Design Time Reliability Predictions for Supporting Runtime Security Measuring and Adaptation

Antti Evesti, Eila Ovaska
VTT Technical Research Centre of Finland
Oulu, Finland
{antti.evesti, eila.ovaska}@vtt.fi

***Abstract*—The reliability of a quality-critical software component affects the security level that is achieved. There is currently no runtime security management approach that uses design time information. This paper presents an approach to exploiting design time reliability predictions in runtime security management. The Reliability and Availability Prediction (RAP) method is used to predict reliability at software design time. The predicted reliability values are stored in ontology form to support runtime use. The use case example illustrates the presented approach. The presented approach makes it possible to use design time reliability predictions at runtime for security measuring and adaptation. Hence, the reliability of security mechanisms is taken into account when security adaptation is triggered.**

Keywords - information security; quality; evaluation; metric; architecture

I. INTRODUCTION

A variety of quality prediction and testing techniques are used at software design time. The results of these predictions are used to enhance architecture designs, select better component alternatives, and reveal implementation errors. The use of these prediction results ends when satisfactory quality is achieved for a component or system and the product is delivered. However, these prediction results could also be used in runtime situations. This is reasonable, especially in reliability and security management. Reliability is an important factor in achieving a required security level, as can clearly be seen from the security decomposition presented in [1]. Weak reliability of a security-related software component ruins the offered security. Hence, the reliability information of component is valuable for security-related decision-making. This paper therefore presents an approach to bring the design-time reliability prediction results for runtime security measuring and adaptation purposes. To achieve this, ISMO (Information Security Measuring Ontology) [2] is extended in a way that allows prediction results to be stored at design time.

In the literature, different security adaptation approaches exist. The adaptive SSL presented in [3] sets parameters for the SSL session based on the environment information. An Extensible Security Adaptation Framework [4] adds a middleware layer for security mechanisms. The application sets the required security policy and, based on the policy, the middleware layer selects security mechanisms. Context-sensitive Adaptive Authentication [5] uses time and location information to calculate a confidence level for the authentication. In some situations, a low confidence level is sufficient while others require adaptation of the authentication method used. Our earlier work presents an approach that uses ontologies and risk-based measures for security adaptation [6]. These adaptation approaches are intended to work at runtime by observing the system's resources and environment. Based on the observations, different security mechanisms or parameters are set. To our knowledge, none of the existing approaches uses design-time information for adaptation purposes.

Figure 1. presents the broader context of the contribution of this paper. In the first phase, the Reliability and Availability Prediction (RAP) method [7] is used to predict future reliability from software designs. The prediction results are stored in ontology form in order to ensure exploitation at runtime. In this paper we will focus on this first phase. In the second phase, application security is measured at runtime. Reliability predictions are used as input information for security measuring. The third phase is security adaptation, which is triggered by the measuring phase. The adaptation also uses reliability predictions to select the most suitable security mechanism for different situations. After the adaptation, the execution returns to the measuring phase.

Figure 1. Broader scope

The contribution of this paper makes it possible to use design-time reliability predictions for runtime security measuring and adaptation. Hence, a wider information set is available for triggering and making a decision on the adaptation. In other words, information for runtime use can be collected in different phases of the application lifecycle. Thus, the adaptation is not only based on the measurements made just before adaptation but also on knowledge of the whole life cycle of the component.

The paper is organised as follows. After the introductory section, background information is presented. Next, Section 3 is divided into three parts describing the design steps towards applications with security adaptation, design time

 ISBN: 978-1-61208-174-8

reliability predictions, and a way to transform prediction results into the ontology form. Section 4 illustrates the presented approach by means of a case example. A conclusion and future work ideas close the paper.

II. BACKGROUND

Reussner et al. define reliability as the probability of failure-free operation of a software system for a specified period of time in a specified environment [8]. ISO/IEC defines security as follows: The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them [9].

The RAP method evaluates the reliability of a designed software system and its components already at architecture design time [7, 10]. The RAP method reveals design flaws and critical components from the reliability viewpoint. The evaluation is based on architectural models, which means that the first evaluation results are available before any implementation effort is required. Hence, modifications can be performed easily. The RAP method uses state-based models, i.e., Markov models, to predict the reliability of components. The path-based models are used to predict the reliability of a single execution path and the whole software system. The RAP method produces the following reliability values, known as probability of failure (pof) values: 1) independent pof values for software components, 2) pof values for execution paths, 3) the component's pof value in each execution path, 4) the components' system-dependent pof values, and 5) the pof value for the whole software system. The RAP method supports the feedback loop from software testing [11]. The prediction results can therefore be replaced with more accurate values when measured reliability values are available from the software testing. Tool support for the RAP method, called the RAP tool, is also available. The RAP tool reads architectural models from UML diagrams, i.e., state, component, and sequence diagrams. In addition, the RAP tool uses usage profiles that describe system usage, i.e., how many times each execution path is called. The usage profiles make it possible to perform own predictions for different user groups, e.g., professional and normal users. In this work, the results from the RAP tool will be made available for runtime use.

Evesti et al. present the ISMO ontology in [2]. The ISMO composes security ontology and general software measuring terminology. The ISMO thus offers a generic and extendable way to present security measures. These measures are connected to security threats and/or supporting mechanisms, depending on the measure. Measures are divided into base measures, derived measures, and analysis models. The base measure is the simplest measure and is used for more complex measures, i.e., derived measures and analysis models. The ISMO is instantiated as an example using authentication measures, especially Authentication Identity Structure (AIS) measures [1] for password-based authentication. The ISMO thus contains measures for password age and type, i.e., length and the number of different symbols. The software application uses different measures from those of the ISMO to measure its security level at runtime. In this work, the ISMO is extended to contain design time reliability predictions.

Savola et al. present Basic Measurable Components (BMCs) for security attributes (e.g., authentication, confidentiality, etc.) in [1]. BMCs are derived by means of the decomposition approach. The idea of BMCs is to divide security attributes into smaller pieces that can be measured. For example, authentication is divided into five BMCs in [1] as follows: Authentication Identity Uniqueness (AIU), Authentication Identity Structure (AIS), Authentication Identity Integrity (AII), Authentication Mechanism Reliability (AMR), and Authentication Mechanism Integrity (AMI).

III. RELIABILITY PREDICTIONS FOR SUPPORTING SECURITY MEASURING AND ADAPTATION

This section is dived into three subsections. Firstly, high-level design steps for the application with security adaptation features are described. Secondly, a design time reliability prediction is presented. Finally, a way to store the prediction results in ISMO in a way that supports runtime measuring is described.

A. Designing an Application with Security Adaptation Features

This subsection lists design steps that a software architect has to take when designing an application with security adaptation features. Figure 2. illustrates these design phases. The last three phases of the process are iterative. This is not depicted in the figure, however, for reasons of clarity.

1) Required security attributes

In the first phase, the software architect has a set of required security attributes for the application, for instance, S1 for communication confidentiality, S2 for user authentication, and S3 for data integrity requirements. S refers to a security requirement in general.

2) Adaptable security attributes

The software architect has to design adaptation features separately for each security attribute. From the above-listed required security attributes, the architect has to select which ones to implement in an adaptable manner, i.e., variation will take place at runtime [12]. In Figure 2. user authentication S2 is selected for the adaptable security attribute. Other security attributes are thought of as static security requirements from the runtime viewpoint. In other words, the possible variation in these attributes is taken into account at design time.

3) Mechanisms for adaptable security attributes

The adaptable security requirement has to be met by security mechanisms that can be changed or that have parameters that can be modified at runtime. For example, in the adaptable user authentication case, the architect designs two alternative user authentication mechanisms for the application, e.g., password-based and voice-based authentications. Another alternative is to design one security mechanism and set different parameters for it at runtime.

4) Measurements for triggering adaptation

 ISBN: 978-1-61208-174-8

Software measures are designed for the application in parallel with the mechanism design phase. In particular, this means base measures that require measuring probes inside the application. Adaptation at runtime will be triggered based on derived measures and analysis models, which both depend on base measures. In other words, base measures are used to compose derived measures and analysis models. Hence, the software architect has to implement these base measures for the application.

5) Architecture design

The architect designs the architecture for the system. From the security adaptation viewpoint, it is important that variation points are designed with care. For adaptable user authentication, this means that an authentication feature can be called without knowing the currently used authentication mechanism.

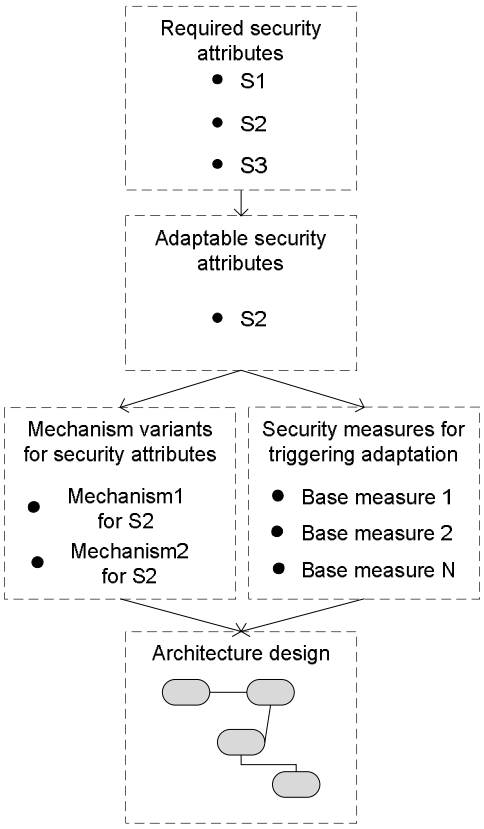

Figure 2. Steps towards adaptable application

B. Design Time Predictions

This subsection describes how the software architect predicts the reliability of components from the architectural designs. The architect uses the RAP method to perform these predictions. Based on the steps listed in the previous subsection, the architect has design documents for the application. Firstly, the component diagram describes the structure of the application. Secondly, the internal behaviour of components is described by means of state diagrams. Finally, sequence diagrams describe the mutual behaviour of components, i.e., how the component calls other components.

The RAP method contains state-based and path-based reliability prediction methods. For runtime security measuring and adaptation purposes, the RAP method is used to predict the probability of failure (pof) values for security mechanism components, i.e., mechanisms designed in phase 3 of the previous subsection.

The state-based prediction method calculates reliability for one independent software component by means of state diagrams. In state diagrams, pof values are given for each state to describe the failure probability in that particular state. Moreover, transition probabilities between states are described. Based on this information, the RAP tool automatically adds a separated failure state and calculates the component's pof value using a state transition matrix *p* and a probability vector *p(n)* as follows:

$$p = \begin{bmatrix} p_{SS} & p_{SA} & p_{SB} & p_{SF} \\ p_{AS} & p_{AA} & p_{AB} & p_{AF} \\ p_{BS} & p_{BA} & p_{BB} & p_{BF} \\ p_{FS} & p_{FA} & p_{FB} & p_{FF} \end{bmatrix} \quad (1)$$

$$p(n+1) = p(n)*p \quad (2)$$

In transition matrix p, p_{SA} presents the probability of transit from the start state S to state A. Similarly, p_{AF} presents a probability of transit from state A to the failure state F. In the beginning, the probability vector takes the form p(0) = [1, 0, 0, 0], which means that the probability of being in the start state is 1 at time moment 0.

The state-based prediction produces independent pof values for the components. These values are further used to calculate the component's pof values in different execution paths. Execution paths are presented by means of sequence diagrams in architectural models. The following equation is used to calculate a component's pof value in a particular execution path:

$$p_{ij} = 1 - (1 - p_i)^{Nij} \quad (3)$$

The previously calculated independent pof of the component is substituted in p_i, and Nij represents the number of execution times of the component in that execution path. Execution paths describe how the particular component is called in different execution paths.

As mentioned in Section 2, the RAP tool is also able to calculate pof values for each execution path, the component belonging to the particular software system, and for the whole software system. The equations for these calculations are presented in [11]. However, our interest is in bringing the previously presented component-related pof values for runtime use.

C. Storing Prediction Results in a Runtime-Applicable Way

After the RAP predictions, the software architect has the components' independent pof values and the components' pof values for the execution paths. Initially, the RAP tool was only intended for use at design time. Thus, the RAP tool

 ISBN: 978-1-61208-174-8

stores these reliability values in the component diagram by means of a UML profile. Hence, the values are available during the implementation and testing phases. Figure 3. shows the security mechanism part of the component diagram after the RAP predictions. Now, the mechanism alternatives designed earlier contain the predicted pof values. This is not practical for runtime purposes however. Reading the pof values from the UML profile requires a connection to a UML tool, which cannot be offered at runtime.

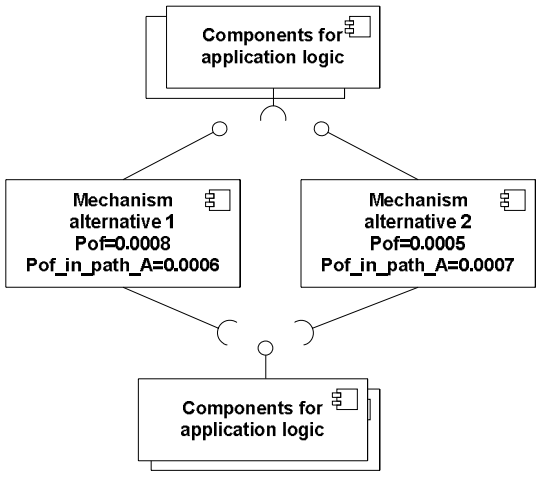

Figure 3. Component diagram after reliability predictions

As mentioned in Section 2, the ISMO supports runtime security measurements. The ISMO is therefore extended to store the components' reliability values. The following information needs to be stored in the ISMO:

1. Software component name
2. Software component version number
3. Which security mechanism the component implements
4. Reference to a place where the pof values are stored
5. Information on the execution path used to calculate path-specific pof values

The component name is intended to separate different alternatives of the mechanisms and is the name taken directly from the component diagram. It is natural to create an instance in the ISMO with a component name. This is because each software component is an individual element.

The version number separates different implementations of the same component. For instance, a new component version that contains bug fixes has a better pof value than the old version. This information therefore has to be separated in the ISMO. The version number is combined with the component name, i.e., an instance name in the ISMO. This naming convention also ensures that the ISMO does not contain instances with the same name.

Information on the security mechanism that the component implements is required because components use different security mechanisms to meet the required security, i.e., the mechanism alternatives in Figure 3. use different security mechanisms. For example, two components can use different authentication mechanisms to achieve user authentication. Countermeasures are described as concepts, i.e., classes, in the ISMO. Thus, it is reasonable to create the instance from the software component under the right countermeasure concept. Figure 4. presents instances created from software components from different countermeasure concepts.

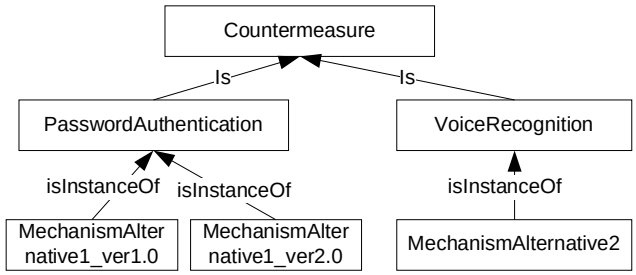

Figure 4. Component instances in the ISMO

The most important information is the components' pof values, and the above-described additions to the ISMO make it possible to put this information into the ISMO. Figure 5. shows a way of presenting pof values in the ISMO. A new base measure called *pof* is added to the ISMO. This is able to offer the component's pof values for the runtime measuring. In the ISMO, each measure is defined for *attribute*, i.e., path-specific pof and independent pof. The attribute relates to *MeasurableConcept*, i.e., Authentication Mechanism Reliability (AMR). Previously, the ISMO contained only measures related to the Authentication Identity Structure (AIS). Both attributes are connected to the countermeasure instance, i.e., MechanismAlternative_ver1.0 in this case, with the *hasMeasurableAttribute* property. Other mechanism instances also contain these attributes. However, for reasons of clarity, these are not presented in the figure.

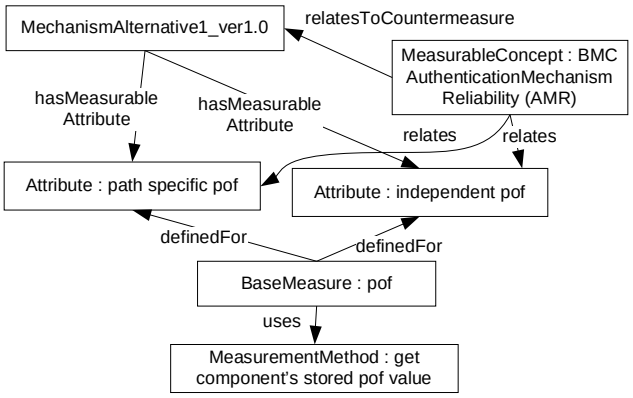

Figure 5. Update for the ISMO

Both attributes use the same pof base measure. The purpose of this base measure is to use *MeasurementMethod*, which retrieves the components' pof values. The measurement method is a concrete measuring probe that is able to retrieve pof values. Hence, it has to know the format

that is used to store pof values. The following structure is used: componentName, componentPof, the component's pof in execution path 1, the component's pof in execution path 2, etc. This structure therefore offers information on the execution path used to calculate path-specific pof values. It is possible to store pof values in a separated file or structure inside the application code. The separated file offers more flexibility, however, i.e., pof values can be updated without knowing the program code. The architect decides where the pof values are stored and creates an appropriate measurement method.

The reason why pof values are not stored directly in the attributes is twofold. Firstly, the measurement part of the ISMO – inherited from the Software Measurement Ontology (SMO) [13] – defines that attributes only define things that can be measured. Secondly, storing pof values outside the ISMO makes the ontology and pof values manageable. An application with security adaptation can contain several security mechanism components and each component can belong to several execution paths. Storing all these values into the ISMO will increase its size and complicate the updating of pof values.

IV. USE CASE EXAMPLE

This section gives a use case example of the presented approach. The purpose of the example is to show how the reliability of the security mechanism component is predicted. The results are stored in a runtime-applicable way in the ISMO.

The software architect designs a software application with security adaptation features. Communication confidentiality and user authentication are required securities for the application, c.f. Figure 2. From these security requirements, it is decided to implement user authentication in an adaptable manner. Hence, the architect designs alternative mechanisms for achieving user authentication, for example, password-based and fingerprint authentication. At the same time, base measures for measuring the user authentication are designed for the application. One of these base measures is pof. The value of the pof base measure is retrieved using a measurement method. It is notable, that the base measures and related measurement method implementations are reusable. Hence, the same base measure is also applicable to other security mechanisms.

After these design steps, there will be a component diagram, state diagrams of components, and sequence diagrams. Both authentication mechanisms are implemented as one independent software component called *passwordAuthentication* and *fingerprintAuthentication*.

Figure 6. presents a state diagram for the password authentication component. In this case, each transition probability is 1, i.e., only one leaving transition from each state. The architect sets the pof values for each state heuristically, and these pof values then affect the transition probabilities. In other words, the state's pof value reduces the occurrence probability of the right state transition respectively. Based on values from Figure 6. the RAP tool automatically adds the failure state and builds the transition matrix p as described in Section 3. From the transition matrix, the RAP tool calculates the pof value for the *passwordAuthentication* component. In this case, the pof value for the *passwordAuthentication* component is 0.000482. Similarly, pof values are given for states in the *fingerpringAuthentication* component, and the pof value of the component is calculated.

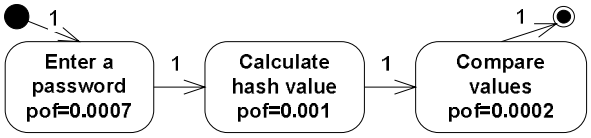

Figure 6. State diagram for the passwordAuthentication component

To exemplify path-specific pof values, the sequence diagram in Figure 7. is used. The RAP tool uses this sequence diagram, previously calculated pof value, and equation 3 to calculate the path-specific pof value. Hence, a pof value of 0.000482 is attained for the *passwordAuthentication* component in this specific execution path. In this case, the independent and path-specific pof values are the same because the *passwordAuthentication* component is only called once in this sequence diagram, c.f. equation 3.

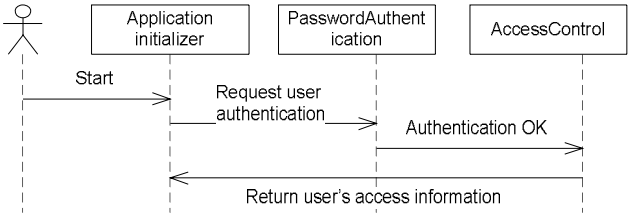

Figure 7. Sample execution path for password authentication

The architect stores this information in the ISMO in the form defined in the previous section and illustrated in Figure 8. In the figure, grey is used to describe information added in this case example. The component name is now *passwordAuthentication* and the version number is 1.0. Hence, the instance named *passwordAuthentication_ver1.0* is created under the password authentication concept in the ISMO. Similarly, the instance for the *fingerprintAuthentication* component is created. Both of these instances contain previously mentioned attributes. Attributes for the *fingerprintAuthentication* are not presented in the figure, however, for reasons of clarity. Calculated pof values are stored in the specific file called pofs. This file is presented in dark grey in Figure 8. because it is a separate part from the ISMO. *MeasurementMethod* contains a link to that file and is able to read pof values from the file. In this case, the file contains pof values for the *passwordAuthentication* and *fingerpringAuthentication* components.

 ISBN: 978-1-61208-174-8

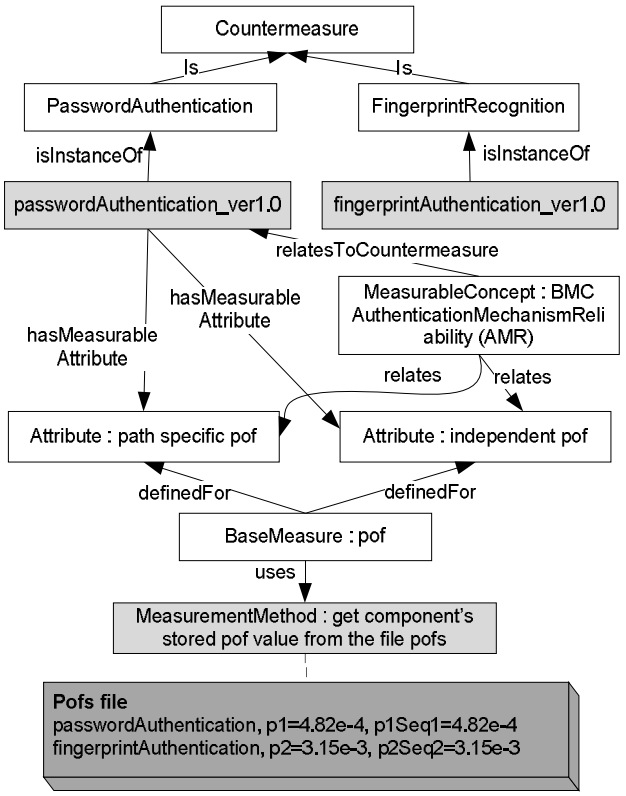

Figure 8. The content of ISMO after design time predictions

V. CONCLUSION AND FUTURE WORK

There is a clear connection between software reliability and security. An unreliable software component that performs security-related actions can ruin the security of the whole application. In this work, an approach was introduced to bring the results from design-time reliability predictions for runtime security measuring and adaptation purposes. Hence, the reliability of the security mechanisms can be taken into account when security adaptation is triggered. The work presented steps on how to produce an application with security adaptation features. Thereafter, reliability was predicted from design documents. Finally, these prediction results were stored in the ISMO, which makes it possible to use the prediction results at runtime. Storing the components' pof values in the ISMO required some extensions to the ontology. Firstly, the way to present individual security mechanism components in the ISMO was added. Secondly, the attributes for pof values were added and finally, a new base measure for pof values was introduced in the ISMO.

To our knowledge, there is no security measuring and adaptation approach that also uses design time information. Thus, the introduced approach is the first step towards enabling the use of the design time reliability predictions for runtime security measuring and adaptation. Reliability values are stored in a way that supports fast and easy updating. This is important when bug fixes for the security components are made. Furthermore, the real use of a component may produce different reliability to that initially predicted and it is then important to update the pof values. The presented approach is not restricted to one particular security mechanism or attribute. Hence, the software architect can make the decision of which attributes will be implemented in an adaptable manner on a case-by-case basis.

In the future, it is important to develop security measures that use the components' pof values in runtime security measuring. Current pof values of components can be used to compare different security components. Moreover, combining the reliability information and security level supported by the component offers valuable information for adaptation purposes. This means that the ISMO will be enhanced by new analysis models. The RAP tool also needs new features for storing information automatically to the ISMO.

ACKNOWLEDGMENT

This work is being carried out in the ARTEMIS SOFIA project funded by Tekes, VTT, and the European Commission.

REFERENCES

[1] R. Savola and H. Abie. "Development of measurable security for a distributed messaging system", *International Journal on Advances in Security, 2(4)*, pp. 358-380, 2010.

[2] A. Evesti, R. Savola, E. Ovaska, and J. Kuusijärvi, "The Design, Instantiation, and Usage of Information Security Measuring Ontology", *MOPAS'2011*, pp. 1-9, 17th April, 2011. 2011.

[3] C. J. Lamprecht and A. P. A. van Moorsel, "Runtime Security Adaptation Using Adaptive SSL", *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium*, pp. 305-312, 2008.

[4] A. Klenk, H. Niedermayer, M. Masekowsky, and G. Carle, "An architecture for autonomic security adaptation", *Ann Telecommun, 61(9-10)*, pp. 1066-1082. 2006.

[5] R. Hulsebosch, M. Bargh, G. Lenzini, P. Ebben, and S. Iacob. "Context sensitive adaptive authentication", *Smart Sensing and Context*, pp. 93-109, 2007.

[6] A. Evesti and E. Ovaska, "Ontology-Based Security Adaptation at Run-Time", *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 204-212, 2010.

[7] A. Immonen, "A method for predicting reliability and availability at the architecture level", in *Software Product Lines* T. Käkölä and J. Dueñas, Eds., 2006.

[8] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures", *J. Syst. Software, 66(3)*, pp. 241-252. 2003.

[9] ISO/IEC 9126-1:2001. *Software Engineering – Product Quality – Part 1: Quality Model.* 2001.

[10] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho, "Knowledge based quality-driven architecture design and evaluation", *Information and Software Technology, 52(6)*, pp. 577-601. 2010.

[11] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software", *J. Syst. Software, 84(6)*, pp. 1054-1070. 2011.

[12] E. Niemela, A. Evesti, and P. Savolainen, "Modeling quality attribute variability", *ENASE – Proc. Int. Conf. Eval. Novel Approaches Software Eng.*, pp. 169-176, 2008.

[13] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement", *Information and Software Technology, 48(8)*, pp. 631-644. 2006.

 ISBN: 978-1-61208-174-8