C2,C1) are similar to those of C1, and are specified as follows:

```

<properties:Secproperties rdf:ID="Properties1">
key_exchanged( $K^{C2,C1}$ ,KD,C2,<9am,5pm>,[0.85,1])←
key_distribution( $K^{C2,C1}$ ,IKE,Diffie-Hellman,<9am,5pm>,[0.8,0.9]).
believes_shared(C2,C1, $K^{C2,C1}$ ,<9am,5pm>,[0.9,1])←
shared( $K^{C2,C1}$ ,C2,C1,<9am,5pm>,[0.8,0.9]),
key_exchanged( $K^{C2,C1}$ ,C2,C1,<9am,5pm>,[0.8,0.9]).
</properties:Secproperties>
<profile:Profile rdf:about="#Client2">
<profile:serviceName>Client2</profile:serviceName>
<profile:textDescription>WS Client2</profile:textDescription>
<securityProperties rdf:resource="#Properties1"/>
</profile:Profile>

```

The contract (CsC) between C2 and KD is established because their required and ensured properties match. The property key_distribution of KD is valid from 9am to 5pm with probability 90%-100% while C2 requires lower probability 80%-90% for this time period. The CsC between C1 and KD is similar.

```

CsC(C2,KD)←believes_shared(C2,C1, $K^{C2,C1}$ ,<9am,5pm>,[0.9,1]).
CsC(C1,KD)←believes_shared(C1,C2, $K^{C2,C1}$ ,<9am,5pm>,[0.9,1]).

```

The CsC ensures confidentiality, which means that other entities cannot see the value of the shared key, and integrity, which states that the value of the shared key cannot be changed or tampered. For CsC(C1,KD), we have the following security goals:

```

confidentiality( $K^{C2,C1}$ )←CsC(C1,KD).
integrity( $K^{C2,C1}$ )←CsC(C1,KD).

```

The security goals for CsC(C2,KD) are similar to CsC(C1,KD) outlined above. As it has been mentioned, WS can invoke other WS only if security properties are satisfied.

6. Related work

As described above, our work enhances SCL [12,13] that utilizes the Common Criteria [3], BAN logic [2] and logic programming [5], and involves time points, time intervals, time sequences, probability, and alternative security properties.

Probabilistic logic programming (PLP) is studied in [16] where clauses are extended by a subinterval of [0,1]. An overview of modal logic, temporal logic, and interval logic is presented in [20]. Time intervals are studied in [9] where a temporal logic based on 12 time intervals is developed. A methodology for modelling security system architectures and for analysing them is developed in [6], where the concept of security constraint patterns is also presented. The problem of specifying and analysing dynamic software architectures is also studied in [1,8]. The security ontology introduced in [15] is partially based on [7] and describes types of security information including security mechanisms, protocols, algorithms, objectives, and credentials. It uses OWL [21] and OWL-S [22] and is applied to a SOA (Service Oriented Architecture) to demonstrate how WS can publish their security requirements and capabilities. However, it does not concern dynamic security properties of WS,

i.e. security properties which can change after certain events or periods of time predictably or unpredictably.

7. Conclusion and future work

To create secure Web service based systems (WSBS), we need to know the security properties of its constituent Web services (WS) which are “black-boxes” implemented by third-party software vendors. The security properties should be published externally and the security capabilities of a requestor WS, which needs to invoke another WS, should satisfy the security requirements of this WS and vice versa. Also, the security properties should be capable of describing the dynamic nature of WSBS. In this paper, we have presented our approach to composing secure and dynamic WSBS. In future work, we will investigate using SWRL [25] to describe and automatically analyse the dynamic security properties of WS. Being inspired by different biological systems [14] and autonomic computing principles [11], we will develop the attack/defence WS ontology and security reconfiguration patterns that will allow WS with “strong” security properties to protect WS with “weak” security properties.

8. References

- [1] R. Allen, R. Douence, and D. Garlan, “Specifying and analyzing dynamic software architectures”, *FASE '98*, pp. 21-37, Lisbon, Portugal, March 28 - April 4, 1998.
- [2] M. Burrows, M. Abadi, R. Needham, “A logic of authentication”, *ACM Transactions Computer Systems*, pp. 18-36, February 1990.
- [3] Common Criteria. <http://csrc.nist.gov/cc/CC-v2.1.html>.
- [4] Cryptography. <http://www.ssh.fi/support/cryptography/>.
- [5] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, “Complexity and Expressive Power of Logic Programming”, *CSUR*, Vol.33 Issue 3, pp. 374-425, September 2001.
- [6] Y. Deng, J. Wang, J. J.P. Tsai, and K. Beznosov, “An Approach for Modeling and Analysis of Security System Architectures”, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, no. 5, pp. 1099-1119, 2003.
- [7] G. Denker, S. Nguyen, and A. Ton, “OWL-S Semantics of Security Web Services: a Case Study”, *SRI International*, Menlo Park, California, USA, 2004.
- [8] I. Georgiadis, "Self-Organising Distributed Component Software Architectures," *PhD Thesis*, Imperial College, 2002.
- [9] J. Y. Halpern and Y. Shoham, “A Propositional Modal Logic of Time Intervals”, *Journal of the Association for Computing Machinery*, Vol 38. No 4, pp 935-962, October 1991.
- [10] KAoS. <http://www.ihmc.us/research/projects/KAoS/>.
- [11] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing", *IEEE Computer*, pp. 41-50, January 2003.
- [12] K. Khan and J. Han, “A Security Characterisation Framework for Trustworthy Component Based Software Systems”, *COMPSAC2003*, pp. 164-169, 2003.
- [13] K. Khan, “Security Characterisation and Compositional Analysis for Component-based Software Systems”, *PhD thesis*, Monash University, April 2005.
- [14] T. R. De Kievit and B. H. Iglewski, “Bacterial Quorum Sensing in Pathogenic Relationships”, *Infection and Immunity*, 68:4839-4849, 2000.
- [15] A. Kim, J. Luo, and M. Kang, “Security Ontology for Annotating Resources”, *In 4th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2005)*, Agia Napa, Cyprus, 2005.
- [16] T. Lukasiewicz, “Probabilistic Logic Programming with Conditional Constraints”, *ACM Trans. on Computational Logic*, Vol. 2, No. 3, pp. 289-339, July 2001.
- [17] METEOR-S. <http://lsdis.cs.uga.edu/projects/meteor-s/>.
- [18] I. Niemelaa and P. Simons, “System smodels”, *Technical report*, Universitat Koblenz, 2000.
- [19] T. O'Reilly, “What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software”, 09/30/2005. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
- [20] M. A. Orgun and W. Ma, “An Overview of Temporal and Modal Logic Programming”, *ICTL'94*, pp.445-479, 1994.
- [21] OWL. <http://w3.org/TR/owl-features/>.
- [22] OWL-S. <http://www.daml.org/services/>.
- [23] RSA Laboratories website. <http://www.rsasecurity.com>.
- [24] A. Stamos, "Attacking Web Services", SyScan05.
- [25] SWRL. <http://www.w3.org/Submission/2004/03/>.
- [26] SWSL. <http://www.daml.org/services/swsl/>.
- [27] WSDL-S: Adding semantics to WSDL - White paper. <http://lsdis.cs.uga.edu/library/download/wSDL-s.pdf>.
- [28] WSMO. <http://www.wsmo.org/>.
- [29] H. Zhuge, “China's E-Science Knowledge Grid Environment”, *IEEE Intelligent Systems*, 19 (1) (2004) 13-17.