Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 839–845
ISBN 978-83-60810-27-9
ISSN 1896-7094

A Security Model for Personal Information Security Management Based on Partial Approximative Set Theory

Zoltán Csajbók
Department of Health Informatics
Faculty of Health, University of Debrecen
Sóstói út 2-4, H-4400 Nyíregyháza, Hungary
Email: csajzo@de-efk.hu

***Abstract*—Nowadays, computer users especially run their applications in a complex open computing environment which permanently changes in the *running* time. To describe the behavior of such systems, we focus *solely* on externally observable execution traces generated by the observed computing system. In these extreme circumstances the pattern of sequences of primitive actions (execution traces) which is observed by an external observer cannot be designed and/or forecast in advance. We have also taken into account in our framework that security policies are partial-natured.**

To manage the outlined problem we need tools which are approximately able to discover secure or insecure patterns in execution traces based on *presupposes* of computer users. Rough set theory may be such a tool. According to it, the vagueness of a subset of a finite universe U is defined by the difference of its lower and upper approximations with respect to a partition of the universe U. Using partitions, however, is a very strict requirement. In this paper, our starting point will be an *arbitrary* family of subsets of U. Neither that this family of sets covers the universe nor that the universe is finite will be assumed. This new approach is called the *partial approximative set theory*. We will apply it to build up a new security model for distributed software systems solely focusing on their externally observable executions and to find out whether the observed system is secure or not.

I. Introduction

NOWADAYS, computer end users especially run their applications in a complex open computing environment which as a rule consists of software components of finite numbers. Each component has an individual behavior, and the global behavior of the whole system is a collection of individual ones.

Personal users watch their applications, work with one of them, and, in general, also follow details of other applications with attention. Moreover, the open computing environment permanently changes in the *running* time. Consequently, the pattern of the sequence of primitive actions, which is observed by an external observer, cannot be designed and/or forecast in advance. Nevertheless, from an execution trace which is actually observed during a short observation time interval someone ought to decide whether the system works safely or not, of course, with respect to a certain security specification.[1]

Under these peculiar circumstances, organizations first of all have to understand what has to be protected, and why. The answers determine the choice of the organization's *security strategy* which is, in turn, expressed in *security policies* [6]. Security policies *prescribe* and *proscribe* behaviors of software systems specifying acceptable and unacceptable execution traces of applications.

In corporate information security management there have been many approaches for security policy specification. Traditionally, security policies are formulated along the so-called CIA taxonomy [18] which sees security as the combination of three attributes—confidentiality, integrity, and availability.

However, there are different challenges between corporate information security management and personal information security management. Under the personal information security we do not exclusively mean the privacy. Under the information security from personal point of view we also mean, among others, the protection of personal desktop PCs or notebooks from the intrusion, applications and data from the damage, etc.

Users in personal computing environment are inundated by recommendations how they should use and operate their system. In headwords only: strong password creation tips and maintenance, virus protection, software downloading and installation, removable media risks, encryption and cryptographic means, system backups, incident handling, e-mail and internet use best practices, etc. Non-professionals, of course, cannot convert these pieces of good advice into security policies, particularly into a formal one.

Arising from the human thinking [26], all computer users have *anticipated hypotheses* how an application or the whole computer system should or should not work. This may range from informal expected behaviors of the system, their elements might be called expected 'milestones', to more formal ones described in user manuals. Without any knowledge about

[1]This strange situation is smartly described by Schneier: "You have to imagine an intelligent and malicious adversary inside your system (the 'Satan' of Satan's computer), constantly trying new ways to subvert it. You have to think like an alien." ([3], from the Foreword by B. Schneier)

running application or having either some informal behaviors and/or more formal descriptions about it, we have to make an attempt to build up an information security management model which is approximately able to discover secure or insecure patterns in execution traces of the running system. Thus, in contrast to the traditional approach mentioned above, in order to describe the behavior of distributed software systems in a personal computing environment, we focus *solely* on externally observable executions generated by the observed computing system.

As usual, the software components can operate with each other. Their interconnections may be intended or *ad hoc*. Notice, however, that in both cases, the mechanisms of these interconnections mostly remain concealed from the external observers. In particular, based on only external observations, we cannot model these synchronization mechanisms.

II. Related Work

Our approach partly relates to the theory of a very special class of security policies called *properties* proposed by Lamport [20]. Informally, a security property is a set of execution traces which is defined exclusively in terms of individual execution traces, or, in other words, a property may not specify a relationship between possible execution traces of an application [21]. Access control or availability are properties, while information flow policy is typically not, the latter is a more general one. According to another terminology, properties are *point-wise* in nature, while the information flow policy is *point-free* in nature. Methods for specifying and reasoning about properties are well understood. Alpern and Schneider have shown [1] that every property is the intersection of a so-called *safety* and a *liveness* property. For more details about properties see, e.g., [20] [30], [21], [22].

We also partly relate to the theory of *hyperproperties* proposed by Clarkson and Schneider in [7]. In this approach *every* set of execution traces is a property. Thereupon, the hyperproperty is a set of properties, or simply, a set of sets of execution traces. In other words, both properties and hyperproperties are defined in a *point-wise* manner—properties based on their elements, the execution traces, and hyperproperties based on their elements, the properties.

Analogous to safety and liveness property, Clarkson and Schneider defined the notions of *hypersafety* and *hyperliveness*. They have proved that every hyperproperty is the intersection of a hypersafety and a hyperliveness [7]. This generalizes Alpern and Schneider's result about properties.

Both theories of properties and hyperproperties have a characterization in terms of topology. In the Plotkin topology on properties, safety and liveness correspond to *closed* and *dense* sets, respectively [1]. In [7], this topological characterization is generalized to hyperproperties, showing that hypersafety and hyperliveness also correspond to closed and dense sets in the lower Vietoris topology which is a construction on Plotkin topology [31], [32].

It is an interesting question of the relation between the traditional approaching to the security policies based on CIA taxonomy and the formulation of them based on properties or hyperproperties. This is an open question. Clarkson and Schneider [7] think that the language of confidentiality, integrity, and availability is orthogonal to hypersafety and hyperliveness. Whereas Benenson et al. worked out a framework which, in their opinion, offers a new perspective onto the CIA taxonomy [5]. Namely, to some extent, confidentiality has a large overlap with information flow, integrity has parallels with safety, and availability has some resemblance to liveness.

However, our approach to hyperproperties as sets of sets of execution traces is different from the Clarkson and Schneider's one [7]. Namely, in contrast to their notion, we do not consider *every* set of execution traces as a property. Our proposal is characteristically a *point-free* approach. We deal with the set of execution traces *per se*, and the same is true for hyperproperties. Of course, in some special cases, a point-free feature can coincide with a feature defined in a point-wise manner. Avoiding the ambiguities, in our framework we will not use the terms 'property' and 'hyperproperty'.

We model distributed software systems as semantic system model, so-called *traced-based model*. A traced-based model describes the behavior of a system as a set of execution traces where a trace models a possible execution sequence of the whole system or its some components. To our notion the Mazurkiewicz' trace theory stands the nearest [23], [13]. We, however, at least temporarily, use it in a very simple form.

III. Problem Statement

We briefly sum up our framework informally as follows.

We model security policies as sets of sets of execution traces. We also have taken into account in our framework that security policies are partial-natured. Typically some policies may apply only to a specific application or type of information. For example, the average response time policy would be practical to be applied to the whole observed system. But, in an open computing environment, we *cannot* influence the response time of unknown applications, such as a database's query via the internet. As another example, probably it is enough to enforce the information flow policy on such software processes which handle confidential information.

To manage the outlined problem we need tools which are approximately able to discover secure or insecure patterns in execution traces. By applying some sort of knowledge discovery method it is possible to some extent to reveal the secure or insecure nature of the observed system. However, the result, obtained from the execution traces in this way, cannot be precise (exact) due to the permanent changes of the open computing environment and insufficient knowledge about it.

Rough set theory is a relatively new data-mining technique used in the discovery of patterns within data. This theory provides a powerful foundation to reveal and discover important structures in data and to classify complex objects. One of the main advantages of rough set theory is that it does not need any preliminary or additional information about data.

The rough set theory was introduced by the Polish mathematician, Z. Pawlak in the early 1980s [27], [28]. It was a

new mathematical approach to *vagueness* [29]. According to Pawlak's idea, the vagueness of a subset of a finite universe U is defined by the difference of its upper and lower approximations with respect to a partition of the universe U.

Using partitions, however, is a very strict requirement. In this paper, our starting point will be an *arbitrary* family of subsets of an *arbitrary* set U. Neither that this family of sets covers the universe nor that the universe is finite will be assumed. Within this new framework, our concepts of lower and upper approximations are straightforward *point-free* generalizations of Pawlak's ones [9]. This is called the *partial approximative set theory*. This new approach suits the partial nature of security policies, as well. We will apply this theory to build up a new security model for distributed software systems *solely* focusing on their externally observable executions and to find out whether the observed system is secure or not.

The rest of the paper is organized as follows. In Section IV we summarize the basic notations. Section V will outline a general approximation framework. Sections VI and VII briefly present the basic principles of the rough set theory and the partial approximative set theory, respectively. In Section VIII we will describe the behavior of distributed software systems by means of partial approximative set theory.

IV. Basic Notations

Let U be any set. Let $\mathfrak{A} \subseteq 2^U$ be a family of sets of which elements are subsets of U. The union of $\mathfrak{A}$ is $\bigcup\mathfrak{A} = \{x \mid \exists A \in \mathfrak{A}(x \in A)\}$, and the intersection of $\mathfrak{A}$ is $\bigcap\mathfrak{A} = \{x \mid \forall A \in \mathfrak{A}(x \in A)\}$. If $\mathfrak{A}$ is an empty family of sets we define $\bigcup\emptyset = \emptyset$ and $\bigcap\emptyset = U$.

The number of elements of the set U is denoted by $\#U$.

If ϵ is an arbitrary binary relation on U, let $[x]_\epsilon$ denote the ϵ-related elements to x, i.e., $[x]_\epsilon = \{y \in U \mid (x, y) \in \epsilon\}$.

Let $\mathcal{A}$ be a finite set of *symbols* called the *alphabet*. A *string* is a finite or infinite sequence of symbols chosen from $\mathcal{A}$. We denote strings by small Greek letters $\sigma, \varrho, \tau, \ldots$, with possible sub- or superscripts. String containing no symbols is called the *empty string* and is denoted by λ.

If $\sigma = a_1a_2\ldots a_n$ is a finite string, $n = |\sigma| = |a_1a_2\ldots a_n|$ is called the *length* of σ. In particular, $|\lambda| = 0$.

Let $\mathcal{A}^*$ and $\mathcal{A}^\omega$ denote the set of all finite and infinite strings made up of symbols chosen from $\mathcal{A}$, respectively. We also use the following notations: $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\lambda\}$, $\mathcal{A}^\infty = \mathcal{A}^* \cup \mathcal{A}^\omega$.

For any finite strings $\sigma_1 = a_1a_2\ldots a_n$, $\sigma_2 = b_1b_2\ldots b_m$, the *concatenation* $\sigma_1 \circ \sigma_2 = a_1a_2\ldots a_n \circ b_1b_2\ldots b_m$ of σ_1 and σ_2 is the string $\sigma_1\sigma_2 = a_1a_2\ldots a_nb_1b_2\ldots b_m$. If $\sigma_1 \in \mathcal{A}^*$, $\sigma_2 \in \mathcal{A}^\omega$ then $\sigma_1 \circ \sigma_2 = \sigma_1\sigma_2$ is also defined. The empty string is the identity for concatenation, that is for any string $\sigma_1 \in \mathcal{A}^*, \sigma_2 \in \mathcal{A}^\omega$, we have $\lambda\sigma_1 = \sigma_1\lambda = \sigma_1$ and $\lambda\sigma_2 = \sigma_2$.

A symbol $a \in \mathcal{A}$ *occurs* in string $\sigma \in \mathcal{A}^\infty$, if $\sigma = \sigma_1a\sigma_2$ for some strings $\sigma_1 \in \mathcal{A}^*, \sigma_2 \in \mathcal{A}^\infty$.

Let $\sigma \in \mathcal{A}^*$, $\tau \in \mathcal{A}^\infty$ be two strings. σ is a *prefix* (initial part) of τ if there exists $\rho \in \mathcal{A}^\infty$ such that $\tau = \sigma\rho$. The prefix is *proper* if $\rho \neq \lambda$ and *nontrivial* if $\sigma \neq \lambda$.

Given $L \subseteq \mathcal{A}^\infty$, let $\mathtt{pref}(L)$ denote the set of all prefixes of all strings in L. Clearly, $L \subseteq \mathtt{pref}(L)$. In particular, $\mathtt{pref}(\{\sigma\}) = \mathtt{pref}(\sigma)$ is the set of all prefixes of the string $\sigma \in \mathcal{A}^\infty$. Note that $\mathtt{pref}(L) = \bigcup\{\mathtt{pref}(\sigma) \mid \sigma \in L\} \subseteq \mathcal{A}^*$.

Let $\mathcal{A}$ be an alphabet and σ be a finite string over an arbitrary alphabet. Then $\pi_\mathcal{A}(\sigma)$ denotes the (string) *projection of σ onto $\mathcal{A}$* defined as follows:

$$\pi_\mathcal{A}(\sigma) = \begin{cases} \lambda, & \text{if } \sigma = \lambda; \\ \pi_\mathcal{A}(\sigma'), & \text{if } \sigma = \sigma' a \wedge a \notin \mathcal{A}; \\ \pi_\mathcal{A}(\sigma')a, & \text{if } \sigma = \sigma' a \wedge a \in \mathcal{A}. \end{cases}$$

Informally, a projection onto $\mathcal{A}$ cleans strings of all symbols not in $\mathcal{A}$. $\pi_\mathcal{A}(\sigma) \neq \lambda$, if at least one symbol $a \in \mathcal{A}$ occurs in string σ.

Similarly, if $\Sigma \subseteq \mathcal{A}^*$, $\pi_\mathcal{A}(\Sigma) = \{\pi_\mathcal{A}(\sigma) \mid \sigma \in \Sigma\}$. $\pi_\mathcal{A}(\Sigma) \neq \{\lambda\}$, if in at least one string in $\sigma \in \Sigma$, at least one symbol $a \in \mathcal{A}$ occurs in σ.

V. A General Approximation Framework

In order that the vagueness can be treated in a general approximate framework, let our initial concept be the following. A pair of maps $f, g : 2^U \to 2^U$ is a *weak approximation pair on U* if

$$\forall X \in 2^U\ (f(X) \subseteq g(X)).$$

As Düntsch and Gediga noticed in [14], this constraint seems to be the weakest condition for a sensible concept of approximations of subsets in U.

The maps f, g is a *strong approximation pair on U* if each subset $X \in 2^U$ is bounded by $f(X)$ and $g(X)$ [14], i.e.,

$$\forall X \in 2^U\ (f(X) \subseteq X \subseteq g(X)).$$

In [26], a new hypothesis about approximation was drawn up recently. According to this assumption, the notion of "approximation" may be mathematically modelled by the notion of Galois connections. From now on, we call it the *approximation hypothesis*.

Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be two posets.

Definition 1. A pair (f, g) of maps $f : P \to Q$, $g : Q \to P$ is a *(regular) Galois connection* or an *adjunction* between P and Q if

$$\forall p \in P\, \forall q \in Q\ (f(p) \leq_Q q \Leftrightarrow p \leq_P g(q)).$$

f is called the *lower adjoint* and g the *upper adjoint* of the Galois connection.

We also write (P, f, g, Q) for a whole Galois connection. If $P = Q$ it is said (P, f, g, P) is a Galois connection on P.

Remark 2. Here we adopted the definition of Galois connection in which the maps are monotone. It is also called monotone or covariant form. For more details, see, e.g., [11], [12], [15], [17].

VI. Fundamentals of Rough Set Theory

The basic concepts and properties of rough set theory can be found, e.g, in [28], [19]. Here we cite only a few of them which will be important in what follows. We partly restate these well-known facts on the language of approximations.

Definition 3. A pair (U, ε), where U is a finite universe of discourse and ε is an equivalence relation on U, is called *Pawlak's approximation space*.

A subset $X \subseteq U$ is *ε-definable*, if it is a union of ε-elementary sets, otherwise X is *ε-undefinable*. By definition, the empty set is considered to be an ε-definable set.

Let $\mathfrak{D}_{U/\varepsilon}$ denote the family of ε-definable subsets of U.

In Pawlak's approximation spaces, the lower and upper approximations of X can be defined in two equivalent forms, namely, in a *point-free* manner—based on the ε-elementary sets, and in a *point-wise* manner—based on the elements.

Definition 4. Let a Pawlak's approximation space (U, ε), a subset $X \in 2^U$ be given. The *lower ε-approximation* of X is

$$\begin{aligned} \underline{\varepsilon}(X) &= \bigcup\{Y \mid Y \in U/\varepsilon, Y \subseteq X\} \\ &= \{x \in U \mid [x]_\varepsilon \subseteq X\}, \end{aligned}$$

and the *upper ε-approximation* of X is

$$\begin{aligned} \overline{\varepsilon}(X) &= \bigcup\{Y \mid Y \in U/\varepsilon, Y \cap X \neq \emptyset\} \\ &= \{x \in U \mid [x]_\varepsilon \cap X \neq \emptyset\}. \end{aligned}$$

The set $B_\varepsilon(X) = \overline{\varepsilon}(X) \setminus \underline{\varepsilon}(X)$ is the *ε-boundary* of X.

X is *ε-crisp*, if $B_\varepsilon(X) = \emptyset$, otherwise X is *ε-rough*.

Based on binary relations on U, lower and upper ε-approximations can be generalized via their *point-wise* definitions [19].

Definition 5. Let ϵ be an arbitrary binary relation on U and $X \in 2^U$. The *lower ϵ-approximation* of X is

$$\underline{\epsilon}(X) = \{x \in U \mid [x]_\epsilon \subseteq X\},$$

and the *upper ϵ-approximation* of X is

$$\overline{\epsilon}(X) = \{x \in U \mid [x]_\epsilon \cap X \neq \emptyset\}.$$

If ϵ^{-1} denotes the inverse relation of ϵ, in the same manner one can also define lower and upper ϵ^{-1}-approximations.

Theorem 6 ([19], Proposition 134)**.** *Let ϵ be an arbitrary binary relation on U. Then the pairs $(\overline{\epsilon}, \underline{\epsilon^{-1}})$ and $(\overline{\epsilon^{-1}}, \underline{\epsilon})$ are Galois connections on $(2^U, \subseteq)$.*

Some other properties of lower and upper ϵ-approximations are expressed by some properties of binary relations, and vice versa.

Theorem 7. *Let ϵ be an arbitrary binary relation on U.*

1) *The pair $(\underline{\epsilon}, \overline{\epsilon})$ is a weak approximation pair if and only if ϵ is connected.*
2) *The pair $(\underline{\epsilon}, \overline{\epsilon})$ is a strong approximation pair if and only if ϵ is reflexive.*
3) *The pair $(\overline{\epsilon}, \underline{\epsilon})$ is a Galois connection on $(2^U, \subseteq)$ if and only if ϵ is symmetric.*

Proof: In [19]. *1.* Proposition 136., *2.* Proposition 137., *3.* Proposition 138. ∎

It can be shown that even if the relation ϵ is symmetric, it is not sufficient that the lower and upper ϵ-approximations defined in a *point-free* manner form a Galois connection [10].

VII. Fundamentals of Partial Approximative Set Theory

In practice there are attributes which do not characterize all members of an observed collection of objects. A very simple example, when one investigates an infinite set via a finite family of its finite subsets. For instance, a number theorist studies regularities of natural numbers using computers.

Throughout this section let U be any non-empty set.

Definition 8. Let $\mathfrak{B} \subseteq 2^U$ be a non-empty family of non-empty subsets of U called the *base system*. Its elements are the *$\mathfrak{B}$-sets*.

A family of sets $\mathfrak{D} \subseteq 2^U$ is *$\mathfrak{B}$-definable* if its elements are $\mathfrak{B}$-sets, otherwise $\mathfrak{D}$ is *$\mathfrak{B}$-undefinable*. A non-empty subset $X \in 2^U$ is *$\mathfrak{B}$-definable* if there exists a $\mathfrak{B}$-definable family of sets $\mathfrak{D}$ such that $X = \bigcup \mathfrak{D}$, otherwise X is *$\mathfrak{B}$-undefinable*. The empty set is considered to be a $\mathfrak{B}$-definable set.

Let $\mathfrak{D}_\mathfrak{B}$ denote the family of $\mathfrak{B}$-definable sets of U.

Definition 9. Let $\mathfrak{B} \subseteq 2^U$ be a base system and X be any subset of U. The *weak lower $\mathfrak{B}$-approximation* of X is

$$\mathfrak{C}^\flat_\mathfrak{B}(X) = \bigcup\{Y \mid Y \in \mathfrak{B}, Y \subseteq X\},$$

and the *weak upper $\mathfrak{B}$-approximation* of X is

$$\mathfrak{C}^\sharp_\mathfrak{B}(X) = \bigcup\{Y \mid Y \in \mathfrak{B}, Y \cap X \neq \emptyset\}.$$

Notice that $\mathfrak{C}^\flat_\mathfrak{B}$ and $\mathfrak{C}^\sharp_\mathfrak{B}$ are straightforward *point-free* generalizations of lower and upper ε-approximations.

Theorem 10 ([10], Theorem 4.5)**.** *Let the fixed base system $\mathfrak{B} \subseteq 2^U$ and maps $\mathfrak{C}^\flat_\mathfrak{B}$ and $\mathfrak{C}^\sharp_\mathfrak{B}$ be given.*

1) $\forall X \in 2^U (\mathfrak{C}^\flat_\mathfrak{B}(X) \subseteq \mathfrak{C}^\sharp_\mathfrak{B}(X))$.
2) $\forall X \in 2^U (\mathfrak{C}^\flat_\mathfrak{B}(X) \subseteq X)$*—that is, $\mathfrak{C}^\flat_\mathfrak{B}$ is contractive.*
3) $\forall X \in 2^U (X \subseteq \mathfrak{C}^\sharp_\mathfrak{B}(X))$ *if and only if* $\bigcup \mathfrak{B} = U$*—that is, $\mathfrak{C}^\sharp_\mathfrak{B}$ is extensive if and only if $\mathfrak{B}$ covers the universe.*

In other words, the pair of maps $\mathfrak{C}^\flat_\mathfrak{B}, \mathfrak{C}^\sharp_\mathfrak{B} : 2^U \to 2^U$ is a weak approximation pair on U, and it is a strong one if and only if the base system $\mathfrak{B}$ covers the universe.

Remark 11. If $\bigcup \mathfrak{B} \neq U$, then $\forall X \subseteq U \setminus \bigcup \mathfrak{B}\ \forall B \in \mathfrak{B}(X \cap B = \emptyset)$. Consequently, for all these subsets $\mathfrak{C}^\sharp_\mathfrak{B}(X) = \bigcup \emptyset = \emptyset$, i.e., the empty set is the weak upper $\mathfrak{B}$-approximation of certain non-empty subsets of U. This uncommon case may be interpreted so that our knowledge about the universe encoded in the base system $\mathfrak{B}$ is incomplete. This case may be excluded by a partial map called strong upper $\mathfrak{B}$-approximation. For more details, see [9].

Definition 12. Let the fixed base system $\mathfrak{B} \subseteq 2^U$ and maps $\mathfrak{C}^\flat_\mathfrak{B}$ and $\mathfrak{C}^\sharp_\mathfrak{B}$ be given. The quadruple $(U, \mathfrak{B}, \mathfrak{C}^\flat_\mathfrak{B}, \mathfrak{C}^\sharp_\mathfrak{B})$ is called a *weak $\mathfrak{B}$-approximation space*.

Theorem 13 ([10], Corollary 4.10)**.** *Let the weak $\mathfrak{B}$-approximation space $(U, \mathfrak{B}, \mathfrak{C}^\flat_\mathfrak{B}, \mathfrak{C}^\sharp_\mathfrak{B})$ be given.*

The pair of maps $(\mathfrak{C}^\sharp_\mathfrak{B}, \mathfrak{C}^\flat_\mathfrak{B})$ forms a Galois connection on $(2^U, \subseteq)$ if and only if the base system $\mathfrak{B}$ is a partition of the universe U.

VIII. The Security Model

An execution sequence consists of linearly ordered *observable* atomic actions. The manner in which execution sequences are represented is irrelevant, they may be finite sequences of primitive events, higher-level system steps, program states, state/action pairs, etc., or a mixture of all these [4], [30].

A. Types of Atomic Actions

Let $A_i = \{a_1, a_2, \ldots, a_{i_m}\}$ be a finite set of externally observable atomic actions of the i^{th} components of the distributed software system called the i^{th} *required component action set* (alphabet) ($i = 1 \ldots, n$). The required component action sets are not necessary pairwise disjoint, and the common atomic actions in different alphabets are undistinguishable. An *execution trace* (string) $\sigma \in A_i^{\infty}$ is a finite or infinite sequence of not necessarily different atomic actions.

Let A_{unsafe} be a finite set of insecure actions, called the *unsafe action set*, which may happen during the running time of the observed system.

$\bigcup A_i = \bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \ldots \cup A_n$ has to be augmented by a number of additional atomic actions which may or may not influence the safety of the system. Their set is called the *doubtful action set* and denoted by $A_{doubtful}$. For instance, usual interactions between end users and peripheral devices probably do not influence the safety of the system. But, too frequent interactions between them may indicate an unsafe behavior.

Note that the required component actions sets A_i are component specific, while the unsafe and doubtful action sets are not.

Let $\mathcal{A} = \bigcup A_i \cup A_{doubtful} \cup A_{unsafe}$, called the *(system) action set*. Let $\mathcal{A}_i = A_i \cup A_{doubtful} \cup A_{unsafe}$ denote the finite action set correspond to the i^{th} component called the i^{th} *component action set* ($i = 1, 2, \ldots, n$).

B. Anticipated Behaviors of Applications

We model the users' anticipated behaviors of a software component by a set of acceptable (feasible) finite linearly ordered sequences of the expected 'milestones' from A_i, in informal cases, or finite linearly ordered sequences of the required component actions corresponding to a given precedence dependency on A_i, in formal cases.

In latter case, let us suppose that the actions in a required component action set A_i have to be performed with respect to the ordering constraints called 'precedence' dependencies. Two actions are precedentially ordered if one has to precede the other specifying the precedential ordering such that starting a certain action depends on other actions being completed beforehand. For a simple example, one must enter into an application before quitting it. In symbols, $A_i = \{`enter\_into\_application', \ldots, `quit\_application'\}$ and $`enter\_into\_application' \leq `quit\_application'$. Of course, there may be many other precedences that must be obeyed.

We can represent these precedences by a set consisting of the pairs of type $(`before\_action', `after\_action')$. This situation is usually modelled as a *scheduling problem* which is a common source of partial orders. That is, the precedence dependency relation is reflexive, transitive, and antisymmetric, and renders the required action set A_i to a poset $(A_i, \leq)$. It is assumed that the precedence dependencies is configurable in this way and it is captured by the system architect in a precedential model.

In formal cases, let $\overrightarrow{A_i}$ denote the set of all topological sorts of the poset $(A_i, \leq)$. By the Szpilrajn's Theorem [33], $\#\overrightarrow{A_i} \geq 1$. An element of $\overrightarrow{A_i}$ can be represented as a finite execution $a_{i_1} a_{i_2} \ldots a_{i_m}$ of all elements of A_i in such a way that for all $i_k < i_l$, either $a_{i_k} \leq a_{i_l}$ or a_{i_k} and a_{i_l} are incomparable in $\leq$. This means that a topological sort in $\overrightarrow{A_i}$ is consistent with the precedence dependencies defined on A_i.

Similarly, in informal cases, let $\overrightarrow{A_i}$ denote simply the set of all acceptable (feasible) finite linearly ordered sequences of the expected 'milestones' from A_i.

Note that, $\overrightarrow{A_i} \subseteq A_i^*$, in both cases.

C. Modelling Software Systems and their Components

A distributed software system is modelled by a non-empty set of *infinite* execution traces over the system action set $\mathcal{A}$.

Definition 14. Let $\Sigma \subseteq \mathcal{A}^{\omega}$ be a non-empty set of infinite executions over the action set $\mathcal{A}$. Then, by a *distributed software system* we mean an $(\mathcal{A}, \Sigma)$ pair.

Similarly, a software component or application is a non-empty set of *infinite* execution traces over the i^{th} component action set $\mathcal{A}_i$.

Definition 15. Let $\Sigma_i \subseteq \mathcal{A}_i^{\omega}$ be a non-empty set of infinite executions over the action set $\mathcal{A}_i$. Then, by a *software component* or an *application* we mean an $(\mathcal{A}_i, \Sigma_i)$ pair ($i = 1, 2, \ldots, n$).

If a software system or its a component execution terminates, i.e., it is representable by finite execution trace, we represent it as an infinite execution trace by infinitely stuttering the empty action λ. The empty action λ represents the fact that the software system or its any application has not started yet, or their running has already finished.

D. Modelling Observations

An observation of a distributed software system is modelled by a non-empty set of *finite* execution traces over the system actions set $\mathcal{A}$.

Definition 16. Let $\Sigma^{obs} \subseteq \mathcal{A}^*$ be a non-empty finite set of finite execution traces over the system action set $\mathcal{A}$.

Then, by a *system observation* we mean an $(\mathcal{A}, \Sigma^{obs})$ pair.

Observations on components can be analogously defined over $\mathcal{A}_i$ ($i = 1, \ldots, n$).

Definition 17. Let $\Sigma_i^{obs} \subseteq \mathcal{A}_i^*$ be a non-empty finite set of finite executions traces over the component action set $\mathcal{A}_i$.

Then, by a *component observation* or an *application observation* we mean an $(\mathcal{A}_i, \Sigma_i^{obs})$ pair ($i = 1, 2, \ldots, n$).

In particular, Σ^{obs} and Σ_i^{obs} may consists of exactly one execution trace.

E. Modelling Security

According to the expected behavior of the system, an application is *acceptable* if it fully meets the requirements of the precedential model (prescription).

The component which contains at least one unsafe action is *unacceptable* (proscription). However, the transition from the 'acceptable' decision to the 'unacceptable' decision is a *vagueness* problem.

For the i^{th} component ($i = 1, \ldots, n$) we define

$$\begin{aligned} \mathfrak{S}_i &= \mathtt{pref}(\overrightarrow{A_i}) \\ &= \{\mathtt{pref}(\sigma_{i_j}) \mid \sigma_{i_j} \in \overrightarrow{A_i},\, i_j = 1, \ldots, \#\overrightarrow{A_i}\}, \end{aligned}$$

where $\mathtt{pref}(\sigma_{i_j})$ is the set of all prefixes of either a topological sort σ_{i_j} of the poset $(A_i, \leq)$ in formal cases or a linearly ordered sequence of the expected 'milestones' in informal cases.

Let $\mathfrak{S} = \bigcup_{i=1}^{n} \mathfrak{S}_i$. Clearly, $\mathfrak{S} \subseteq 2^{\mathcal{A}^*} \subseteq 2^{\mathcal{A}^*}$.

The sets in $\mathfrak{S} \subseteq 2^{\mathcal{A}^*}$ are not necessarily pairwise disjoint, and, in general, $\bigcup \mathfrak{S}$ does not cover $\mathcal{A}^*$. In other words, $\mathfrak{S}$ *is a base system over the universe* $\mathcal{A}^*$ *by the terminology of the partial approximative set theory*.

F. The Vagueness of Acceptability

Let an observation set of the i^{th} application Σ_i^{obs} be given.

If Σ_i^{obs} includes at least one finite execution trace over $\mathcal{A}_i$ in which at least one unsafe action occurs, i.e.,

$$\pi_{A_{unsafe}}(\Sigma_i^{obs}) \neq \{\lambda\},$$

the observed action set Σ_i^{obs} is *unacceptable*.

Let us assume, that the observed action set does not include unsafe action, i.e., $\pi_{A_{unsafe}}(\Sigma_i^{obs}) = \{\lambda\}$. By based on only this observation set Σ_i^{obs}, can the observed application be considered acceptable or not? This problem, as we also mentioned above, is a vagueness problem.

In order to answer this question, let us form the lower and upper approximations of $\mathtt{pref}(\Sigma_i^{obs})$ with respect to the base system $\mathfrak{S}$.

The lower $\mathfrak{S}$-approximation of $\mathtt{pref}(\Sigma_i^{obs})$ is

$$\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs})) = \bigcup\{S \in \mathfrak{S} \mid S \subseteq \mathtt{pref}(\Sigma_i^{obs})\},$$

and the upper $\mathfrak{S}$-approximation of $\mathtt{pref}(\Sigma_i^{obs})$ is

$$\mathfrak{C}_{\mathfrak{S}}^{\sharp}(\mathtt{pref}(\Sigma_i^{obs})) = \bigcup\{S \in \mathfrak{S} \mid S \cap \mathtt{pref}(\Sigma_i^{obs}) \neq \emptyset\}.$$

By means of lower and upper $\mathfrak{S}$-approximation, the set $\mathtt{pref}(\Sigma_i^{obs})$ of all prefixes of observed execution traces may be approximated, consequently, *the safety of the observed action set* Σ_i^{obs} *can be estimated to a certain degree, by the information encoded in* $\mathfrak{S}$.

$\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs}))$ and $\mathfrak{C}_{\mathfrak{S}}^{\sharp}(\mathtt{pref}(\Sigma_i^{obs}))$ can be interpreted as a security evaluation of the system working at the moment of the end of an observation time interval.

The execution traces in $\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs}))$ can be classified with certainty as members of $\mathtt{pref}(\Sigma_i^{obs})$. Formally, all elements of $\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs}))$ suit an element in $\mathtt{pref}(\overrightarrow{A_i})$, or, in other words, a prefix of an execution trace in $\overrightarrow{A_i}$. Its elements can be interpreted as certainty safe execution traces of the observation set Σ_i^{obs} with respect to $\mathfrak{S}$, i.e., they fully correspond to the users' expectations.

$\mathtt{pref}(\Sigma_i^{obs}) \setminus \mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs}))$ may include, e.g., action sequences that miss some required actions or contain required actions in wrong order. It may also include right ordered action sequences which, however, contain doubtful actions between two required actions.

The execution traces in $\mathfrak{C}_{\mathfrak{S}}^{\sharp}(\mathtt{pref}(\Sigma_i^{obs}))$ are not guaranteed that all of them are members of $\mathtt{pref}(\Sigma_i^{obs})$. It may happen, e.g., that an execution sequence for a while suits a prefix of an execution trace in $\overrightarrow{A_i}$, but the next action is not a required one, it may be missing or it is a doubtful one.

By the partial feature of the base system $\mathfrak{S}$, it may take place that $\mathtt{pref}(\Sigma_i^{obs}) \not\subseteq \mathfrak{C}_{\mathfrak{S}}^{\sharp}(\mathtt{pref}(\Sigma_i^{obs}))$. It may emphasize that our knowledge about the system encoded in the base system $\mathfrak{S}$ is not enough to approximate $\mathtt{pref}(\Sigma_i^{obs})$.

The situation $\mathtt{pref}(\Sigma_i^{obs}) \subseteq \mathfrak{C}_{\mathfrak{S}}^{\sharp}(\mathtt{pref}(\Sigma_i^{obs}))$ can be interpreted so that all observed execution traces for a while suit a prefix of an execution trace in $\overrightarrow{A_i}$, but some of them has a wrong continuation or the next action is a doubtful one.

Of course, $\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs})) \subseteq \mathtt{pref}(\Sigma_i^{obs})$. If $\mathfrak{C}_{\mathfrak{S}}^{\flat}(\mathtt{pref}(\Sigma_i^{obs})) = \mathtt{pref}(\Sigma_i^{obs})$, it means that all elements of the observation set Σ_i^{obs} can be viewed as safe execution sequences with respect to the users' expectations.

IX. Conclusions and Future Work

We have shown a framework in which many questions corresponding to security features of distributed software systems can be represented uniformly.

The presented model is a theoretical outline. The next most important task is to work out a partial approximative information system model analogous to Pawlak's one [28]. This will enable to set up models which can be applied to solving practical problems. Such a model could be a base on which expert systems for enforcement of security policies can be built up.

Theorem 13 shows that the pair of weak lower and upper $\mathfrak{B}$-approximations forms a Galois connection only in a very special case. This restriction can be partly exceeded by using the notion of partial Galois connection [24].

Acknowledgment

The author would like to thank Tamás Mihálydeák for valuable discussions and suggestions.

References

[1] Alpern, B.," Schneider, F., *Defining liveness*. Inf. Process. Lett. 21, 4 (Oct.), 181–185, 1985.

[2] Alpern, B.," Schneider, F., *Recognizing safety and liveness*. Distributed Computing, 2:117–126, 1987.

[3] Anderson, R., *Security Engineering: A Guide to Building Dependable, Distributed Systems*. First edition. Wiley Computer Publishing, 2001.

[4] Bauer, L.," Ligatti, J.," Walker, D., *More enforceable security policies.* In Cervesato, I. (Ed.), Foundations of Computer Security: Proceedings of the FLoC'02 workshop on Foundations of Computer Security (2002), pp. 95–104.

[5] Benenson, Z.," Freiling, F.C.," Holz, Th.," Kesdogan, D.," Penso, L.D.: *Safety, Liveness, and Information Flow: Dependability Revisited*. In: Karl, W., Becker, J., Gropietsch, K.E., Hochberger, Ch., Maehle, E. (eds.) ARCS 2006—19th International Conference on Architecture of Computing Systems, Workshops Proceedings, pp. 56–65. Frankfurt am Main, Germany (2006)

[6] Caelli, W.," Longley, D.," Shain, M., *Information security handbook*, Stockton Press, New York, 1991.

[7] Clarkson, M. R.," Schneider, F.B., *Hyperproperties*, Proceedings 21st IEEE Computer Security Foundations Symposium (Pittsburgh, PA, June 2008), IEEE Computer Society (2008), pp. 51–65.

[8] Cousot, P.," Cousot, R., *Abstract interpretation and application to logic programs*, Journal of Logic Programming, 13(2–3):103–179, 1992. http://www.di.ens.fr /∼cousot/COUSOTpapers/JLP92.shtml

[9] Csajbók, Z., *Partial Approximative Set Theory*, In Programs, Proofs, Processes, Sixth Conference on Computability in Europe, CiE 2010, Ponta Delgada (Azores), Portugal, June 30 - July 4, 2010, Abstract and Handout Booklet, 2010. To appear.

[10] Csajbók, Z., *Partial Approximative Set Theory: A View from Galois Connections*, In Kovács, Emöd (ed.) et al., Proceedings of the 8th international conference on applied informatics (ICAI 2010), January 27–30, 2010, Eger, Hungary. Eger: Eszterházy Károly College. To appear.

[11] Davey, B. A.," Priestley, H. A.: *Introduction to Lattices and Order*, Second edition, Cambridge University Press, Cambridge, 2002.

[12] Denecke, K.," Erné, M.," Wismath, S.L., *Galois Connections and Applications*, Kluwer Academic Publishers, Dordrecht, London, Boston, 2004.

[13] Diekert, V.," Rozenberg, G., *The Book of Traces*, World Scientific, Singapore, 1995.

[14] Düntsch, I.," Gediga, G., *Approximation Operators in Qualitative Data Analysis*, In: H. de Swart et al. (Eds.): TARSKI, Lecture Notes in Computer Science Vol. 2929, Springer-Verlag, Berlin, Heidelberg, 214–230, 2003.

[15] Erné, M.," Koslowski, J.," Melton, A.," Strecker, G. E., *A primer on Galois connections*. In: S. Andima et al. (eds.), Papers on General Topology and its Applications. 7th Summer Conf. Wisconsin. Annals New York Acad. Sci. **704**, New York (1994), pp. 103-125.

[16] Focardi, R.," Gorrieri, R., *Classification of security properties (Part I: Information flow)*. In Foundations of Security Analysis and Design 2000, volume 2171 of Lecture Notes in Computer Science, pages 331-396. Springer, 2001.

[17] Gierz, G.," Hofmann, K. H.," Keimel, K.," Lawson, J. D.," Mislove, M.," Scott, D.S., *Continuous Lattices and Domains*, Encyclopedia of Mathematics and its Applications 93, Cambridge University Press, 2003.

[18] *Information Technology Security Evaluation Criteria (ITSEC)*. Version 1.2, Juni 1991.

[19] Järvinen, J., *Lattice theory for rough sets*, In: *Transactions on Rough Sets VI*. LNCS, vol. 4374, Springer, Heidelberg, 2007, pp. 400–498.

[20] Lamport, L., *Proving the correctness of multiprocess programs*. IEEE Transactions of Software Engineering, 3(2):125-143, Mar. 1977.

[21] Ligatti, J.," Bauer, L.," Walker, D., *Edit Automata: Enforcement mechanisms for run-time security policies*. International Journal of Information Security, 4(1-2):2-16, February 2005

[22] Ligatti, J. A., *Policy Enforcement via Program Monitoring*. PhD thesis, Princeton University, June 2006.

[23] Mazurkiewicz, A., *Trace Theory*, In W. Brauer et al., editors, Petri Nets, Applications and Relationship to other Models of Concurrency, number 255 in Lecture Notes in Computer Science, pages 279-324, Berlin-Heidelberg-New York, 1987, Springer

[24] Miné, A., *Weakly relational numerical abstract domains*, Ph. D. thesis, École Polythechnique, 2004.

[25] Naldurg, P.," Campbell, R.H.," Mickunas, M.D., *Developing Dynamic Security Policies*, Proceedings of the 2002 DARPA Active Networks Conference and Exposition (DANCE 2002), San Francisco, CA, USA, IEEE Computer Society Press, May 29-31, 2002

[26] Pagliani, P. and Chakraborty, M., *A Geometry of Approximation. Rough Set Theory: Logic, Algebra and Topology of Conceptual Patterns*, Trends in Logic, Vol. 27, Springer, 2008.

[27] Pawlak, Z., *Rough Sets*, International Journal of Information and Computer Science, Vol. 11(5) (1982), pp. 341–356.

[28] Pawlak, Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht, 1991.

[29] Read, S., *Thinking about Logic: An Introduction to the Philosophy of Logic*, Oxford University Press, Oxford, 1995.

[30] Schneider, F.B.," Morrisett, G.," Harper, R., *A Language-Based Approach to Security.* In: Informatics: 10 Years Back, 10 Years Ahead. Lecture Notes in Computer Science, Vol. 2000. Springer-Verlag, 2001. pp. 86-101.

[31] Smyth, M.B., *Powerdomains and predicate transformers: a topological view*, In J. Diaz, editor, Automata, Languages and Programming, pages 662-675, Berlin, 1983. Springer-Verlag. Lecture Notes in Computer Science Vol. 154.

[32] Smyth, M.B., *Topology*, In Abramsky, S.," Gabbay, D.M.," Maibaum, T.S.E. (eds.): Handbook of Logic in Computer Science. Volume I. Background: Mathematical Stuctures, pp. 641-761, Clarendon Press, Oxford, 1992.

[33] Szpilrajn, E., *Sur l'extension de l'ordre partiel*, Fund. Math. 16 (1930), pp. 386-389.