A Dynamic Risk-Based Access Control Approach: Model and Implementation

by

Sergey Savinov

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2017

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Dr. John Mylopoulos Professor
Supervisor(s)	Dr. Paulo Alencar Adjunct Professor Dr. Daniel Berry Professor
Internal Member	Dr. Don Cowan Distinguished Professor Emeritus
Internal-external Member	Dr. Ladan Tahvildari Associate Professor
Other Member(s)	Dr. Frank Tompa Distinguished Professor Emeritus

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Access control (AC) refers to mechanisms and policies that restrict access to resources, thus regulating access to physical or virtual resources of an information system. AC approaches are used to represent these mechanisms and policies by which users are granted access and specific access privileges to the resources or information of the system for which AC is provided. Traditional AC approaches encompass a variety of widely used approaches, including attribute-based access control (ABAC), mandatory access control (MAC), discretionary access control (DAC) and role-based access control (RBAC). Emerging AC approaches include risk adaptive access control (RAdAC), an approach that suggests that AC can adapt depending on specific situations.

However, traditional and emerging AC approaches rely on static pre-defined risk mitigation tasks and do not support the adaptation of an AC risk mitigation process (RMP). There are no provided mechanisms and automated support that allow AC approaches to construct RMPs and to adapt to provide more flexible, custom-tailored responses to specific situations in order to minimize risks. Further, although existing AC approaches can operate in several knowledge domains at once, they do not explicitly take into account the relationships among risks related to different dimensions, e.g., security, productivity. In addition, although in the real world, risks accumulate over time, existing AC approaches do not appropriately provide means for risk resolution in situations in which risks accumulate as different, dangerous tasks impact risk measures.

This thesis presents the definition, the implementation, and the application through two case studies of a novel AC risk-mitigation approach that combines dynamic RMP construction and risk assessment extended to include forecasting based on multiple risk-related utilities and events; provides support for a dynamic risk assessment that depends on one or multiple risk dimensions (e.g., security and productivity); offers cumulative risk assessment in which each action of interest can impact the risk-related utilities in a dynamic way; and presents an implementation of an adaptive simulation method based on risk-related utilities and events.

Acknowledgements

I would like to express my appreciation to my advisor Paulo Alencar. It has been an honour to be his Ph.D. student. While he was supervising my research I felt that I had enough liberty to be fully creative while trying new approaches for complicated challenges. Moreover, I felt that his experience in dealing with issues and decisions that novel research always faces allowed me to complete this thesis.

I also would like to thank my co-advisor Daniel Berry for his work as part of my thesis committee and for providing insightful comments and occasional jokes that always lifted my spirit.

I am also grateful to Donald Cowan and to Frank Tompa for their work as part of my thesis committee, specifically for their invaluable feedback on early stages of this thesis that helped to shape both the problem and the solution.

I would like to thank my parents Juri and Tatiana and my brother Pavel for their help and encouragement. It is because of my father's guidance I chose to pursue the Ph.D., and it is my mother who is and always was a cornerstone of our family. I was fortunate to be raised in a family that strives for knowledge and always offers help and support.

Finally I am thankful to my wife Daria, who has been with me through the most difficult parts of this journey for her faithful support, love and patience. Thank you.

Dedication

This thesis is dedicated to Daria, Juri, and Tatiana.

Table of Contents

List of Figures xii

List of Tables xiv

List of Abbreviations xv

1 Introduction 1

1.1 Motivation 1

1.2 Motivating Example 4

1.3 Problem Statement 9

1.4 Proposed Approach 10

1.5 Contribution 13

1.6 Application of the Approach 14

1.7 Outline of the Thesis 15

2 Related Work 16

2.1 Access Control Approaches 17

2.1.1 Role-Based Access Control (RBAC) 18

2.1.2 Mandatory Access Control (MAC) 19

2.1.3 Attribute-Based Access Control (ABAC) 20

2.1.4 Discretionary Access Control (DAC) 20

2.1.5 eXtensible Access Control Markup Language (XACML) 20
2.1.6 Obligations in Access Control 22
2.2 Risk Adaptive Access Control (RAdAC) Based on Risk-Benefit Analysis 22
2.3 Risk Management Approaches 24
2.4 Business Process Modelling and Simulation 24
3 Description of the Approach 26
3.1 Approach Architecture 26
3.1.1 Risk Quantification and Analysis Model 30
3.1.2 Risk Mitigation Process Construction 34
3.1.3 Forecasting Core Model 38
3.1.4 Domain Models 53
3.1.5 Reserves 53
3.1.6 Strategies 54
3.2 Details about the Application of the Approach 57
3.2.1 Example of an Application of the Approach 59
3.2.2 Discussion 65
3.3 Formal Definitions 65
4 Case Studies 79
4.1 Patient-Privacy Case Study 80
4.2 Software Development Case Study 87
4.2.1 Overview 87
4.2.2 Simulation Results 96
4.2.3 Discussion of the Results 106
4.2.4 Domain Models in the Case Study 106
4.3 Implementation 109

5 Summary and Future Work 110
5.1 Summary 110
5.2 Limitations 113
5.3 Future work 113
5.3.1 Additional Case Studies 113
5.3.2 Explicit Risk Trading Measures 114
5.3.3 Context-Aware Models 114
5.3.4 Introducing Software Agents and Anomaly Discovery Methods 114
6 Bibliography 115
7 Appendices 124
8 Appendix A - Source Code of the Classes Implementing the Approach 124
A.1 Main.java 124
A.2 ActionsQueue.java 129
A.3 Random.java 134
A.4 AccessControlDecisionMaker.java 134
A.5 FirewallDecisionMaker 134
A.6 FirewallPolicy.java 136
A.7 ServerLocalAccessControl.java 136
A.8 Workstation.java 138
A.9 CompUnit.java 138
A.10 CPU.java 139
A.11 HardDriver.java 140
A.12 HardwareEntity.java 141
A.13 NetworkingInterface.java 141
A.14 RAM.java 142

A.15 DataSource.java 143
A.16 DataSourceGetResult.java 151
A.17 DataSourceGetResults.java 152
A.18 FileWithRecords.java 153
A.19 Message.java 153
A.20 RecordsCollection.java 157
A.21 DoubleRecord.java 159
A.22 EmailRecord.java 159
A.23 EmailRecordClass.java 161
A.24 IntegerRecord.java 161
A.25 Record.java 162
A.26 StringRecord.java 164
A.27 UserRecord.java 164
A.28 UserRecordClass.java 165
A.29 Location.java 165
A.30 Customer.java 165
A.31 DeveloperRole.java 166
A.32 Role.java 180
A.33 Browser.java 181
A.34 Component.java 182
A.35 DatabaseServer.java 183
A.36 Firewall.java 185
A.37 FTPServer.java 185
A.38 Listener.java 187
A.39 MailServer.java 188

A.40 OperatingSystem.java 189
A.41 RemoteControlServer.java 194
A.42 Server.java 196
A.43 Session.java 198
A.44 SourceCodeWebServer.java 199
A.45 WebServer.java 201
A.46 Entity.java 203
A.47 FileNameCleaner.java 203
A.48 Lib.java 203
A.49 NamedEnvironment.java 204
A.50 Pair.java 205
A.51 Locator.java 205
A.52 Risk.java 207
A.53 RiskCollection.java 208
A.54 Focus.java 210
A.55 ProcessTree.java 212
A.56 Task.java 212
A.57 TaskResult.java 214
A.58 TaskResults.java 215
A.59 RMS 217
9 Appendix B - UML Diagram of the Implementation Classes 221
10 Appendix C - Documentation Describing Class Attributes and Methods 222
11 Appendix D - Basic Design Model and Risk Assessment 252
12 Appendix E - Process of the Patient-Privacy Case Study 263

List of Figures

Figure 1. Possible ATM process adaptations 7
Figure 2. XACML architecture 12
Figure 3. Proposed approach architecture 12
Figure 4. Overview of the RBAC model 19
Figure 5. XACML data flow [66] 21
Figure 6. XACML architecture 27
Figure 7. Proposed approach architecture 27
Figure 8. Examples of the original process and the constructed set of RMPs for the software development setting 36
Figure 9. Examples of the original process and the constructed set of RMPs for the patient-privacy application 37
Figure 10. UML class diagram, forecasting core model 42
Figure 11. Focus and event lifetimes 45
Figure 12. Task logic 46
Figure 13. Task switching 47
Figure 14. Events queue 50
Figure 15. Behaviour of the forecasting model 52
Figure 16. Behaviour of the improved design model 56
Figure 17. High level approach overview 58
Figure 18. Deriving the utility from risks and benefits 60
Figure 19. Forecasting through simulation 60
Figure 20. Risk mitigation tasks 62
Figure 21. Possible RMPs and their results 64
Figure 22.Object A: possible states and preceding events 73

Figure 23. Patient-privacy simplified process 81
Figure 24. Utility diagram of the regular emergency scenario 83
Figure 25. Utility diagram of the falsified credentials scenario 85
Figure 26. Utility diagram of denied access during the emergency scenario 86
Figure 27. Software developer process graph 88
Figure 28. Network diagram of case study initial data 89
Figure 29. Security risk value over time 98
Figure 30. Productivity risk value over time 99
Figure 31. Total utility value over time 100
Figure 32. Total utility for another RMP 103
Figure 33. Total utility value over time for method body B, documentation C, method header C, method header D 105
Figure 34. UML class diagram for the domain model in the software development case study 107
Figure 35. UML class diagram for the approach and the domain model in the software development case study 108
Figure 36. Project development diagram 253
Figure 37. UML class diagram of risk assessment data including risk of delay extension 259
Figure 38. Process of the Patient-Privacy Case Study 263

List of Tables

Table 1. Forecasting model data 59
Table 2. Domains, risks and dimensions 87
Table 3. Documentation and code attributes 92
Table 4. Simulation results for documentation B, method body B, documentation C, and method body C 97
Table 5. Simulation results for method header B, documentation C, email from project manager 101
Table 6. Simulation results for method header B, documentation C, email from project manager 102
Table 7. Simulation results for method body B, documentation C, method header C, method header D 104
Table 8. Initial scenario data 254
Table 9. Connection between reputation and funds 255
Table 10. General risk analysis data 255
Table 11. Risk of having the code stolen data 256
Table 12. Risk of having security compromised data 257
Table 13. Benefit of completing project without delay data 258

List of Abbreviations

ABAC - Attribute-Based Access Control

AC - Access Control

ATM - Automated Teller Machine

DAC - Discretionary Access Control

FM - Forecasting Model

MAC - Mandatory Access Control

PIN - Personal Identification Number

QRM - Quantitative Risk Related Measure

RMP - Risk Mitigation Process

RAdAC - Risk Adaptive Access Control

RBAC - Role-Based Access Control

UML - Unified Modeling Language

XACML - eXtensible Access Control Markup Language

Chapter 1

Introduction

1.1 Motivation

Risks are pervasive in modern software systems. These risks can be classified in many categories, which include risks associated with unauthorized or illegal data access, dissemination through intentional or accidental disclosure, modification, erasure, or copying. There are also risks involving productivity and financial organizational aspects. They can negatively impact all segments of society, including private companies, governments, and the general public, which often have sensitive and proprietary information that needs to stay confidential, have proprietary value, or resources that need to be protected. Traditionally, risks of unauthorized disclosure, modification, erasure or copying data are related to the security domain and can be mitigated by techniques such as an access control (AC).

However, risks that are not directly related to the security domain, such as those related to productivity, public relations, sales or financial aspects, are usually not within the scope of an AC approach. Moreover, any of these risks can be interrelated with security procedures, causing each risk mitigation decision to influence other aspects of the system functionality. For example, adopting a complex procedure to verify security-related

information can discourage a client from using the provided service, which directly affects the performance of a sales department.

While any traditional AC approach generally results in a permit or deny outcome, an emerging approach, which is called risk adaptive access control (RAdAC), provides a paradigm for AC that considers a set of factors such as productivity and financial aspects to complement an analysis of security risks and benefits as part of an AC decision. RAdAC also assumes that any situational condition can influence the relative weight of these factors in determining the AC outcome. However, both traditional AC and RAdAC methods are unable to support the construction of RMP dynamically based on utility measures associated with any existing risk factor.

In addition, these methods can provide a permit or deny outcome, but they fall short in offering a permit or deny outcome combined with a set of extra tasks that need to be executed to mitigate any existing risk. For example, the solution to an AC request could permit the addition of a new user after the head of the new user's department approves him or her or could permit the addition of the new user with limited privileges. Each of these alternatives would have a different risk level associated with it. This work aims at showing that in practice, there is a combined outcome in any situation that is less risky and more appropriate in terms of risk utilities than the traditional permit or deny outcome.

Related work on RAdAC either provides a conceptual framework [1] or focuses on only a domain-specific solution [2] that cannot be reused in other knowledge domains, and supports only restricted outcomes (i.e., permit or deny). In contrast, this thesis presents a novel approach to AC risk assessment that focuses on dynamic risk mitigation process (RMP) construction based on risk-related utilities and events and extends risk assessment to forecasting. This thesis also provides practical case studies related to patient privacy and software development in which a pure permit or deny outcome can be dynamically combined with a set of additional tasks in order to mitigate any relevant risk. Using case studies, this thesis shows that the permit or deny outcome combined with the set of additional tasks can, in specific cases, be more useful than the pure permit or deny outcome.

This research is motivated also by limitations of the current approaches such as de-facto industry standards including role-based access control (RBAC) and eXtensible Access

Control Markup Language (XACML) and emerging approaches, including RAdAC. The limitations of existing AC approaches due to the lack of AC risk accumulation over time have a significant role in critical problems in various areas within existing AC systems, including breaches in information security caused by a number of information leaks such as those involving Edward Snowden, Bradley (now Chelsea) Manning, and Julian Assange [3] and private information leaks [4], as well as an array of accidents related to proprietary data theft [5].

Moreover, existing approaches face certain challenges that are not appropriately addressed by existing approaches. Although most existing AC approaches rely on manual processes, the desire for an increased level of automation is often expressed in the literature [6, 7, 8, 9, 10]. However, existing AC approaches are unable to provide a roadmap to improve automation significantly. This lack of automation leads to a heavy involvement of human analysts, which is costly, error-prone, and vulnerable to an attack based on social engineering.

As well, current approaches have problems with handling each risk in real-time, specifically when handling a previously unidentified threat [11, 12, 13]. Many of existing approaches base their decisions on a set of policies that are composed by a security analyst [14, 15], who cannot resolve AC situations in real time, but can deal only with problems that were identified previously. On the other hand, critical situations involving security often happen after the risk assessment and the analysis of potential vulnerabilities are performed [16, 17]. Existing static risk assessment and AC methods do not provide an appropriate solution that can support the resolution of a complex dynamic AC situation.

Existing approaches lack feedback [18] and possible options for resolving AC situations when a legitimate user or component is unable to continue its activity due to the necessity of access to a requested resource or service, when AC refuses this access for some reason. One of the common messages regarding denied access attempt states, that ‘access was denied’ without providing any other details. Such a message causes the user to ask a system administrator to make an exception for his or her activity, which interrupts ongoing business processes and increases load on system administrators [19, 20]. In addition, the traditional approach implies that there has to be an often-used system administrator role with wide access to services and data and unrestricted access with respect to time. However,

compromised access to an account with the system administrator role leads to the system being completely exposed to malicious actions, and if the account is widely used, it is not possible to limit the risk of such an account being compromised.

1.2 Motivating Example

Denial to a legitimate AC request is not the only problem that existing and emerging approaches need to resolve. As a motivating example consider the following scenario involving a customer's performing a daily banking transaction via an automated teller machine (ATM). The goal of the customer is to complete the transaction as quickly as possible, entering a minimal amount of information. The goal of the ATM software is to provide services to the bank client, helping each customer to achieve his or her goals while mitigating the risks related to unauthorized access. The customer follows a process specified by the ATM, which can be modified to attain a proper balance between reducing the time and effort spent by the customer and maintaining a sufficient level of security.

Assume that a customer arrives at the ATM without his or her card. Normally, the ATM software would require the customer to insert a card into the machine and enter a PIN. In contrast, the proposed approach can modify the regular ATM process so that the customer would complete the transaction without the card. To achieve this, the RMP would use devices already available in existing ATM machines to: (i) perform a facial recognition procedure for authentication purposes; (ii) ask the customer to answer some security questions the bank software has on file; (iii) require the customer to provide his or her signature that could be scanned by bill acceptor; and (iv) send a security code to the customer's cell phone for verification. Any or all of these and similar procedures could be in the RMP so that the customer, who was not able in the first place to finish the transaction, could perform low-risk tasks such as paying bills to payees or transfer funds between accounts.

Lack of Support for Process Adaptation

Assume that the process of using the ATM requires the user to enter the amount related to his or her last credit card operation, but a server providing this data to the ATM is not responding for some reason. As a result, the customer would not be able to proceed with his or her transaction. In this case, the process being followed was pre-defined and cannot change in order to remediate the situation. There is a lack of support for adapting this process so that the customer would be able to complete the ongoing transaction. A possible RMP would be to change the authentication procedure to use data that is available to an ATM. In this case, the original pre-defined authentication procedure could not be completed since the access to the server was not possible. However, if the authentication process had support for adaptation, the ongoing process could be adapted to user another available server, for example, by making the ATM ask the user for his or her online banking password to verify the user's identity. It would be nice not to be restricted by the pre-defined process and to be able to adapt the process under specific circumstances. In this way both the user and the AC system would be able to meet their goals: the user would be able to complete the transaction and the AC system would be able to prevent an unauthorized access.

Lack of Support for Adapting Risk Assessment

Assume that there is a customer transaction that is originating from Mexico, and the last known customer location is Canada. Historically, discrepancies between locations is a strong indication of an unauthorized access attempt. Therefore, this situation presents a significant security risk. Existing solutions can either accept the risk of unauthorized access or deny operation. Accepting risk of unauthorized access by allowing operation to proceed, can lead to fraud. Declare the transaction a fraud and deny operation can result in a dissatisfied customer. While each option has a risk associated with it, a pre-defined process with a static logic, which existing AC systems rely on, has no way of adapting to the situation. However, it is possible to adapt the process. Instead of allowing or denying the transaction, the bank can implement additional security checks, e.g., to send message (SMS, email or call) to the customer with a verification code and ask the customer to enter it at the ATM. This process adaptation, RMP, allows the ATM to authenticate the customer, reducing the risk of unauthorized access, and allows the customer to proceed with his or her goals. While it is

possible to incorporate all significant risk combinations into a logic of a business process, it overcomplicates logic and makes process both non-atomic and hard to revert.

However, existing processes can benefit from adaptation even without significant risks present. For example, given that a customer wishes only to check a current balance, assuming that the balance is not considered highly sensitive information and a card is present in ATM, it is possible to provide this information without asking the customer to authenticate by entering a PIN. This process adaptation, RMP, would reduce the time and effort spent by the customer to perform the transaction. Given that there are no significant security risks present, the adaptation of omitting the authentication would increase the utility of the RMP. However, if subsequently the customer attempts to perform a risk-sensitive operation, the ATM would adapt the process by asking the customer to login before the operation can be performed. On the other hand, if the balance is considered a highly sensitive information, for example in case of a criminal stealing credit card information in batches, during evaluation of these cards, it is possible not only to block cards in question, but also to report wrong information about the balance which would alert legitimate cardholders and prevent criminal from obtaining required information. A set of possible ATM process adaptations for a number of different situations involving the ATM and a specific customer is presented in Figure 1.

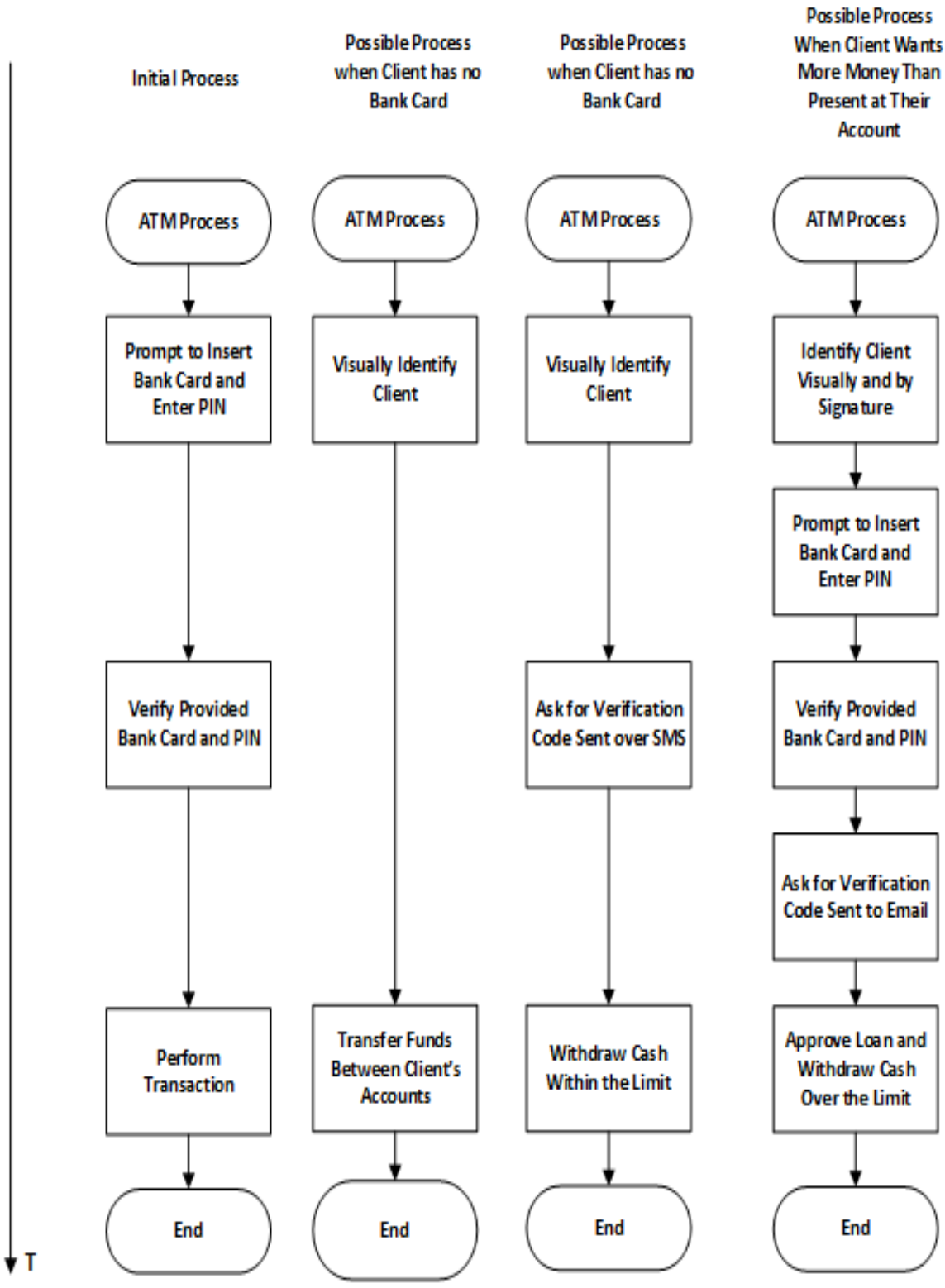


Figure 1. Possible ATM process adaptations

Absence of Support for Accumulating Risks over Time.

In situations with no significant risks present, AC typically just accepts insignificant risks without accumulating them. The analysis and design often includes limits set for security procedures to be activated. For example, as a part of the ATM requirements, there is usually a daily limit of the amount of cash a customer can receive from his or her card. This feature was designed to prevent a significant damage caused by a breach of security, e.g., by a malicious person making a copy of an existing bank card and using it to withdraw cash from ATM. However, after a day ends, the amount the customer can withdraw is reset and thus the risks are not accumulated, but disregarded.

Another related example is a feature of a banking system involving not requiring the PIN while buying items with lower than a predefined threshold total value. While this feature reduces the time required by a customer to complete a transaction, it also reduces security levels by making it easier for a malicious attacker to abuse the security protocol. A typical example of this abuse is a check performed by a buyer of credit card numbers often sold in batches. To check a batch, the buyer would typically attempt to buy something cheap with one of the cards randomly selected from the batch. An insignificant transaction typically is unlikely to be classified as a fraud. Therefore, if the selected card is valid, the transaction would be allowed and therefore, the buyer can verify validity of these credit cards. Accumulating all encountered risks over time allows risk analysts and risk analysis algorithms to improve the identification of anomalies, risk quantification, and correlations. The analysis of complete risk chains can result in discovery of previously unknown cause and effect data.

1.3 Problem Statement

Current AC approaches rely on static risk mitigation tasks. However, using a predefined, static AC reaction to certain risks can lead to a potential vulnerability. For example, assume that a malicious user has gained an unauthorized access to confidential information. An AC could attempt to notify a security analyst by sending email, following a static predefined AC policy. The malicious user can divert or intercept the email, e.g., by changing the configuration of a mail server or using a man-in-the-middle attack on the routing.

Each of RAdAC and RBAC is a predefined approach that can be exploited. A decision made by an AC can be anticipated to benefit some malicious entity (e.g., by obtaining the credentials of a user with a high reputation, an attacker can make security skip part of an analysis, simplifying an attack problem). Both AC approaches do not support forecasting as time progresses and new risk-related events, tasks, and situations occur.

Almost every AC approach operates in several knowledge domains at once, but it does not explicitly take into account the relationship among risks related to different dimensions. For example, when an AC system involves security and sales, it needs to take into account not only the set of requirements related to any risk of each dimension, but also the set of requirements related to how risks in one dimension affect risks in another dimension. In this case, if an AC system requires a customer to complete multiple forms to improve security prior to placing an order, the performance level of the sales department personnel decreases.

The domain models for the case studies are presented to illustrate the applicability of a proposed approach. These models describe various domain, including security, business processes and computer-based systems. The set of the considered risks includes the risk of code being stolen and the risk of project delays. The corresponding risk dimensions include security, measured in security units, and productivity, measured in productivity units.

Existing AC approaches are not designed to accumulate risks within ongoing dynamic processes, as opposing tasks. For example, in the case that an analyst tries to access a file that he or she is supposed to have access to, associated access risks are acceptable, because, of course, the analyst is allowed to access to this information. However, the risks of the analyst's accessing many different files accumulate in the real world. However, current

AC approaches do not support such an accumulation. The accumulated risks present a significant problem, e.g., when the analyst tries to copy an entire database file by file. That risk cannot be handled by considering only each access to a file. Instead, there is a need to define a RMP that combines multiple access attempts to different files in a dynamic way. Although in this example, a RMP involves only a single action performed a number of times, in general, a RMP can combine different tasks that, when performed, can lead to an undesirable risk level.

Current AC approaches rely on human analysts to investigate AC problems. The resulting analysis is predefined. Therefore, the existing AC approaches cannot take into account a set of risk consequences due to an array of real-time user tasks. A predefined human analysis also often assumes that the privilege level that a user has is independent of a possibly adverse real-time action that the user may try to perform. However, these tasks need to be taken into account and should lead to dynamic decrease in the privilege level of the user. The adaptation of AC policies to a number of tasks with acceptable risk is not supported in existing AC approaches.

Therefore, current AC approaches are based on predefined static risk mitigation methods, do not provide explicit risk dimensions differentiation, are unable to support accumulation of various risks adequately and make human intervention in risk analysis mandatory.

1.4 Proposed Approach

This thesis expands on the XACML architecture for AC as shown in Figure 2. Here a business process such as modifying a personnel record or withdrawing funds or paying a bill at an ATM uses a software component to access a resource such as a personnel record or one or more bank account records. Before accessing the record(s) the component must be given permission by a policy enforcement point. The policy enforcement point typically asks for some identifying credentials from the component, which are provided by the individual or systems implementing the business process. Two examples of credentials could be a userid and password, or a debit card and a personal identification number (PIN).

Once the credentials are supplied to the policy enforcement point they are sent to a policy decision point, which checks the credentials against a static AC policy. This policy could check against its list of valid credentials and if the user has supplied the correct information then access is approved, otherwise it is denied.

As explained under the ATM example earlier in this chapter in Section 1.2, one would like to have more flexible access policies, which would depend on the circumstances surrounding the access request and the risk associated with the specific access conditions. This means changing the policy decision point and static policy to a dynamic one and so they are shown in bold in Figure 2.

Figure 3 provides the details that are implemented in this dynamic AC approach, that is, in the boxes shown in bold. Once the policy decision point determines the circumstances surrounding the request and constructs possible alternative sequences of tasks to mitigate the risks, then it simulates the alternatives to assess the risks.

For example in the ATM situation, if users do not have their debit cards, what is the risk associated with sending an access code to their phone and allowing users to acquire cash from the ATM. If the user just wants to pay a bill, then the risk associated with the transaction is much smaller. Based on the simulation, the policy decision point might deny access to obtaining cash but allow payment of a bill.

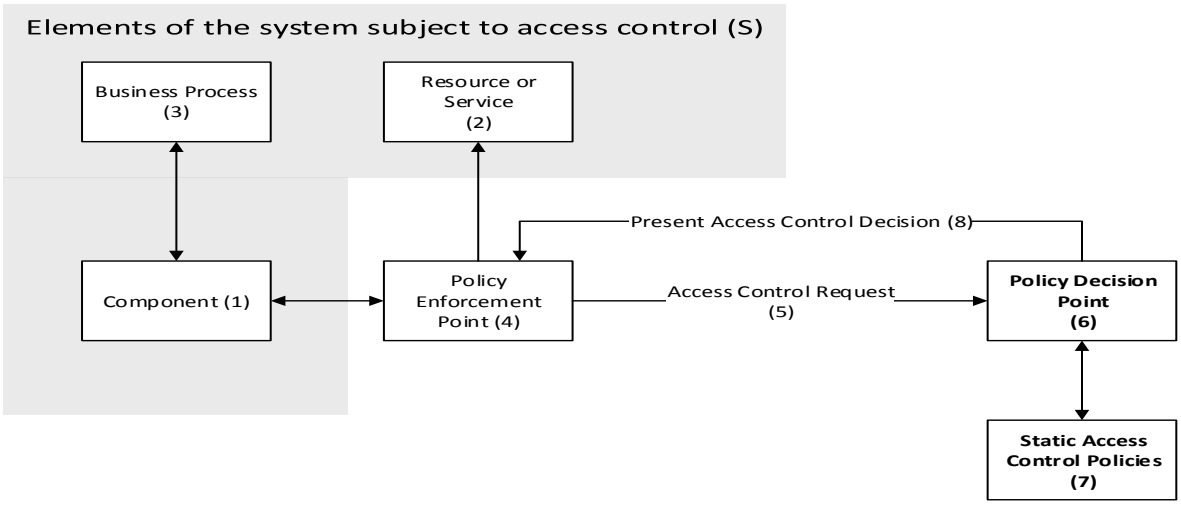


Figure 2. XACML architecture

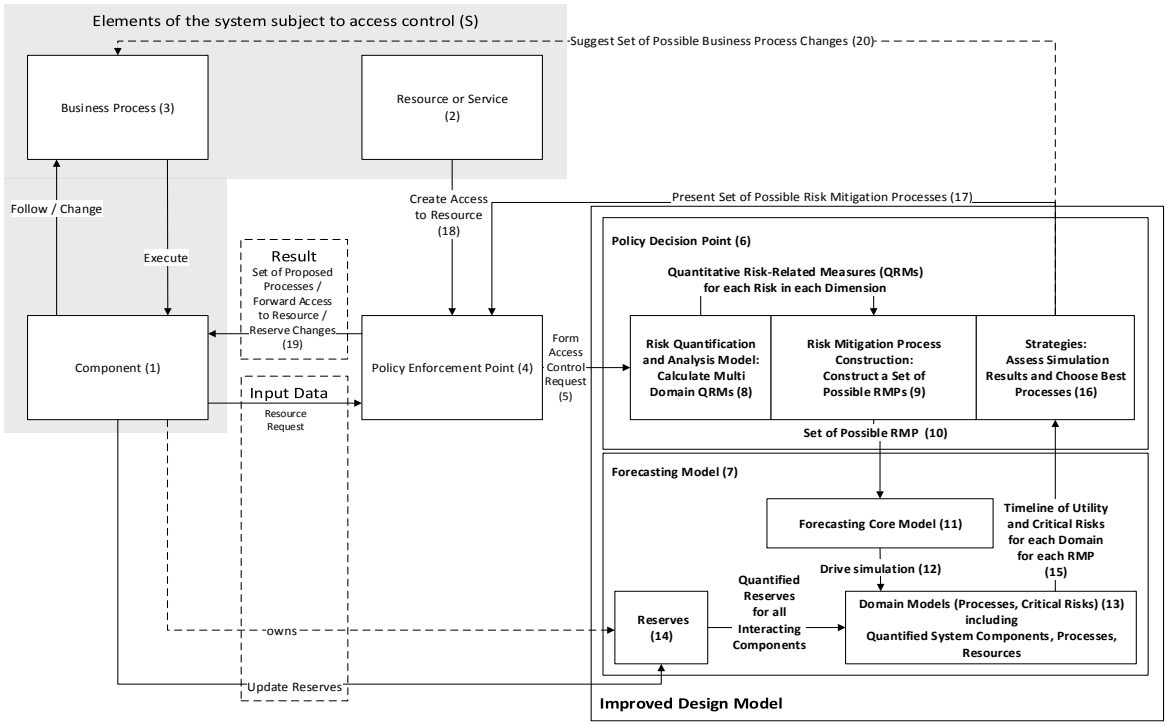


Figure 3. Proposed approach architecture

Similarly in a health application, what if the medical staff does not have access to a patient's medical records and must wait for the person or organization that does? If the patient is being treated for a broken arm, then access can probably wait, but if the condition is life-threatening then the policy decision point should decide to allow access as the risk of not doing so is too high.

Thus the policy decision point uses the simulation to compute the risks based on the circumstances surrounding the business process, and construct and select a sequence of risk mitigation tasks, which we call RMPs, that can minimize the risks. These sequences of tasks can be seen as alternatives that can be used to have access to a resource.

These examples summarize the ideas behind the approach. Chapter 3 provides the details of how the policy decision point and a forecasting model (FM) used for simulation are designed.

1.5 Contribution

This work provides the definition of a novel approach to risk assessment and in particular to AC risk assessment that focuses on a dynamic RMP construction based on risk-related utilities and events. This new approach extends risk assessment to include forecasting, which has resulted in the development of a new risk mitigation FM, an automated support for the risk mitigation approach and its implementation, and two case studies involving risk-oriented dynamic applications.

The thesis statement related to this research work follows:

By introducing a new approach to AC risk assessment that focuses on a dynamic RMP construction based on risk-related utilities and events, a new design model and its implementation can be provided that supports the evaluation of risks and benefits at runtime, as opposed to the current static and predefined processes. Instead of giving a yes/no answer, the new design model provides alternative sequences of tasks that components (or users) can follow to obtain access to specific resources.

The proposed risk-mitigation approach has the following features:

1. the combination of dynamic RMP construction and risk assessment is extended to include forecasting based on multiple risk-related utilities and events;
2. the risk assessment scope varies depending on the dynamic association of the components with one or multiple risk dimensions (e.g., security and productivity);
3. the approach provides cumulative risk assessment in which, each action of interest impacts the risk-related utilities associated with components in a dynamic way;
4. the adaptive simulation method is based on risk-related utilities and events.

The contributions of this thesis include:

- the introduction of a novel approach to risk assessment and remediation;
- the definition of a new risk mitigation FM;
- the provision of automated support for the risk assessment and mitigation approach;
- the applicability of the approach is illustrated through modelling and implementing case studies involving dynamic risks which provide risk assessment and set of custom-tailored RMP.

1.6 Application of the Approach

The proposed risk management approach offers a new method of AC adaptive risk assessment. The applicability of the proposed approach is illustrated by using two case studies: a first case study about the resolution of a medical emergency in a patient-privacy setting and a second case study about software development.

Specifically, the application of the approach involves: (i) a manual quantitative risk assessment evaluation in the context of the software development and patient-privacy scenarios; (ii) the implementation of the core framework for supporting the automation of the AC risk assessment and the construction of a set of RMPs; and (iii) an automated quantitative risk assessment evaluation in the context of a scenario involving software development in a company.

1.7 Outline of the Thesis

Chapter 2 illustrates the related areas and explains the background necessary to understand the details of the proposed approach.

Chapter 3 describes the details of the proposed approach, including architecture and the interaction for the external entities.

Chapter 4 illustrates the applicability of the approach using two case studies and a comparison with RBAC and RAdAC.

Chapter 5 states the conclusions of the thesis and identifies future work.

Appendix A lists the complete source code of the implementation. Appendix B presents a Unified Modeling Language (UML) diagram of the implementation classes. Appendix C contains a list of the implementation classes attributes, methods and their descriptions. Appendix D contains details about the risk calculation process. Appendix E provides a detailed process diagram used in the patient-privacy case study described in Chapter 4.

Chapter 2

Related Work

The proposed approach is related to several areas of the research, including:

- AC approaches and policies;
- RAdAC-based risk-benefit analysis;
- XACML;
- Obligations in AC;
- Risk management approaches; and
- Business process modelling and simulation.

In comparison with related work, this thesis introduces a new approach to AC risk assessment that focuses on a dynamic RMP construction based on risk-related utilities and events, and a new design model and its implementation is provided that supports the evaluation of risks and benefits at runtime, as opposed to the current static and predefined processes. Instead of giving a yes/no answer, the new design model provides alternative sequences of tasks that components (or users) can follow to obtain access to specific resources.

2.1 Access Control Approaches

AC refers to mechanisms and policies that restrict access to resources, thus regulating access to a system for which AC is provided or to a collection of physical or virtual resources [21]. An AC approach is used to represent these mechanisms and policies wherein users are granted access and specific privilege to a system for which AC is provided, its resources or information [22]. There exists a variety of widely used AC approaches that include attribute-based access control (ABAC) [23, 24], mandatory access control (MAC) [25], discretionary access control (DAC) [26] and role-based access control (RBAC) [27, 28, 29, 30].

Traditional AC approaches were designed to be reactive, not proactive[22]. Some of these approaches are based on a client-server architecture, in which an interaction has a reactive nature because it is started by each client's explicitly invoking a server call. In contrast, a proactive interaction is dynamic and relies on implicit invocation. Further, traditional AC approaches cannot address dynamic context-aware scenarios and they lack support of dynamic and adaptive policies [22]. For example, a client-server application programming interface requires a fixed set of variables as input, but does not take into account a context of operation [31].

In addition to the well-known AC approach mentioned above, many other approaches have been proposed in the literature [32, 33, 34, 35]. For example, using the break glass AC approach suggests that "it is possible for a subject to break-the-glass and explicitly override the denied request" [36, 37]. In addition, certain approaches can be used to transform one form of AC approach into another, e.g., such as a "framework that uses attribute-based policies to create a more traditional RBAC" approach [30].

An AC policy specifies how an access is managed and who may gain the access to information or service and under what conditions the access is granted. A standardized approach to represent AC policies is the XACML. XACML uses XML to encode a policy and an AC decision request/response language to execute a combination of policies and any AC request. This approach also provides a framework for describing data flow elements, including service requesters, resources, and AC policy decision points [38].

Traditional AC approaches consist of RBAC, MAC, ABAC and DAC. These approaches were successfully applied in different environments [39] to solve various

problems [40, 41]. However, all of them were designed to provide a relationship between pieces of information associated with an AC rule logic [42] and a resource or record for which access is requested. For example, for RBAC, these pieces of information [43] are an action or a resource and a component role. Their association within the AC rule logic defines a set of conditions [44], which can be extended with a number of attributes that are optional to the AC approach, such as location. This method allows the implementation of the AC approach to formally guarantee [45] that only the intended set of users [46] or components would be able to obtain the access to the resource or service [47]. However, the implementation of the AC approach can be a subject to manipulation, which can range from an unexpected situation [48], including poorly written AC policies [49] to a number of malicious entities acquiring access to a set of existing accounts with elevated AC privilege; in many of these cases the AC analysis [50] becomes invalid. At that point, a set of guarantees that the implementation of the AC approach was supposed to provide are no longer enforced [51]. Obviously, there are extensions to AC that can identify such problems [52, 53] including these involving pattern recognition. However, their possible integration into the AC approach is limited by narrow scope of these patterns and consequent insufficient level of precision including high number of false positives.

2.1.1 Role-Based Access Control (RBAC)

RBAC is "used to regulate access to systems, resources or information based on the roles of individuals within an organization" [54]. According to this approach, only "individuals performing certain roles have the ability to access required resources or perform specific tasks such as view or modify data" [55]. This approach provides fine granularity and scalability, and can be easily implemented for a small and simple system for which AC is provided consisting of only a few services.

The RBAC approach is based on following entities: users, roles, operations, and objects [56]. A permission to access a resource is represented as a pair of an operation and an object, and the permission assignment is defined as a pair of permission and role. In addition, user assignment is defined as a pair of a user and a role. Figure 4 provides an overview of this approach.

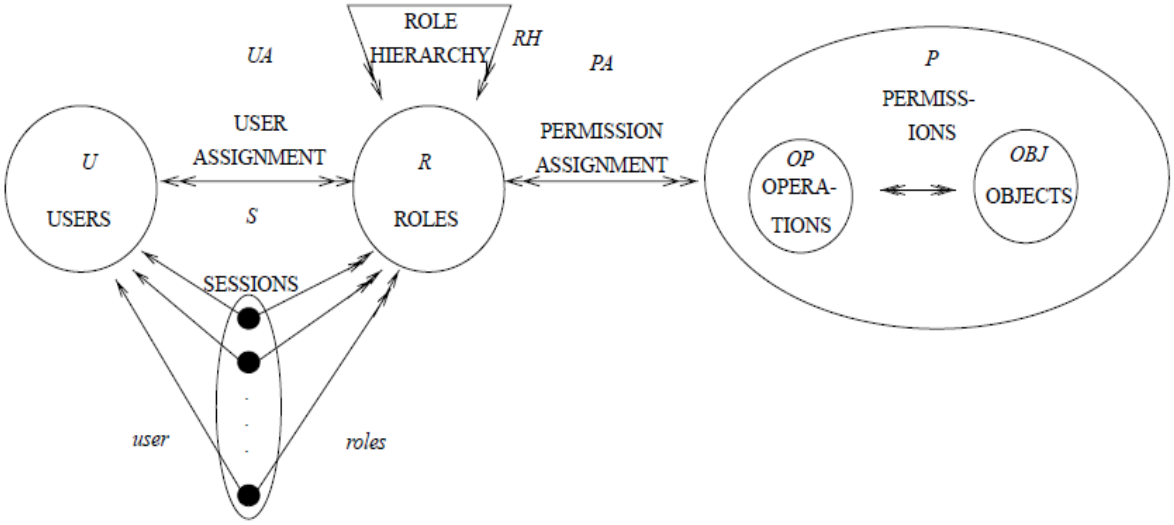


Figure 4. Overview of the RBAC model

Although an entire implemented AC approach based on RBAC can be reviewed by one analyst or a few analysts when adopted within a small system for which AC is provided, the RBAC approach becomes increasingly complex for a large system due to the high number of rule exceptions involving a wide range of users, different knowledge domains and diverse scenarios [57, 58]. This complexity may lead a security analyst to omit critical scenarios that need to be addressed. Further, AC rule exceptions made for certain cases accumulate [59] and the AC policies becomes increasingly difficult to understand. However, security analyst needs a good understanding while maintaining them [27].

2.1.2 Mandatory Access Control (MAC)

MAC refers to an approach in which "classifications or security labels are assigned to resources and access is granted only to entities (components, users, processes, devices) with a distinct level of authorization or clearance" [26, 60, 61]. This approach is often applied in systems with strict hierarchy, such as those used by the military.

2.1.3 Attribute-Based Access Control (ABAC)

ABAC is an approach in which "access rights are granted to users (e. g., components, persons, devices, processes) via the use of policies that combine attributes" [24, 62, 63]. These policies involve different types of attributes (e. g., component attributes, user attributes, resource attributes). "Time and space attributes are especially relevant when temporal and geo-spatial access control policies are required" [64, 65].

2.1.4 Discretionary Access Control (DAC)

DAC constitutes an approach that "restricts access to resources based on the identity of users (e. g., components, persons, devices, processes) or groups to which they belong" [22]. This AC approach is called discretionary because in this approach a user with specific access permission can pass that permission to any other user at his or her discretion. Therefore, a user owning a resource can grant or deny access to other users.

2.1.5 eXtensible Access Control Markup Language (XACML)

The XACML "is a widely used standard language for expressing access control policies. The XACML is developed by OASIS that uses XML to describe a policy language and an access control decision request/response language. The policy language allows administrators to define the requirements for accessing their application resources in a system for which AC is provided. The request/ response language describes request authorization queries and their responses. However, XACML lacks the knowledge representations needed to address computer-interpretable effects". XACML data flow is defined according to [66] and presented in Figure 5.

The XACML data flow contains ten components and thirteen tasks related to an AC request. The first action in the XACML data flow is for the personnel controlling an AC process to define policies [67] using a policy administration point. Then, the XACML-based implementation of the AC approach waits for a service requester to submit a request for a resource to a policy enforcement point, which forwards the request to a context handler.

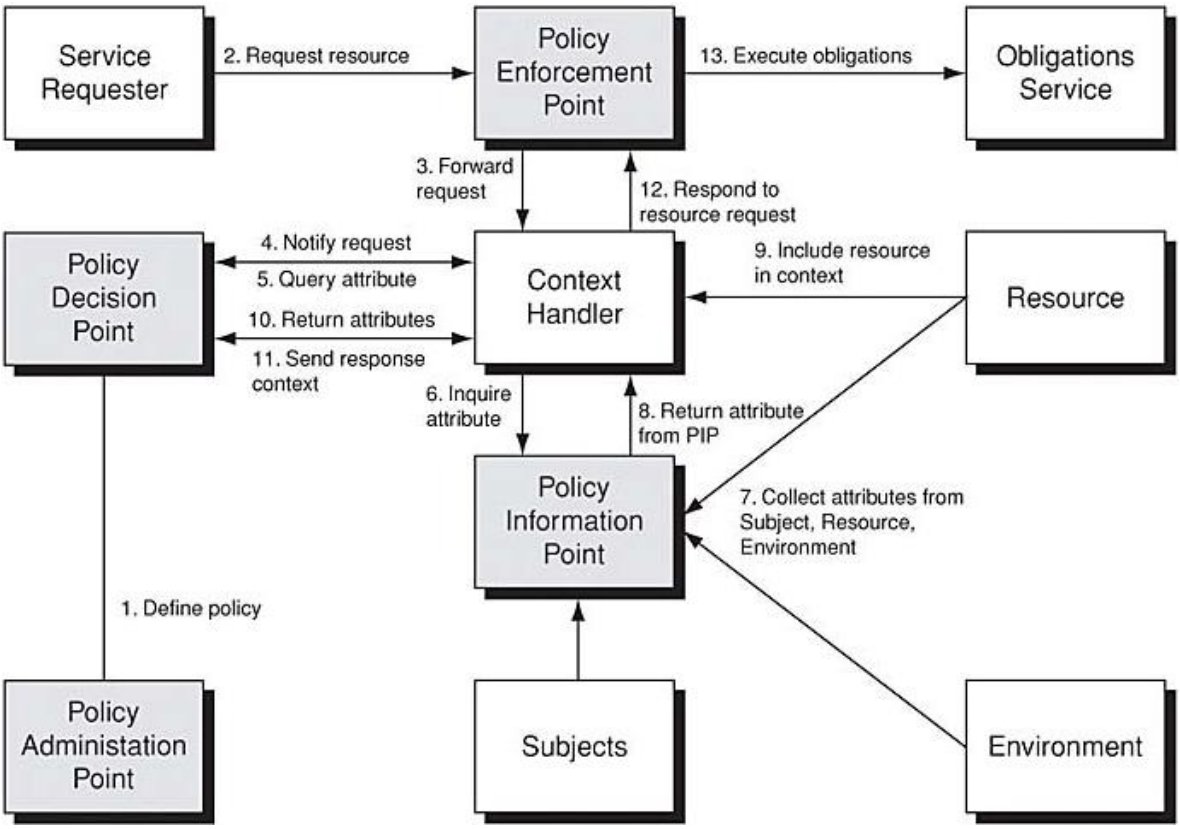


Figure 5. XACML data flow [66]

The context handler notifies a policy decision point of the new request that needs to be resolved. The policy decision point queries the context handler for a set of required attributes. The context handler forwards the request to a policy information point, which collects the set of required attributes from a subject, the resource, and an environment. Once all required information is collected, including the resource context information, it is then passed to the context handler, which, in turn, forwards it to the policy decision point. After the policy decision point made its decision regarding the requested resource, the policy decision point sends the AC decision with its context to the Context Handler, which forwards the request to the policy enforcement point. The policy enforcement point then executes obligations from the AC decision in an obligations service. If the AC request is resolved positively, then once obligations are fulfilled, the service requester can finally obtain access to the requested resource.

2.1.6 Obligations in Access Control

Traditional AC approaches offer a limited number of responses to a resource request. A common set of possible responses include ‘access granted’, ‘access denied’, ‘policy inapplicable’, and ‘result undefined’. However, conditions such as “Clearly there is no unique response to whether one option is better than the other, and which one is to be preferred depends on the specific context and information involved” [68] is not uncommon during analysis, and therefore, AC need more flexibility [69] in adjusting the component behaviour. “XACML defines obligations as actions that have to be returned to the policy enforcement point with the policy decision point response” [70]. Obligations examples include “obligation to anonymise data or an obligation that expresses authorisation timeout” [71], and “the policy may permit access to a resource but have the obligation that the owner of the resource is notified” [72].

2.2 Risk Adaptive Access Control (RAdAC) Based on Risk-Benefit Analysis

RAdAC has been an area of active research [73, 74, 75, 76] from approximately 2009 to 2016, and new methods have been introduced to resolve the problems inherent in traditional AC approaches, which, as software of systems for which AC is provided increase in size and complexity, start to compromise the AC approaches proper function and performance. RAdAC uses a risk calculation to determine whether access to resource should be granted or denied [77]. The risk calculation is based on a function that takes set of risks [78] into account through quantitative measures such as the ones involving reputation, security levels, and reliability [79]. In comparison with traditional AC approaches, RAdAC allows for a resolution of dynamic AC situation using the quantitative measures [80] in a way that minimizes entire collection of risks [81]. The functions that compute risk based on these measures rely on a risk-benefit calculation.

The risk-benefit calculation indicate that it is not enough to calculate just a risk of granting an access [82], and that four types of risks and benefits are required. The first type relates to a risk of granting access to resource. For example, if an AC grants access to certain

information, there is the risk that this information can be maliciously disseminated e.g., information leak. The second type relates to the risk of denying access to a resource. For example, if access is denied to a requested piece of source code, the developer might be encouraged to follow a path that will lead to project delays. The third type relates to a benefit of granting access to resource. For example, by granting access to a resource such as software code, a possible project delay may be reduced. The fourth type relates to the benefits of denying access to a resource. For example, denying access to a file on a heavily loaded server can lead to a load decrease.

A RAdAC approach uses “risk calculations to determine whether access to resources should be granted or denied” [83]. The risk calculation is based on a function that takes risks [84] into account through quantitative [85] measures such as the ones involving reputation, security levels, and reliability [86]. In comparison to traditional AC approaches (RBAC, MAC, DAC, ABAC), this approach allows for the “resolution of dynamic access control situations” [87] using a set of quantitative measures in a way that minimizes risks [88]. The function that computes risk based on these measures relies on risk-benefit calculations [89].

Regarding information leakage, "the proposed applications that are available to tackle incidents of data leakage from outbound email are proposed as additional plugins in commercial email security platforms (McAfee Data Loss Prevention, Symantec Data Loss Prevention), as standalone applications (WebsenseTruWeb DLP, Proofpoint Enterprise Privacy, MailMarshal) or as features in networking devices (CISCO IronPort email)" [90]. However, "several security software vendors now offer “data loss prevention” solutions that use simple algorithms, such as keyword lists and hashing, which are too coarse to capture the features what makes sensitive documents secret" [91]. In contrast, a RAdAC approach can provide the basis [92] for a solution that not only makes it possible for critical document features to be captured, but also to relate these features to business processes, process-based context and risk levels [93].

2.3 Risk Management Approaches

Risk management has been used since 1955 [94]. A number of standards have been developed [95, 96, 97] to resolve risks in general and to provide a framework and a methodology to handle new risks that are not yet handled in specific knowledge domains.

A proposed risk management approach is related to risk analysis. Risk analysis is a technique of projecting [98] known facts about the present and historical facts about the past into the future to provide a set of risks [99, 100] and probabilities of their realization [101, 102]. Risk analysis has been applied for various purposes over the last 50 years in various knowledge domains [103, 104], including engineering [105], finance [106], and security [107]. Overall, risk management helps to mitigate risks by providing methods for their rigorous identification. Several standards were developed to provide specific risk analysis in an AC system [108, 109].

There are several approaches that provide risk management in specific knowledge domains. For example, in information security, Bayesian networks [110] are used to "better understand the causal relationships between preconditions, vulnerability exploitations, and postconditions. This is facilitated by computing the likelihoods of different outcomes possible as a result of the cause-consequence relationships" [111]. Risk occur in every aspect of knowledge, for example there is an entrepreneurial risk [112]. M.K. Sadgrove states that it "applies to any management decision that could have a good or bad outcome. It follows that most management projects and decisions contain risk" [113]. Another kind of risk management occurs in the health care field/industry, for instance, when physicians consider genetic testing for at-risk relatives of cancer patients [114].

In summary, the existing approaches to risk management vary from the general [115] to the specific and are designed to facilitate risk analysis and to develop risk mitigation strategies. However, the process of combining risks originating from different knowledge domains can be improved.

2.4 Business Process Modelling and Simulation

A risk management approach is related to business process modelling [116, 117, 118] due to the need to forecast the future state of the system for which AC is provided via simulation

[119]. Simulation relies on modelling of the business process to present a model [120] that will be used to assess a set of relevant risks. “Processes seen as systems might be modelled using simulation for the purpose either of understanding the behaviour of the process or of evaluating various strategies for the operation of it either for decision-making or for learning purposes” [121].

A simulation can be defined as a process of creating a FM as an abstract image of any system for which AC is provided to analyze variables and tasks controlling a state of the system to project this state into the future. Any existing and proposed systems for which AC is provided described quantitatively, using attributes, equations, and laws can be simulated.

The goal of a simulation is to define mechanisms controlling a behavior of a system for which AC is provided. Moreover, the simulation can forecast a future state of the analyzed system and determine the best way to impact a state of the system in order to achieve its goals. Thus the identified required alterations to help an AC system to achieve its goals can be performed.

For example, performing a number of asteroid trajectory simulations each modified by a possible way of an asteroid trajectory alteration can lead to a set of quantitative tasks to make sure that an asteroid does not impact any space vessels, or Earth. Indeed, an asteroid impacting Earth is a very real and ever-present possibility and a lot of efforts has been made to identify such threat to Earth as early as possible. Simulation is an essential method relevant to any spacecraft design or a mission involving asteroids, which include elements such as an asteroid identification, its orbit determination, and collision risk mitigation.

Risk assessment accuracy is a critical feature of the proposed approach [122]. FM of the approach is complex with its several risk dimensions [123, 124], and thus, many details in a simulation are considered to forecast a possible future with a sufficient level of accuracy. Forecasting allows prediction of multiple details, including critical risks and utilities for a set of dimensions in one run. “Having accurate forecasts about the performance of the system based on an on-line fine-tuned simulation model also means more accurate decisions at any point in time” [125].

Chapter 3

Description of the Approach

The proposed approach extends existing AC approaches, including RAdAC-based ACs. This chapter describes the overall structure of the approach and the functions of each of its parts, as well as details about how the approach can be applied. The main formal definitions of the concepts pertaining to the approach are also presented.

3.1 Approach Architecture

The proposed approach is an extension of the XACML model presented in Figure 6. Notice that this model uses static AC policies that are applied to control the access to a resources of a system as depicted by the shaded area in this figure. In the case of a banking application, for example, to get money (i.e., a service, box 2) from an ATM (i.e., following a business process, box 3), when a bank customer (i.e., a component, box 1), according to the bank's static AC policy, needs to access (i.e., AC request, box 5) his or her account, a PIN is required by the ATM. The PIN has to match the bank record associated with this customer, and if the customer is unable to provide the PIN or provides it incorrectly the bank's server (i.e., more specifically via a policy enforcement point, box 4), a decision is made by a policy decision

point (box 6) to deny access to the account based on a set of static AC policies (box 7). In this example, the AC process is restricted to a single option, that is to provide a PIN.

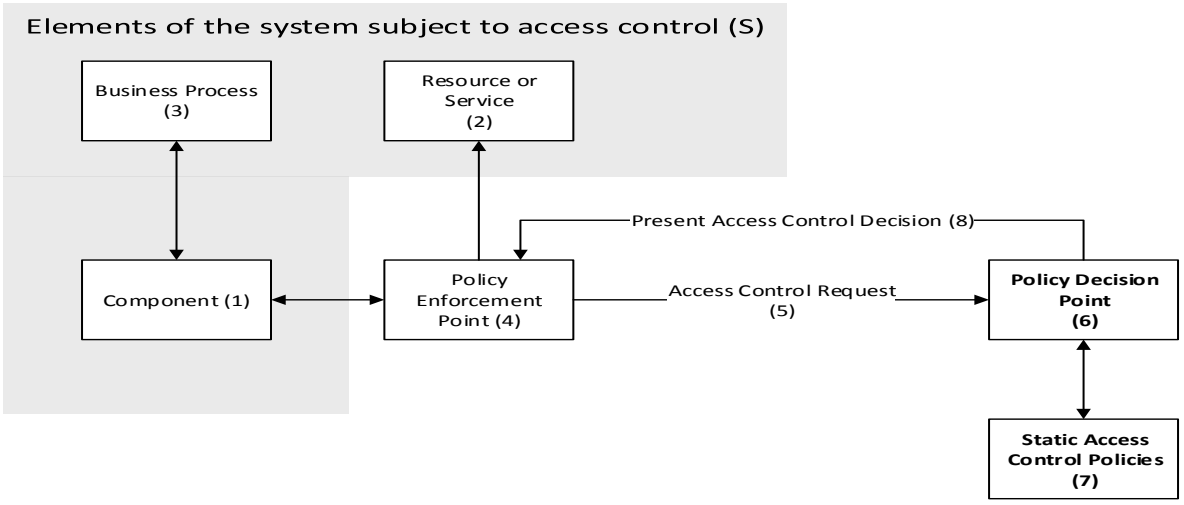


Figure 6. XACML architecture

The proposed approach architecture is presented in Figure 7.

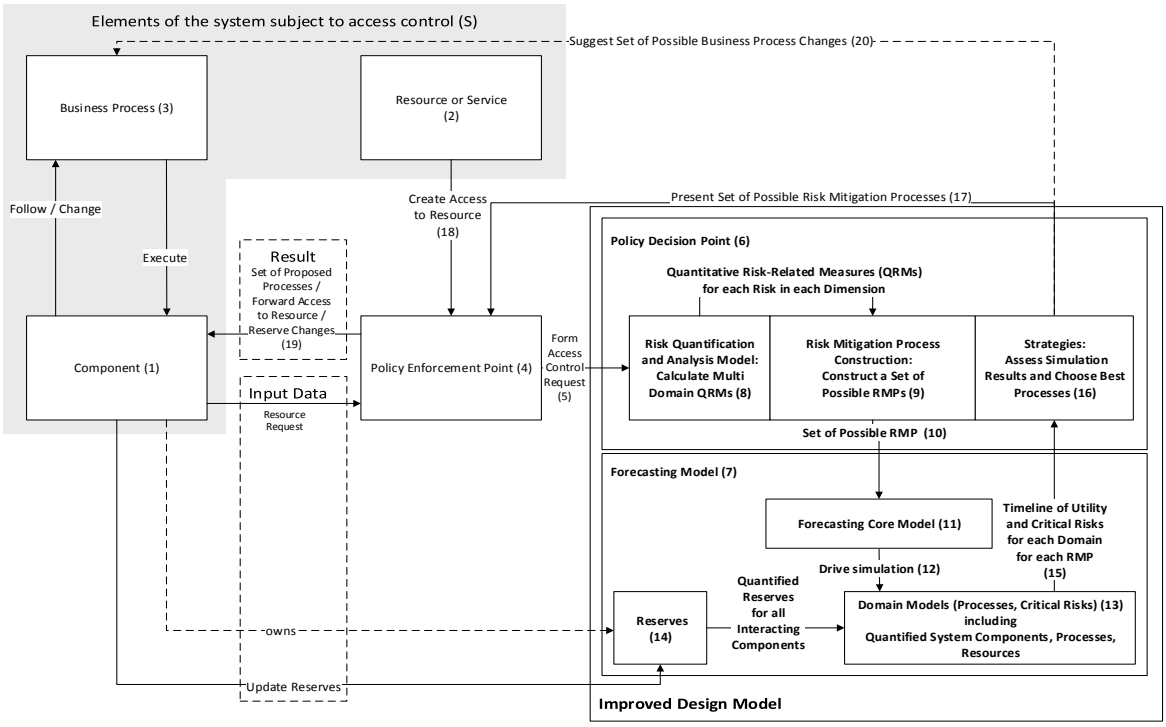


Figure 7. Proposed approach architecture

No alternatives are provided that take into account AC process variations related to different identification methods or to different risk levels of a transaction (e.g., checking the account balance is less risky than withdrawing money). First, AC decisions made by the policy decision point (box 6) is based on a set of static AC policies (box 7). Second, the AC decisions can only follow a predefined AC process that is unable to adapt to variations such as the ones previously described, for example, to different identification methods and different risk levels. Therefore, the traditional XACML model falls short in providing a dynamic and more flexible AC process. For example, in terms of a banking application, if customers forget their PIN, but have their cell phone with them, the banking system can send them a message containing a verification code to use as a substitute for requesting a PIN; also, if a customer frequently visits this location, it might be possible to omit the request for a PIN in case there is a positive match provided by facial recognition software and the customer wants to perform a low-risk transaction (e.g., to pay his or her credit card balance). Beyond these examples, there is a wide variety of other variations related to customer identification, risks, and other factors, that could, but are not currently supported.

In addition, it would be beneficial to support the AC process variations such as the ones described in the ATM example (page 4), which include among others variations related to non-card based transactions, facial recognition, additional security questions, signature requests, and additional security codes sent to a cell phone. These cases further illustrate examples of possible AC process adaptations that could be made, but are not part of a static and non-adaptive AC solution.

Besides providing variations involving access to resources based on existing access privileges, another important class of variations are those that restrict access privileges, which can happen especially in the context of malicious threats. For example, in a banking application, unauthorized access privileges resulting from ATM software bugs or exploits can be used to make the ATM dispense cash, and it would be helpful if the AC process can limit the remote unauthorized access once the ATM starts to malfunction. As another example, in a patient-privacy application, stolen doctor's credentials can be used to obtain unauthorized access to a number of privacy-sensitive medical records, and it would be appropriate if the AC process could limit the unauthorized access.

In order to address the limitations described in the previous paragraphs regarding the XACML model policy decision point (box 6) and static AC policies (box 7), a new dynamic and risk-based AC approach needs to be introduced. In Figure 7, which shows the proposed approach, the two boxes are replaced by parts that support dynamic AC process adaptations based on risks. These parts are represented in this figure by boxes in which bold style text is used. In essence, the new boxes 6 and 7 consist of a new policy decision point and a new FM that will be described in the following paragraphs.

The policy decision point from the XACML model is replaced with parts (box 6 in Figure 7) that still need to make AC decisions, but they achieve this purpose in a very different way because these parts need to make the decisions by following the tasks related to a business process, assessing the risks and benefits related to these tasks dynamically, and constructing sequences of tasks that minimize the risks. Risks and benefits are counterparts in describing the negative and positive impact on the system and are measured through their utility, which is a number that represents the level of the risks and benefits. For example, some tasks such as adding a rule to the firewall in a software development setting to allow developers to access software code remotely may lead to an increase in the risk of this code being stolen because the new firewall configuration is more prone to threats. However, asking the project manager permission to add such a rule may reduce the risk of this code being stolen, because the project manager will be aware of the timetable of the developers and may be able to identify and prevent unauthorized access to the source code outside of regular working hours. Therefore, in this case, constructing and executing a sequence of these two tasks would minimize the risks related to source code being stolen.

The static AC policies are replaced with the FM (box 7 in Figure 7) that is designed to project the constructed sequences of tasks into the future to find the outcomes of these sequences. A projection is performed through a simulation involving the same elements of the system subject to AC, which are depicted as a shaded area in Figure 7, that function as instantiated components in the FM. The outcomes of these sequences are the timelines of the risks that have occurred during the simulation. For example, projecting into the future a task of adding a rule to the firewall in a software development setting to allow developers to access software code remotely in a course of one day can result in an extra 22.5% probability

of the code being stolen compared to the previous configuration of the firewall. However, as time to access a source code is also reduced, there is also a 25.4% increase in productivity of the software developers. Thus, projecting the constructed sequences of tasks into the future allows the approach to assess and quantify the risks related to the simulated execution of the constructed sequences of tasks.

The new policy decision point (box 6) involves three parts: the risk quantification and analysis model (box 8 in Figure 7), which is responsible for converting the input containing data received from a number of different data sources into quantified risk-related data; the RMP construction part (box 9 in Figure 7), which provides a process to mitigate risks; and a part comprising a set of strategies (box 16 in Figure 7) that assess the timelines of risk utility in a form of utility graphs and choose the best applicable RMP candidates, that is, the sequences of tasks that minimize the risks. These sequences will be provided to a requesting component.

The FM also includes three parts: the forecasting core model (box 11 in Figure 7) that initiates and drives the projection of each of the RMP candidates; the domain models (box 13 in Figure 7), which represent the behaviour in domains such as banking and health; and reserves (box 14 in Figure 7) that are owned by components and represent previous accumulated risks and benefits as resources that can be used to mitigate future risks.

In the next subsections, details about each part in box 6 and box 7 in Figure 7 are provided. Together, these parts constitute the new design model introduced by the proposed approach, which is called “Improved Design Model” in Figure 7.

3.1.1 Risk Quantification and Analysis Model

The first part of the new policy decision point, which is described in this subsection, is a risk quantification and analysis model (box 8 in Figure 7) designed to convert the input containing data received from several different data sources into quantified risk data. For example, in a software development setting if an non-experienced developer starts to write code that can

result in a 30% risk of a two-day project delay assuming this task is critical in the project timeline. This number can be derived from a historical database of projects based on entries containing information about developers that have similar experience and estimated task durations. In general, the risk quantification and analysis model is used to identify and quantify risks, and perform a risk analysis that depends on the current state of the system.

The risk quantification and analysis model receives an AC request as input. The AC request contains a requested resource (e.g., software code in a software development setting or a medical record in a patient-privacy setting) or service, an action type such as read or write, and the requesting component role and risk-related attributes (e.g., in a health setting a doctor and the number of resolved emergencies that he or she can handle). Based on a specific resource type and action type, certain risks can be applied. For example, in a patient privacy application access to a medical record might result in the unauthorized dissemination of this record, but refusing this access request can result in poor treatment quality. The risks to be applied are quantified through relevant risk related attributes known as QRMs. These QRMs are used in the risk analysis, which results in risk probabilities (e.g., the probability of the unauthorized dissemination of a medical record). These probabilities follow, for simplicity purposes, a normal probability distribution. For more details about the risk analysis refer to Appendix D. The output of the risk quantification and analysis model is a set of risks that must be addressed by the RMP candidates and set of QRMs describing the risk-related attributes.

RMPs need to deal with both risks and benefits, where risks lead to negative consequences and benefits lead to positive consequences. For example, the risk of unauthorized medical record dissemination can lead to patient privacy violation, while the benefit of asking a medical doctor for an additional authentication reduces the probability of the risk of unauthorized medical record dissemination. Each identified risk that must be addressed by the RMP candidates is defined by an event or a set of events and the probability of the consequences of these events. Examples of risks are the risk of unauthorized dissemination of patient medical records, the risk of poor quality of medical treatment resulting from insufficient information, the risk of software product completion delay and the risk of source code leaks. Example of consequences related to these four risks are

penalties for violating patient's privacy, providing treatment of poor quality, delaying product completion, and loosing reputation for unauthorized source code disclosure. The consequences can be either static or depend on the context and attributes of a system for which AC is provided. The probability of these consequences is defined by the likelihood and its dispersion (i.e., standard deviation). In addition, each risk is associated with one risk dimension, which is a group of risks related to a specific knowledge category such as security, productivity, and privacy.

Each task sequence has a set of risks and benefits associated with it. The goal of the FM simulation is to project based on the current state of a system for which AC is provided each RMP into the future states of the system. After the simulation the future states of this system related to each RMP are compared and one or more task sequence is produced. For example, in a patient privacy application, when a doctor is handling an emergency, the requested information has a critical time to be delivered and if this information is not delivered at the proper time, the patient could die and the information would no longer become relevant. In addition, for example, in a software development setting, if the involvement of a developer in the development process is critical and the deadlines for project completion are achievable only if access to the source code is granted, the AC might choose to set an exception in the firewall and thereby allow access. This access leads to the benefit of reducing the project delay risk. However, the approach has to compare this benefit with a security risk that is created when the source code is allowed to be downloaded to a location that is deemed insecure. The approach can choose other tasks (non-mutually exclusive tasks) to mitigate the security risk of allowing access, such as verifying and or installing an antivirus, a firewall, disabling any potential access to any third party can have to this machine, or taking any additional steps to verify the identity of the developer. Given that these steps sufficiently mitigate a security risk and it would no longer outweigh the productivity benefits, the approach might accept such course of action.

FM (6) is a model that represents the components of the system for which AC is provided, the external entities that communicate with these components, and the processes of the external entities. These processes are represented as directed graphs in which the tasks are vertices and the edges denote task sequences. A process uses context as one of its

attributes and the context is a set of dynamic variables that can influence the task sequence executions at runtime. In this sense, process executions can be called context-aware.

As an example of risk analysis in a software development setting, this analysis typically uses historical databases of software development projects. Using these databases experts can estimate the duration of projects and these measures can be used in the risk analysis. For example, project delays can be calculated using well known project estimation techniques as the time $T_{delay} = T_{passed}*(T_{reserve}-T_{estimated})/T_{reserve}$, where T_{passed} is the current project duration; $T_{reserve}$ is defined as the duration of the critical project path minus the time left to complete the project on time; and $T_{estimated}$ is the current estimate of the remaining time to complete the critical path. However, in order to calculate the project delays more accurately, the risk analysis has to take into account other factors such as the availability of resources. Of course, if more information about which resources are available or not available is provided, the more accurate the project delays can be calculated and, therefore, the more accurate the risk analysis will be.

The proposed approach is based on the RAdAC method. However, in contrast with the existing RAdAC approaches, this approach is designed to be modular and extensible in that additional risks from new risk dimensions can be added to or removed from the risk analysis. The analysis is conservative: the results of the risk analysis based on one risk dimension are preserved when additional risk dimensions are introduced. In this way, it is possible for the FM to address not just one risk dimension, but a set of them (e.g., computer network, financial risks). The combined risks derived from the FM allow for a detailed projection of the processes into the future that involves a wide variety of interrelated risk dimensions. This evaluation of the combined risks is too complex to be compiled manually by experts.

The risk analysis can be used both in RBAC (to infer what resources can be accessed by certain groups of users) and in RAdAC (to conduct static risk analysis). However, both approaches rely on a manual risk analysis paradigm and do not create possible projections over time, whereas in the proposed approach the risk analysis is automated and the projections are addressed provided over time.

3.1.2 Risk Mitigation Process Construction

After the risks are identified and quantified there is a need to construct a sequence of tasks, that is a process that mitigates these risks. That is a responsibility of the RMP construction component (box 9 in Figure 7). For example, in a patient-privacy setting, when a medical doctor has identified an emergency, it may be useful to request other medical personnel participating in the emergency resolution to confirm that this is indeed a medical emergency as opposed to a situation in which a malicious entity has gained access to one of the medical doctor credentials and is abusing the emergency procedure to gain access to data that would not be provided otherwise. Thus, the RMP construction provides as output a set of processes where the risks are mitigated and each of these processes can serve as a candidate in the AC decision.

As mentioned previously, the processes are defined by directed graphs in which the vertices are tasks. The process description contains the following attributes: the name of the process graph, which is used for an identification process; the entry points, which is a list of starting tasks; and the exit points, which is a list of ending tasks.

The results of the risk quantification and analysis model are forwarded to RMP construction (9), which takes as an input all relevant risks and risk-related attributes and constructs a set of possible RMPs (10) addressing the identified risks such as adding a task for adjusting the firewall rules to prevent code retrieval in a software development application or adding a task for shredding printed documents after a medical emergency has been resolved in patient-privacy setting. These RMPs (10) are an output of the RMP construction (9) and they contain the set of processes to address the identified risks. However, to identify the best RMPs among the constructed set there is a need to project each of them into the future using the FM (7) and compare them based on the projected risks.

Figure 8 presents examples of an original process and a constructed set of three RMPs for a software development application. The initial software development process can be seen as a business process (3) in Figure 7, which a component (1) follows before an AC decision is reached. Some tasks may be mutually exclusive, which may lead sequences of tasks to be mutually exclusive. For example, options 1 and 2 are mutually exclusive sequences of tasks

because the AC cannot allow and deny access at the same time. Option 2 represents an original process modified by adding a mutually exclusive task (that is, to allow access) and a mutually non-exclusive task (that is, to adjust firewall rules). Option 3 is option 2 extended with one additional mutually non-exclusive task (that is, to ask the project manager for permission).

The RMP construction is in general a complex process, because the exploration of possible sequences of tasks that mitigate the involved risks involves a large number of possible task combinations. In order to provide a feasible solution the approach initially assesses all mutually exclusive tasks. After that, each mutually non-exclusive task, if there is any, is added to the task sequences in various positions to generate other task sequences, and these sequences are compared (16) with the previous sequences and compared with themselves based on the timelines of the risks (that is, the risks that are projected over time). The prerequisites of tasks and the task parameters (e.g., a code fragment identification for the task "get code") define the order in which tasks (including tasks from the original process) are positioned. In selecting a possible task to build a RMP, a number of process-related attributes is used, including the component (1) role, resource type (2), action and context. For example, in a software development application, when there is a need for a software developer (role) to move (action) code (resource) from one location to another during his or her normal work hours (context), in order to reduce the risk of code being stolen, the task "adjusting firewall policies" is selected to limit the exposure of the code elements through adjusting the firewall policies.

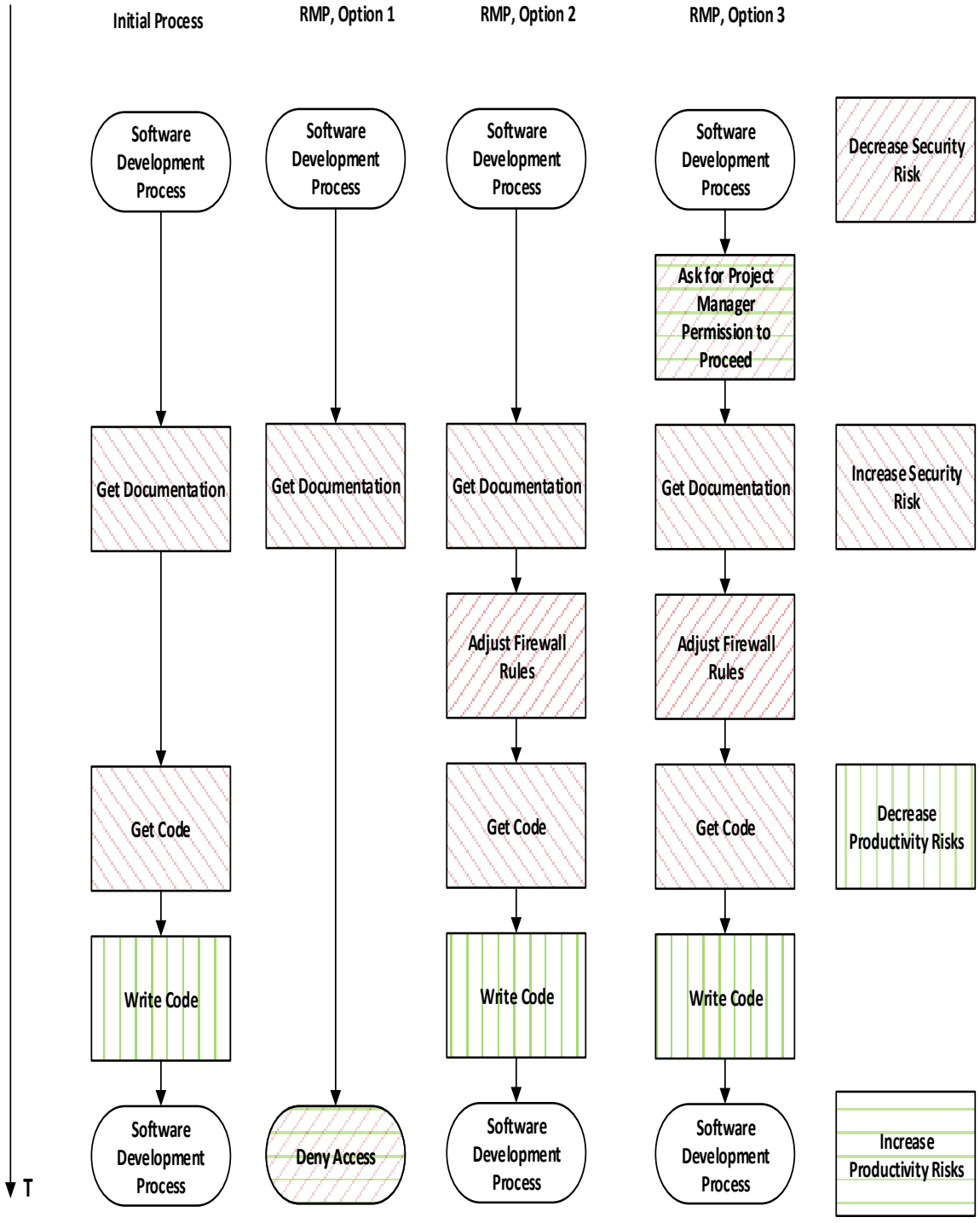


Figure 8. Examples of the original process and the constructed set of RMPs for the software development setting

Figure 9 presents examples of the original and the constructed RMPs for the patient privacy application.

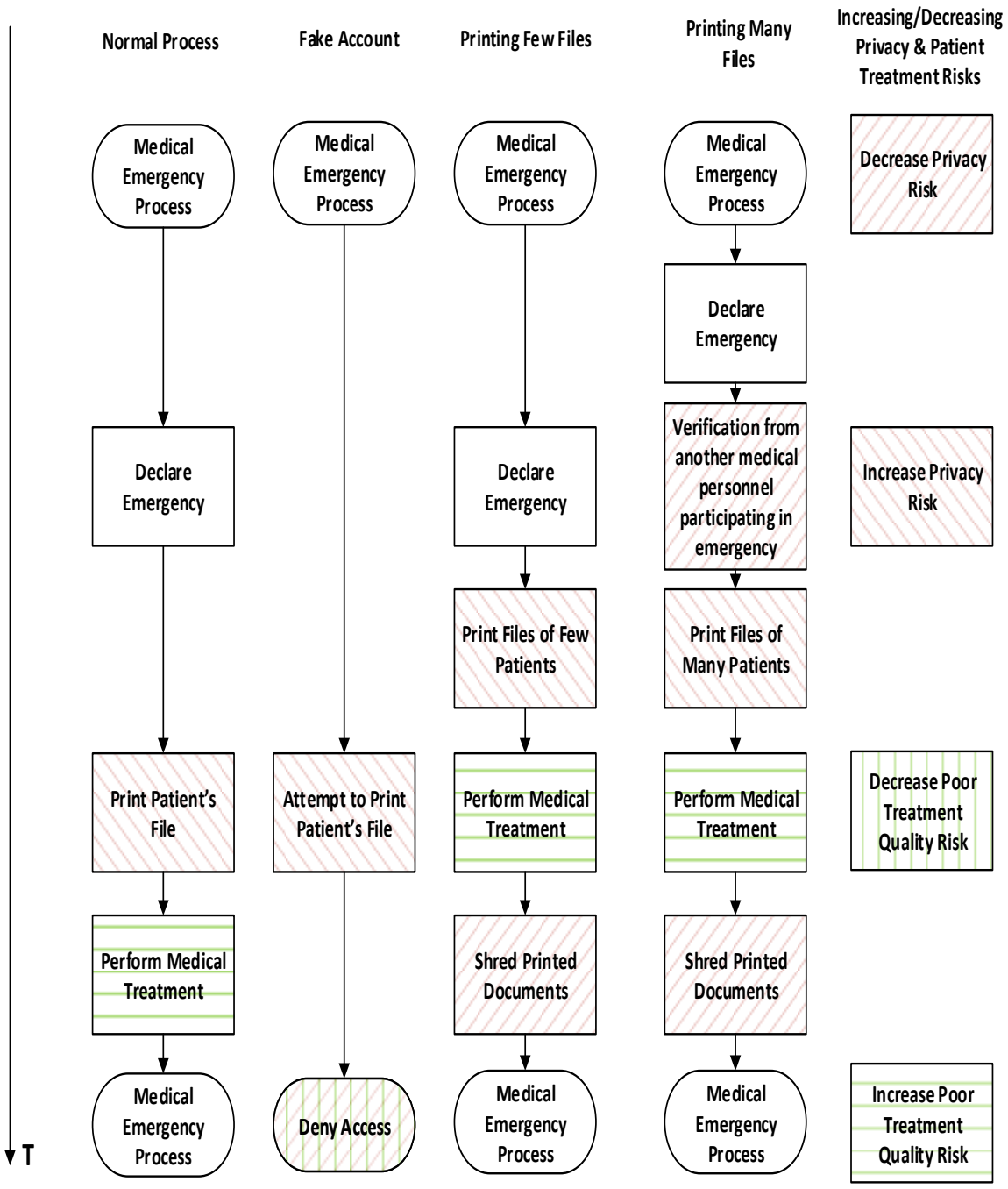


Figure 9. Examples of the original process and the constructed set of RMPs for the patient-privacy application

3.1.3 Forecasting Core Model

As it was previously mentioned in subsection 3.1, the FM is designed to project RMPs into the future to find out the outcomes of these processes in terms of risks. The forecasting core model (box 11 in Figure 7) initiates and drives the projection of each of these RMPs, where a projection is a simulation of the processes taking into account the elements of the system for which AC is provided, namely the components, the business processes, and the resources used in a specific application. The simulation is event-based and driven by one or more foci of its components. Each event represents an occurrence, such as an action execution or a context change, within the simulated system. For example, an event happens when a doctor is notified that the requested medical record is available. There are cases in which one event is associated with risks from multiple risk dimensions. For example, allowing access (event) to source code provides productivity benefits and constitutes security risks at the same time. The focus of a component is a pointer to a task of a process graph that is being executed, and the focus pointer is stored as an attribute of some components. An example of a focus is a pointer to the task of retrieving a medical record that the doctor, which is represented as a component, is performing to resolve an emergency.

Some of the components that represent human users are associated with a role. This role describes a possible class of components which is associated with a specific process graph. For example, a specific doctor (a human user represented as a component) has the role of a medical personnel associated with a process graph that represents medical routines. The role has the following attributes: the location, which refers to the current location of the user with this role; the role name, which is the name of the role and is used for identification purposes; the computational unit for work that points to the computer selected by the user working in this role; data sources, which is a list of applications known to this particular user, including usernames and passwords that represent a user's memory of applications he or she has access to; and tasks, which lists the current tasks the user performs in this role.

The process execution starts with the focus pointing to one of the starting points of a process graph and traverses the process graph selecting and scheduling new tasks until it executes one of the ending tasks of the graph, at which point the process is considered to be

completed and no future tasks are scheduled. In terms of each task, the process execution schedules an event for the execution of the task logic, which is the behaviour of the task.

The task concept serves two purposes. First, it describes the current state of the task that is executing and, second, it is used as a template from which the actual tasks are copied when instantiated. The task has two relevant methods, which are called choosecustomnext and taskactions. Because a task is represented as a template these two methods need to be defined when a task is introduced. The choosecustomnext method is invoked within the choosenext method, which selects the next task that must be executed after the current task has been completed (that is, task switching). The taskactions is called when the task event has occurred (that is, the task logic). The task class has six attributes. These attributes consist of: the task name, which is used for different task identification purposes; the standard duration, which is an expected duration of the task; the actual duration that initially is equivalent to a standard duration, but can be changed depending on the results of the execution of the previous tasks and on the process attributes used to set the time at which the event of completing this task must happen; the parent, which is an object responsible for the task execution and that can be a role or a component; the status, which refers to the status of the task execution and can be a success, failure, or not defined; the next nodes attribute, which is a set of possible next tasks to be executed after this task is completed; and the previous nodes attribute, which is a set of tasks that can lead to the execution of this task.

Each task has a method called taskactions which represents the task logic. This method includes a number of execution control structures such as conditions and cycles, and three types of operations:

1. Send message: during this step, the component executing the task sends a message to another component.
2. Update attributes: the component's actions can update attributes, which can be associated with a task, a process, or the system for which AC is provided. For example, the component can set a task status (one of the task attributes) to failed when an error that cannot be corrected within this task has been encountered. Another example occurs when the component acting on behalf of a software developer creates a new database. The database activity needs to be forecasted by the FM. In this case, the new component or database is referenced both in the

process attributes, where the component would populate the database after the creation in the same process, and in the attributes of the system, where other components can also query this database.

3. Perform action: the component can perform an action such as creating a new record or storing information about a new computer that has been added to the system dynamically. An example in a software development application of a record is when a server creates a new user account when there is a need to record information about a new user to allow him or her access to the system. In a patient-privacy application, when a document is being printed, the database that is tracking the state of all printed documents creates a new record that specifies details about this procedure.

The simulation results in the timeline (15) of utility (that is, the level of risks and benefits) and critical risks. Risks have a probability of occurrence and an impact associated with them. The impact of a risk is a negative consequence that will occur if the risk occurs. For example, if there is a risk of code being stolen in a software development application, if the code happens to be stolen there is a consequence that is in this case a financial loss. The utility of a risk is defined as the risk impact multiplied by the risk probability. In general, risks can be divided into common risks and critical risks. Common risks have an impact that can be dealt with using the existing system reserves, meaning that in the case of a financial penalty there are enough financial reserves accumulated in a system to cover the financial loss. Critical risks have an impact that is prohibitively high and exceeds the accumulated reserves. In this case, it does not make sense to calculate a risk utility, because the impact is too high. Instead, only a probability is used to characterize these critical risks. For example, if a company goes bankrupt, individual productivity cannot not be taken into consideration anymore, and therefore the risk analysis that involves risk dimensions such as productivity can no longer be applied.

The forecasting core model design consists of a set of classes. These classes are presented in the UML diagram depicted in Figure 10. That is a simplified UML diagram, presenting only the data and selected relationships among the classes. A detailed diagram is provided in Appendix B.

The UML diagram displays classes using boxes with three sections for general UML and two sections for data only diagrams. The top section states whether it is a class or an interface, and the class and an interface name and its package. The middle section contains the class members except where the class members are represented by an association link. The links are represented by a solid line and a non-triangle arrow, e.g., from recordscollection class to record class, where the type and name of the class member is represented in the text, the public access type is shown as circles and the protected access type is shown as triangles. The final class members are depicted with the F letter modifier and static members are represented with S letters. The bottom section (not present for data diagram) contains signatures for class methods. Please note that the class called actions queue represents events queue.

The class signatures consist of the method name, method parameters, the value returned by this method, and method access level, where a circle represents the public method, an up arrow represents the protected method, a C character next to an access type represents the constructor method(s) for this class, and a underscored method name with an S symbol next to the access type is a static method. In Figure 10, the class dependencies are indicated with dashed arrows (e.g., from risk collection to events queue).

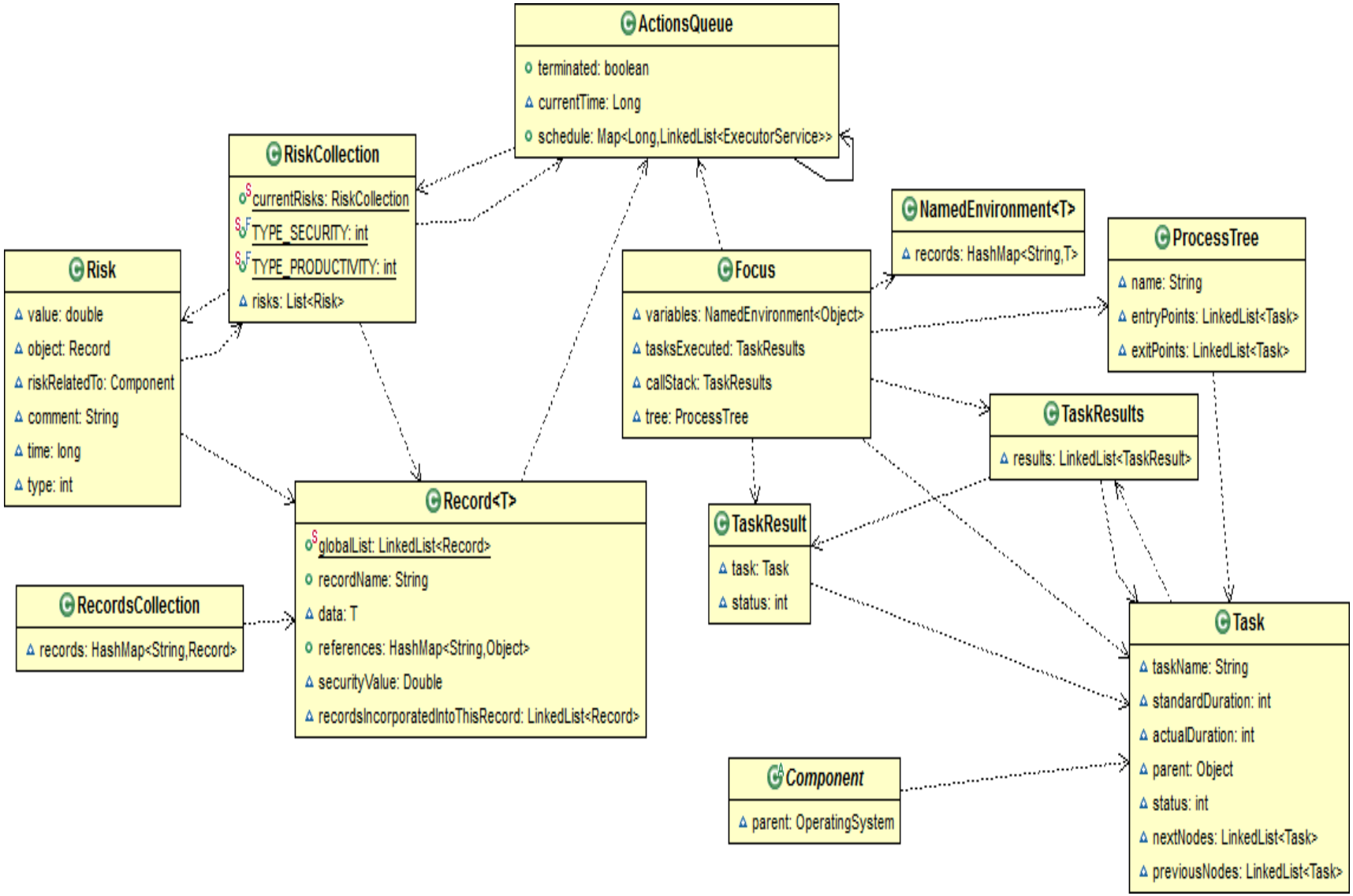


Figure 10. UML class diagram, forecasting core model

After having described the forecasting core model, the details about the simulation process are provided. First, the description of the concept of scenario is introduced, which specifies a domain-specific situation, which includes the components (such as doctors, software developers, computers) and their behaviour. After this, the concept of focus of components, which was previously introduced as a pointer to a currently executing task, is described in more detail. Then, an in-depth description of the routines for task execution and for the next task to be executed. The explanation about how events are processed continues in this subsection, providing details on how events are scheduled, stored and executed. Notice that depending on the events that occur the risks may increase or decrease. Finally, an overview of the FM behaviour is provided. The discussion starts by describing how to represent domain-specific situations as a scenario.

Scenario

Scenarios are represented using the following elements: components both internal and external to the system for which AC is provided, the processes that the components go through (sequences of tasks), and resources that are needed by the components to perform their processes. For example, in a patient-privacy domain, a description of a scenario can represent a situation, for example, when a doctor performs an emergency procedure with the help of two assistants to save the life of a patient who suffered a trauma in a traffic accident. In order to perform the required medical procedures properly the doctor in this situation needs to know a set of medical attributes, which may include the patient's blood type required to perform a transfusion. This description is complemented by many risk-related details, including the level of experience of the doctor and the list of previous accesses to this patient medical records. These details are used to instantiate a FM because they contain the necessary domain-specific information required to project the system into the future.

A scenario can be associated with a number of risks. For example, in a patient-privacy domain, two risks are considered, namely risk of unauthorized medical record dissemination and the risk of poor treatment quality. Because of the specified risks, risk mitigation tasks are also represented in a scenario. For example, based on the same domain, risk mitigation tasks need to be specified to (i) provide the doctor with a required set of medical records, including the specification of a patient's blood type; (ii) refuse access to a requested set of

medical records to enforce the patient's privacy; and (iii) shred the printed medical records. A refusal of access to the patient's medical record (ii) can happen in case the AC believes that the security of the doctor's credentials has been compromised and, instead of a genuine emergency, the AC is dealing with an AC request possibly created by a malicious entity attempting to gain access to the patient's medical records. As it was previously mentioned, these risk-related details are used in the FM projections.

Focus

The focus can be activated either when a scenario is initialized at the beginning of the FM simulation or by another focus within its task logic execution. The activated focus must follow the specified process graph. Once the focus is activated, it selects one of the process graph entry points, either the task switching construct is used or, if there is only one entry point, this entry point is selected. The selected entry point is set to be the next task to be executed in an event scheduled for a nearest available time. That selection starts the event lifetime, which keeps the focus in a loop until the last task of this process graph is not completed, the last task is defined as a task that is in the list of a process graph endpoints. One this last task was executed and the selection of next task routine identifies that it has executed a task present in a list of process graph endpoints the focus is deactivated and the process is considered complete.

The behaviour related to the concept of focus that is described in the previous paragraph is represented in Figure 11.

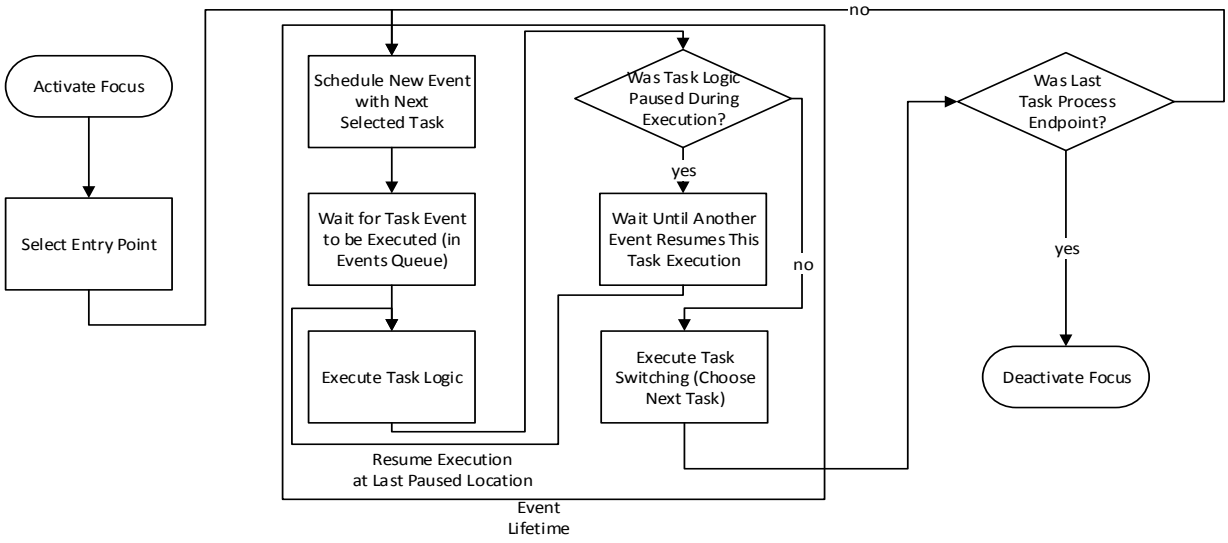


Figure 11. Focus and event lifetimes

Task logic

As previously mentioned, the process graph is a directed task graph. Each node of the graph represents a possible task that can be executed. Each task consists of two parts, the task logic that is presented in Figure 12, which contains the execution logic of the particular task, and task switching, which defines which task should be selected for execution after the current task is completed.

The task logic is represented by a method called taskactions mentioned in task definition. Figure 12 represents the flow of execution of a task. The diagram uses control structures that are represented in the form of predicates. A predicate is a logical function that takes into account predefined attributes. Control structures are used to alter task execution control flow (modifying J and K in Figure 12) or, depending on certain predicates, execute a certain action or skip the execution of an action.

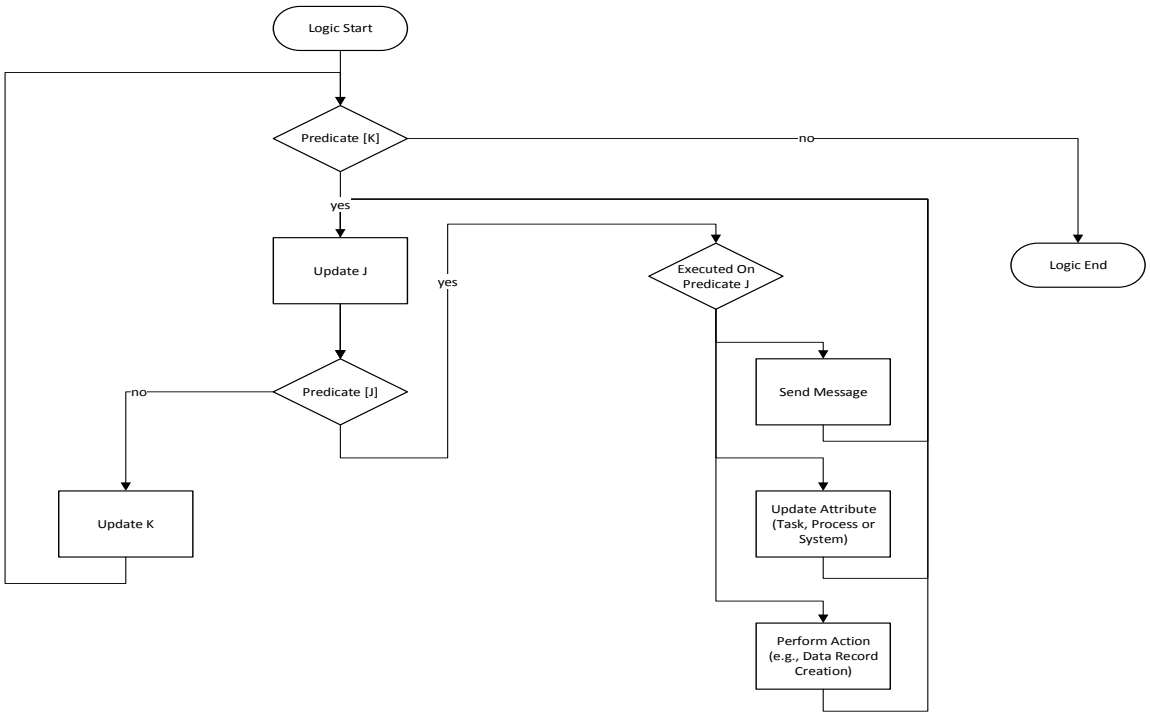


Figure 12. Task logic

Task switching

Task switching is used to select the next task that will be executed after the current task has been completed. The task switching flow of execution is represented in Figure 13. The result of the task switching execution is another task to be executed after the current task is completed. As previously mentioned, the focus is a pointer to a process graph node and it traverses over a process graph containing the tasks that can be executed in a process. Task switching defines a way the graph is traversed. For example, in the case of a developer trying to download source code (e.g., which is a represented by a task) and failing (e.g., which is represented by a task attribute, known as status) once (which is derived from the history of previous task calls), repeating the same task can lead to a success. Therefore, task switching would set the next task to be executed to be the same task, but when the second attempt fails, the component representing the developer assumes that it is not possible to obtain the source code. This component then can try to download the code from an alternative location.

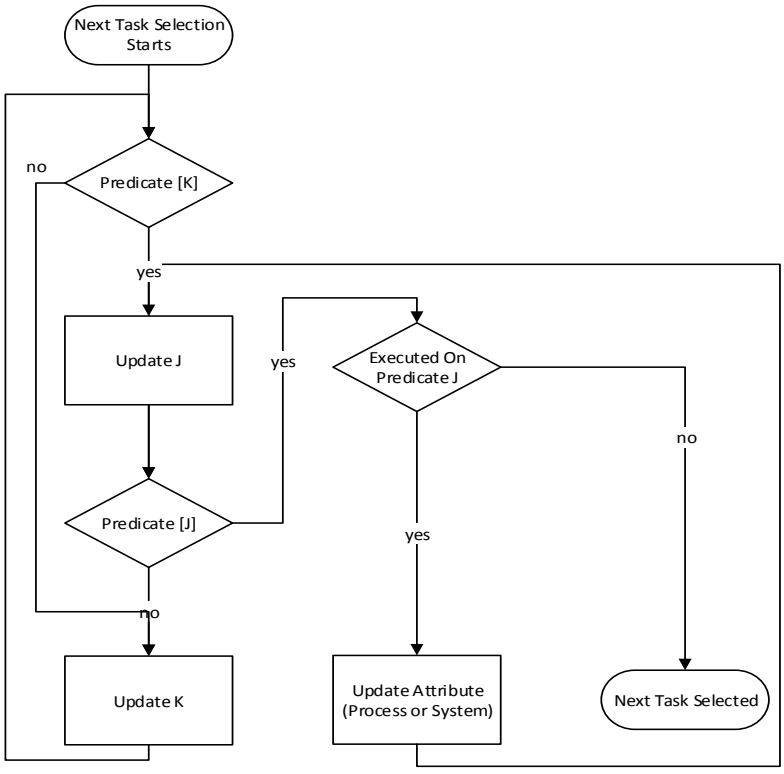


Figure 13. Task switching

Events types

Two event types are supported: task events, which are responsible for the execution of each task, and infrastructure events, responsible for messages related to the software infrastructure (e.g., events that send TCP packets). Once an event is scheduled to be executed, it is stored in the events queue. When the task event happens, the task associated with that event is executed according to the task logic. If the task execution is paused in case a message is sent out and reply for the message needs to be received, the task execution waits for the reply to be received or a timeout to be reached. Once the task execution is complete, task switching selects the next task to be executed.

Regarding infrastructure events, the path of a message is built at the time whenthe message is created. However, for each message that is travelling in the network, there is a need to update its location and store the location of the records associated with it. As a message is travelling from machine to machine, the location of the message is stored and the risks are updated based on the risks associated with each specific machine. For example, in a software development setting, a message containing critical source code can be intercepted by a malicious listener in a specific machine, and therefore the security risks has be updated. A message can be sent in two ways: in a non-blocking mode, where a component sends a message and continues its execution, or in a blocking mode, in which a component sends a message, and waits for its reply or a timeout.

Events queue

As it was previously explained, the FM is event-based. All scheduled events are stored in the events queue. The flow of behaviour that represents how events are scheduled and executed is shown in Figure 14. Each event placed in the events queue has a planned execution time. While adding events in the events queue, it is possible to schedule only events with a planned time greater than the current simulation time, which is an attribute of the events queue.

When an event is scheduled, this does not guarantee the event will be executed because the FM simulation can be interrupted or the event can be cancelled. For example, when the message is sent in a blocking mode, a component waits for the reply or a timeout. However, if a reply for the message is received by the component, the timeout event is cancelled.

The events queue stores the scheduled events. When the processing of all the events that have started to be processed is complete or paused, the events queue starts processing the next set of events. The events that have started to be processed are those scheduled for a time that is the closest to the current simulation time. There are two conditions that cause the events queue to stop the simulation: either the queue becomes empty, which means that all the component processes are finished and the simulation is complete or the attribute called terminated is set to true which means that the processing of all events that are currently being processed is finalized. However, no other scheduled events in the events queue are processed.

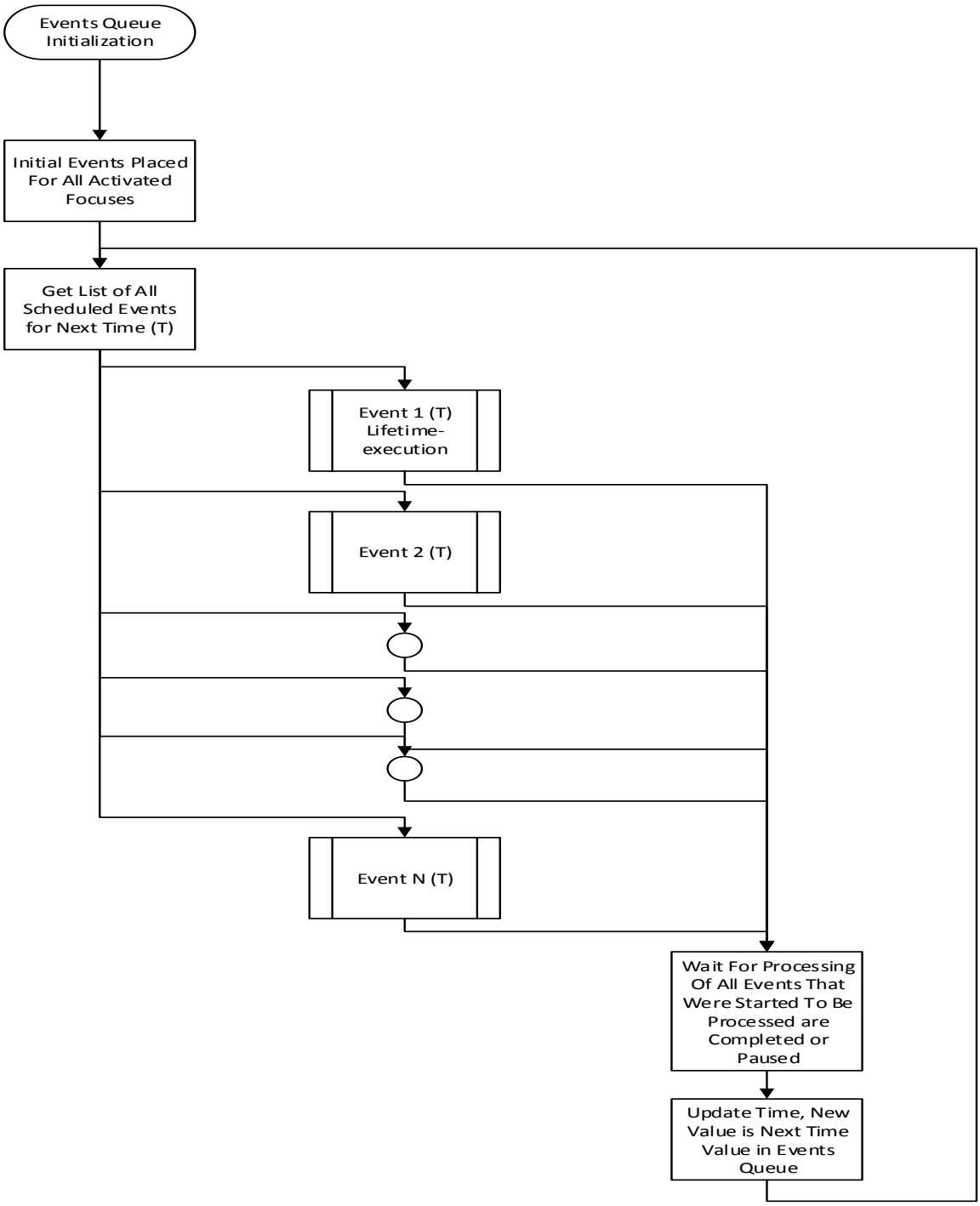


Figure 14. Events queue

Forecasting model behaviour

When initializing the system projection, the FM creates a focus for each component that has a focus defined for it. The activation of foci schedules a set of first events at time zero in the events queue. After this, the events queue is activated. Once activated, the events queue retrieves the set of all events scheduled for a time that is the closest to the current simulation time. All retrieved events are processed and removed from the events queue. During the processing of each event, first the task logic is invoked and then task switching is executed to determine which task should be executed next. While events are being processed, the events queue remains in an inactive state. Once all retrieved events are processed, the events queue updates the current simulation time to the time of the closest scheduled event it is holding. Then, the process repeats itself and this loop continues until the events queue is stopped by the policy decision point or the events queue has no events scheduled for execution. The event execution time is not always the time from the event being scheduled to the time it is being executed, because the event can go into a waiting state.

Figure 15 presents the behaviour of the FM. The behaviour starts with the instantiation of a scenario, and then proceeds with the creation of a focus, which points to tasks in a process graph. After that, events are processed until the last task of the focus is executed.

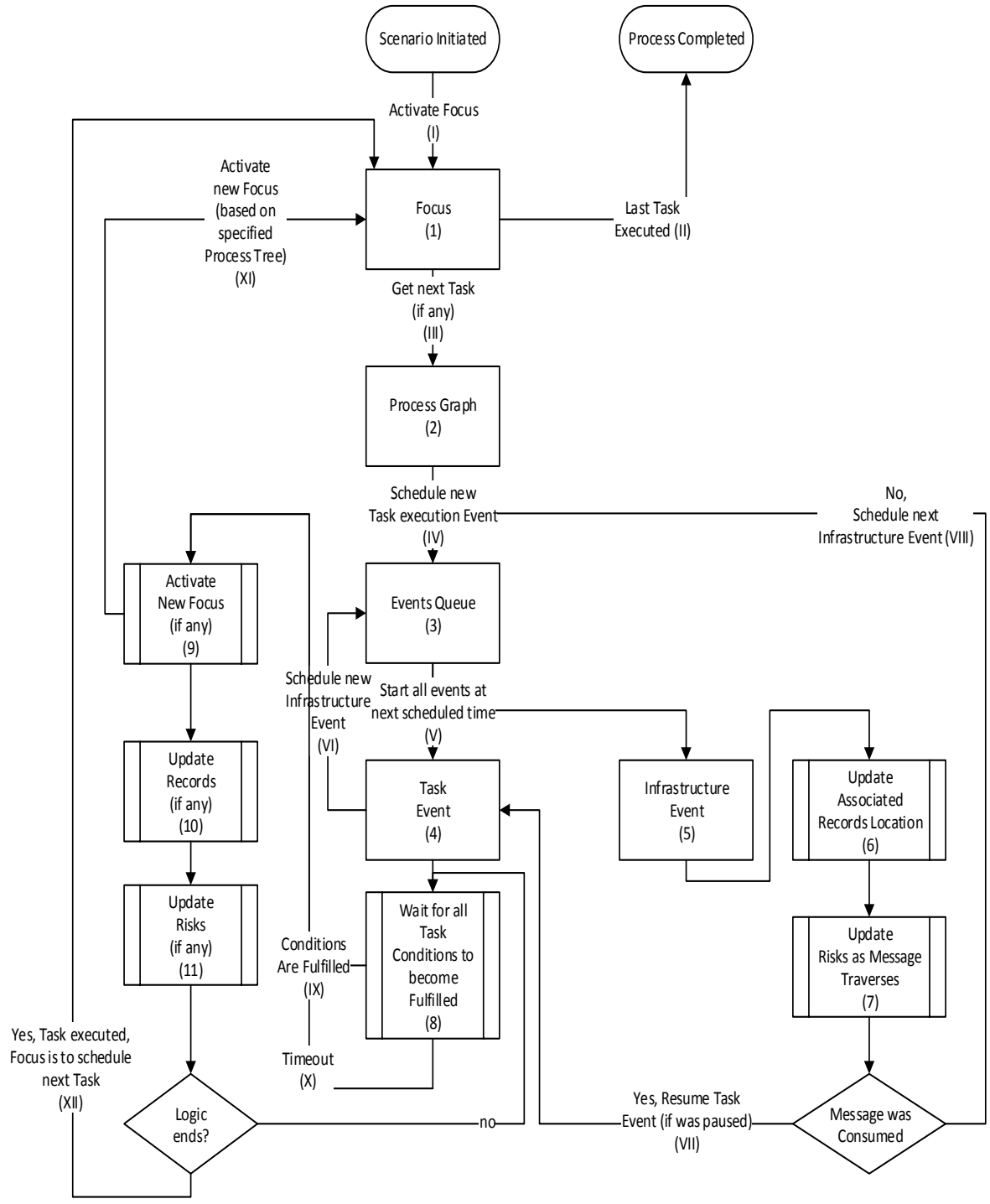


Figure 15. Behaviour of the forecasting model

3.1.4 Domain Models

Domain models (box 13 in Figure 7) are used to extend the core FM with domain-specific information. This information includes specifics about risks, events, components, processes and scenarios. For example, in a patient privacy setting, when a doctor is notified that the requested medical record has been printed or is displayed, the domain model describing medical aspects of handling a medical emergency increases the probability of a successful patient treatment, since the required information, for example, a list of the patient's allergies is available to the doctor in charge of the emergency procedure.

There can be various domain models representing different aspects of the functionality of a system for which AC is provided. In contrast with the existing approaches, there is no need to integrate different domain models into a whole system manually. Risk dimensions are defined in a modular way so that each of these dimensions encapsulates its own events and risks. It means that if a new risk dimension is defined, the new events and risks associated with it, will not interfere with the events and risks that belong to the other risk dimensions. As a result, the approach is extensible in the sense that new risk dimensions can be added to the system in a modular way.

3.1.5 Reserves

Reserves (box 14 in Figure 7) are associated with all interacting components and used to quantify the previously stored risks and benefits (QRMs) as resources that can be used to mitigate future risks. For example, in a software development setting if a software development company was completing projects on time and has accumulated some benefits as reserves, these reserves can be used in case risks need to be incurred. In this case, because of the accumulated reserves, for example, if a risk needs to be taken related to underestimating the project duration, the reserves can be used to pay for the impact value of this risk. However, if the reserves are not enough to pay for this impact, the risk associated with the underestimation of the project duration becomes critical because the reserves are not sufficient to pay for the impact. Therefore, the sum of the interacting components reserves compared to the impact of the risks can be used to assess which risks can be paid using these reserves, and therefore mitigated, and which risks cannot, and are therefore, critical.

The proposed approach uses an array of QRMs, which belong to each individual component to offer incentives for cooperation. Each QRM refers to a specific risk dimension and by design decision each QRM can be changed into QRM belonging to any other dimension according to the current level of exchange set for appropriate QRM types. QRM can also be set not to be exchangeable. Depending on the implementation of components' storage, each type of stored QRM also known as reserves can be anonymous, pseudonymous or strongly identifiable. Methods for operation with QRMs include simple or conditional transferring QRMs from one component to another component, storing QRM as reserves for future usage, checking for a sufficient level of reserves in a component, generating a new QRM as a reward for certain conditions being met, or destroying some reserves for the violation of some conditions.

3.1.6 Strategies

As it was mentioned previously, the output of the FM is a set of timelines of risk utility in a form of utility graphs and critical risks for each RMP and for each risk dimension. Consequently, there is a need to assess these timelines and choose the best applicable RMP candidates. The assessment of a risk timeline can be done, for example, by adding the risk utilities present in the timeline to produce a number that can be seen as a risk assessment measure. This is a function of the strategies (box 16 in Figure 7). Each strategy evaluates each timeline and selects the best applicable one. For example, a safe strategy is created to reduce the possible critical risks. This strategy will select RMP candidates with the lowest probability of critical risks. Another example of strategies is a greedy strategy that will try to get benefits by reducing the overall level of risk and maximizing the benefits. This strategy will choose RMP candidates with a higher probability of critical risks than in the case of a safe strategy if these RMP candidates offer less risks and more benefits. The use of this strategy helps to accumulate reserves that can be used to deal with these critical risks in the present and in the future. Thus, strategies are assessing the timelines of risk utility and critical risks and each strategy selects the best RMP candidate.

When constructing the RMPs there may be cases in which a task with a negative utility has to preceed a task with a positive utility with a greater magnitude. For example, it might be necessary to perform a task with a high risk before a task with an even higher benefit is available. In a patient privacy setting, it is possible that a large number of medical records must be made available to a doctor during an emergency without formal permission from the patients, but if the records are made available, a patient's life can be saved. In a software development setting, a number of costly source code elements must be made available to the developer before he or she can fix a critical bug.

Existing AC approaches often deny access as a default option, and do not take into account alternative actions that minimize the risks and maximize the benefits. For example, in the health domain, when a doctor is handling an emergency situation and time is of essence, refusing access to a patient medical record can result in the patient's death. By evaluating all possible RMP candidates (that is, the alternative actions), the strategies proposed in this approach can choose the best alternative course of action, which may tolerate high risk levels in order to deal with situations that are even riskier, namely life threatening.

Strategies can also be classified as general or domain-specific. General strategies can compare timelines of risk utility and critical risks independently of a domain. Domain-specific strategies compare timelines of risk utility and critical risks based on domain-specific information. For example, in the healthcare area, domain-specific strategies can be used to avoid loss of human life as described in the previous paragraph.

Behaviour of the improved design model

According to the Figure 16, the AC decision making process starts with a risk analysis. After performing the risk analysis, the list of all possible mutually exclusive tasks is created. Each possible mutually exclusive task is used to construct an RMP, producing an initial set of RMPs. These RMPs are then simulated to produce a set of risk timelines in a form of utility graphs. After this, each mutually non-exclusive task is incrementally added to each initial RMP to produce a new set of utility graphs. The new set of utility graphs is then compared to the previous set of utility graphs and, if a strategy produces a risk assessment measure (that is, a number) that is higher for a new utility graph than for a previous utility graph, the RMP associated with a new utility graph can replace the RMP associated with a previous utility

graph. After all mutually non-exclusive tasks have been evaluated and their utility graphs have been compared using their risk assessment measures, the final set of RMPs is presented to a component. Therefore, if a component represents a human system user that requires access to a specific resource, the AC is not restricted to provide a yes/no answer, as in the case of traditional approaches, but will present a set of possible alternatives in the form of sequences of tasks that the user can execute to obtain access to the resource.

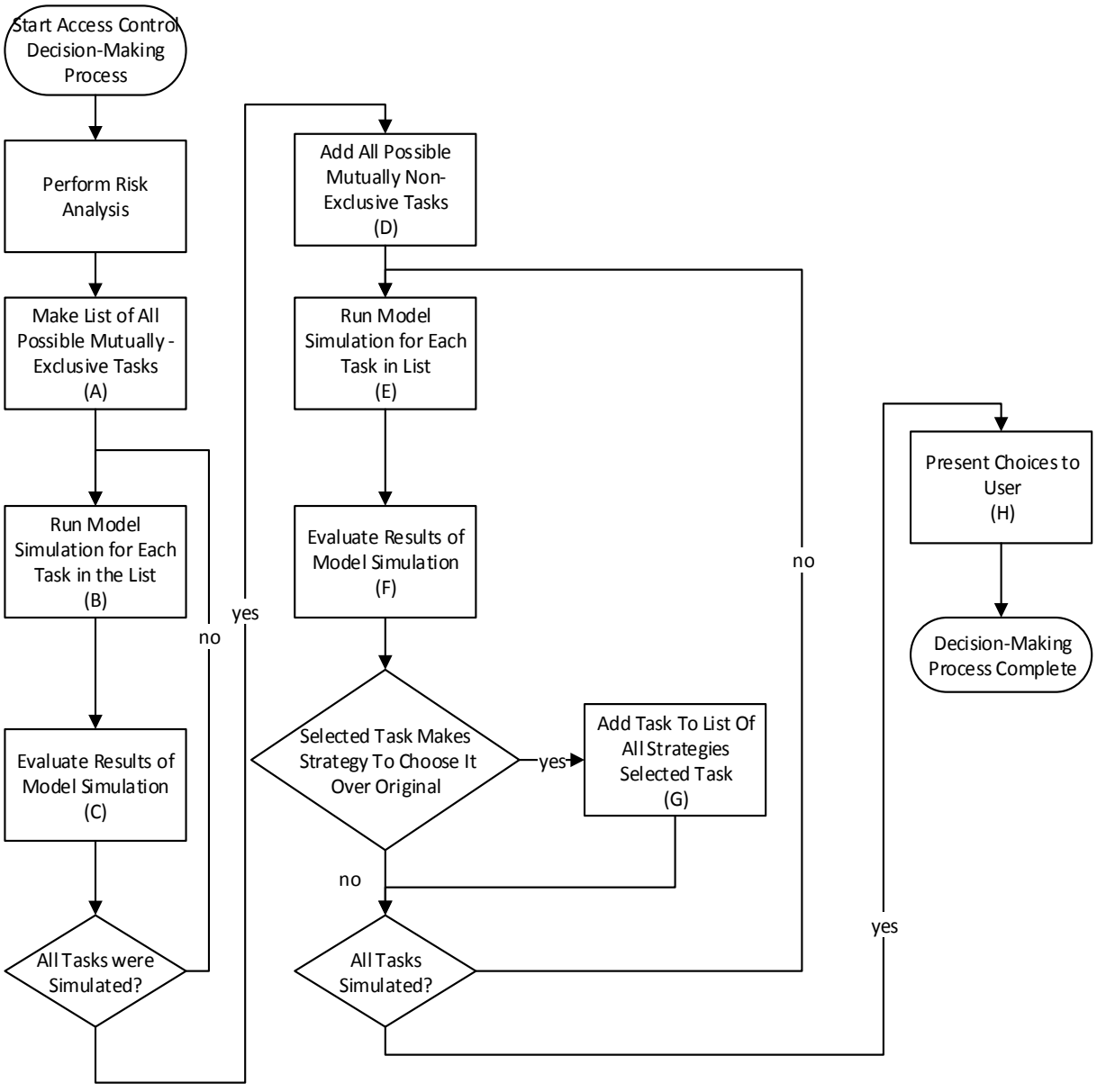


Figure 16. Behaviour of the improved design model

3.2 Details about the Application of the Approach

In this subsection, details about how the proposed approach can be applied are provided. First, a high-level overview is presented about how the system state is projected using simulation to produce the best RMPs. Second, an example illustrating how the approach can be applied is described.

A high-level overview that illustrates how the approach can be applied is presented in Figure 17. The proposed approach constructs a set of RMPs in which each RMP includes a set of risk mitigation tasks (M) and then the approach evaluates these RMP by projecting a state of a system for which AC is provided into the future for each of the constructed RMP through the series of intermediate system states System`1($T_0+\Delta T_1$) to System`1($T_0+\Delta T_1+...+\Delta Tn$). The operator ` denotes possible branches in future, given that certain events happen or given that AC decides to apply certain M. The approach relies on events to describe changes in the system, there are initial events in the scenario and there are events that are generated as system progresses its projection into the future. During this projection, the events lead to a realization or an occurrence of various risks and benefits belonging to the set of predefined risk dimensions. A realized risk or benefit is added to an appropriate risk timeline of the corresponding risk dimension. The end result of each simulation is a timeline of risks and benefits (also referred as timeline of risks or timeline for simplicity purposes). After the simulation is finished, the approach chooses the best risk mitigation task by comparing resulting timelines of risk utility in a form of utility graphs. Once all relevant RMPs are constructed, they are presented as a solution to a user or a component that requested access to a resource or a service.

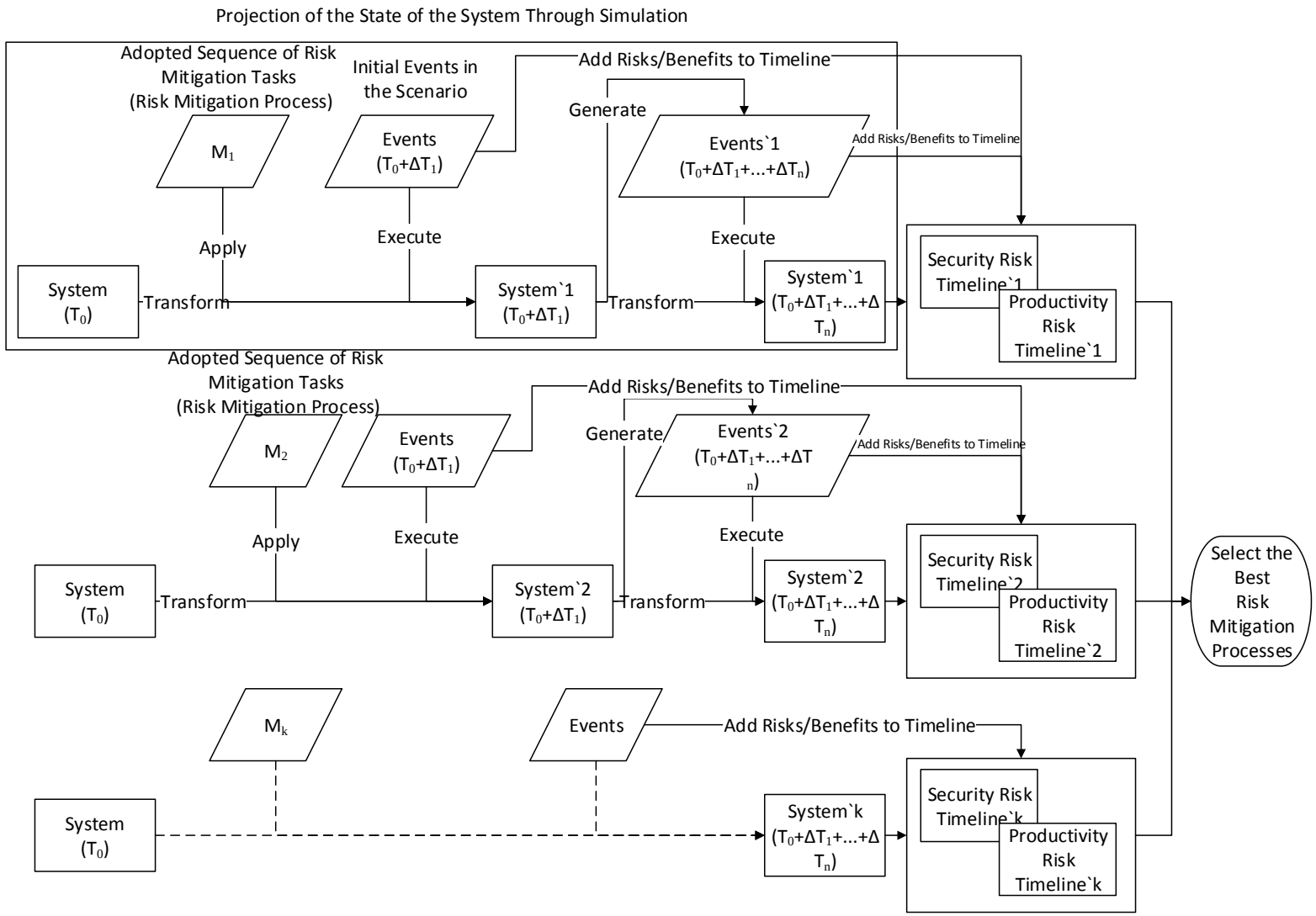


Figure 17. High level approach overview

3.2.1 Example of an Application of the Approach

Consider a software development application involving several types of risks, including security and productivity. Assume that the analysis and design stages of development are finished, and a software developer needs to write code to deliver a product. In addition, we can assume that the software development occurs in a network environment, since various code elements must be retrieved remotely. The task of the approach application is to provide a model that can automatically forecast the consequences of possible AC decisions, taking into account the risks and benefits and allowing the construction of risk-mitigation processes based on a set of obligations that were generated at runtime. Table 1 presents the data required by the FM of the approach.

Table 1. Forecasting model data

Asset Attribute	Documen-tation	Compiled Source Code	Obfuscated Source Code	Plain Source Code
Read	✔			✔
Compile Project		✔	✔	✔
Damage If Stolen($)	200K	600K	800K	2000K

The data presented in Table 1 in conjunction with the data from Figure 18 can be used according to the approach to relate attributes, assets, and risks, and transform all events that occur in the system into a combined utility describing how risky the process is. This utility value allows the approach to compare different RMPs and select the ones that are less risky than the others. Figure 19 describes how, through event generation, a system for which AC is provided evolves from time T_0 to some future time. Note that the system can be represented as a collection of components and processes, $S(T_0)$ is a state of S at time T_0, and $\Delta(S(T_3), S(T_2))$ is the difference between different states of the system at different times that can be represented by the event(s) that caused the system to change.

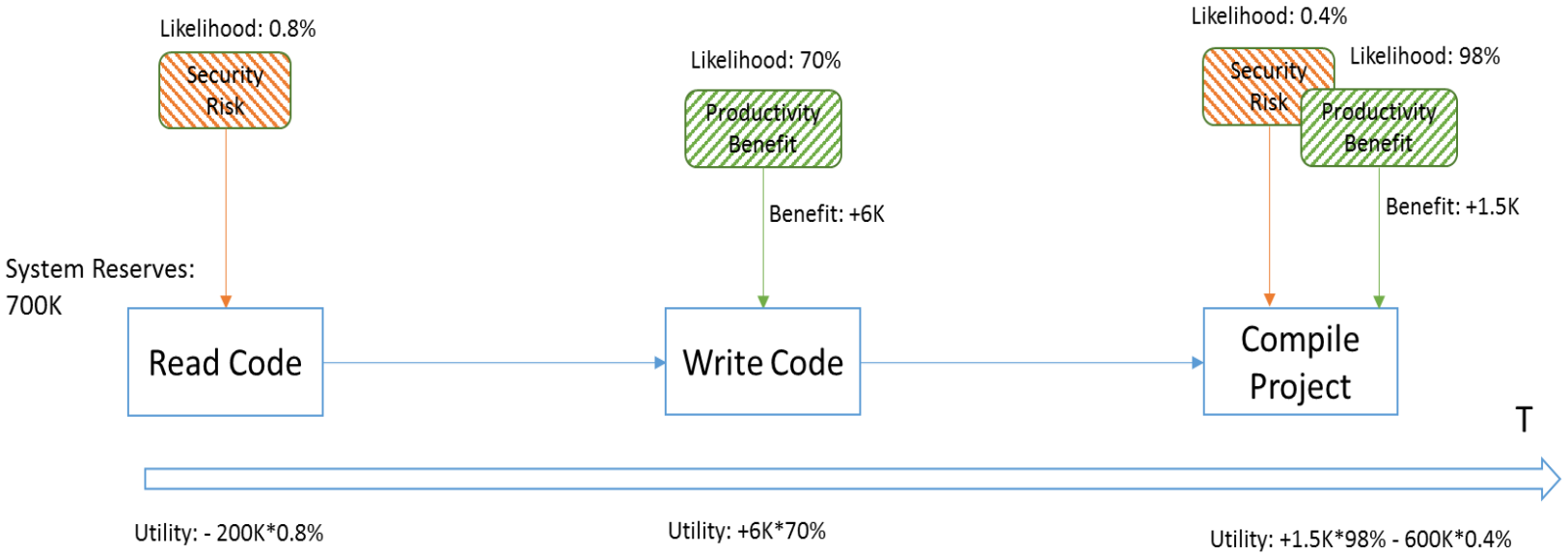


Figure 18. Deriving the utility from risks and benefits

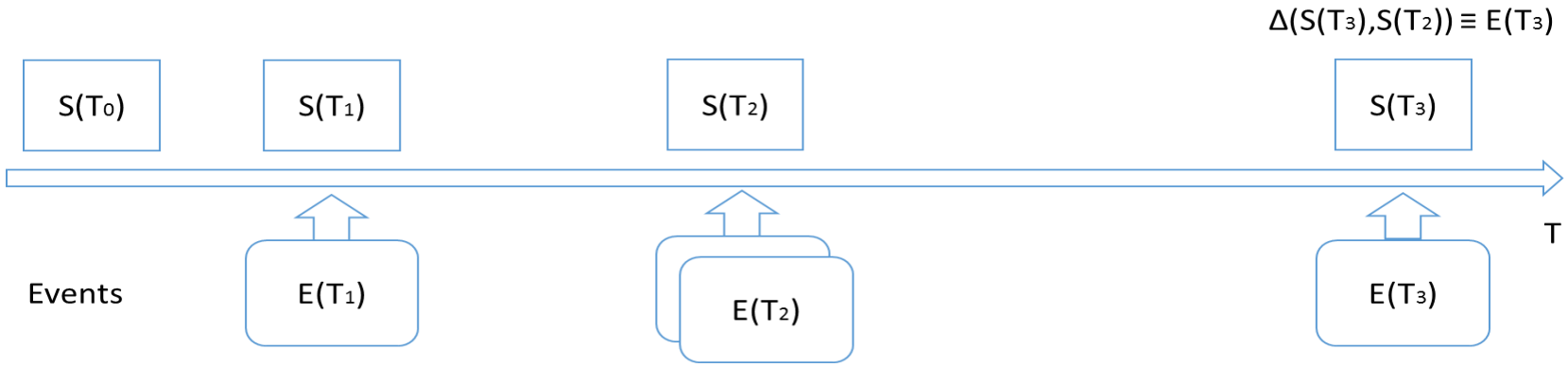


Figure 19. Forecasting through simulation

Figure 20 demonstrates that the timeline of the evolution of the system is not linear, since the system can evolve to different states when different events happen, and each of these events has a different probability. Therefore, it is possible to have different future states of the system at the same time, and these states are labelled by "`i" (i is an integer). Moreover, it is possible to say that certain states of the system (e.g., $S(T_2)$`3, $S(T_3)$`2) are more preferable than the others (e.g. $S(T_2)$`1, $S(T_2)$`2, $S(T_3)$`1). As these states are preferable, therefore, the events that lead them to these states are also desired. In the scope of the proposed approach these events are called risk mitigation tasks and they are introduced to lower risks while driving the system to the desired state.

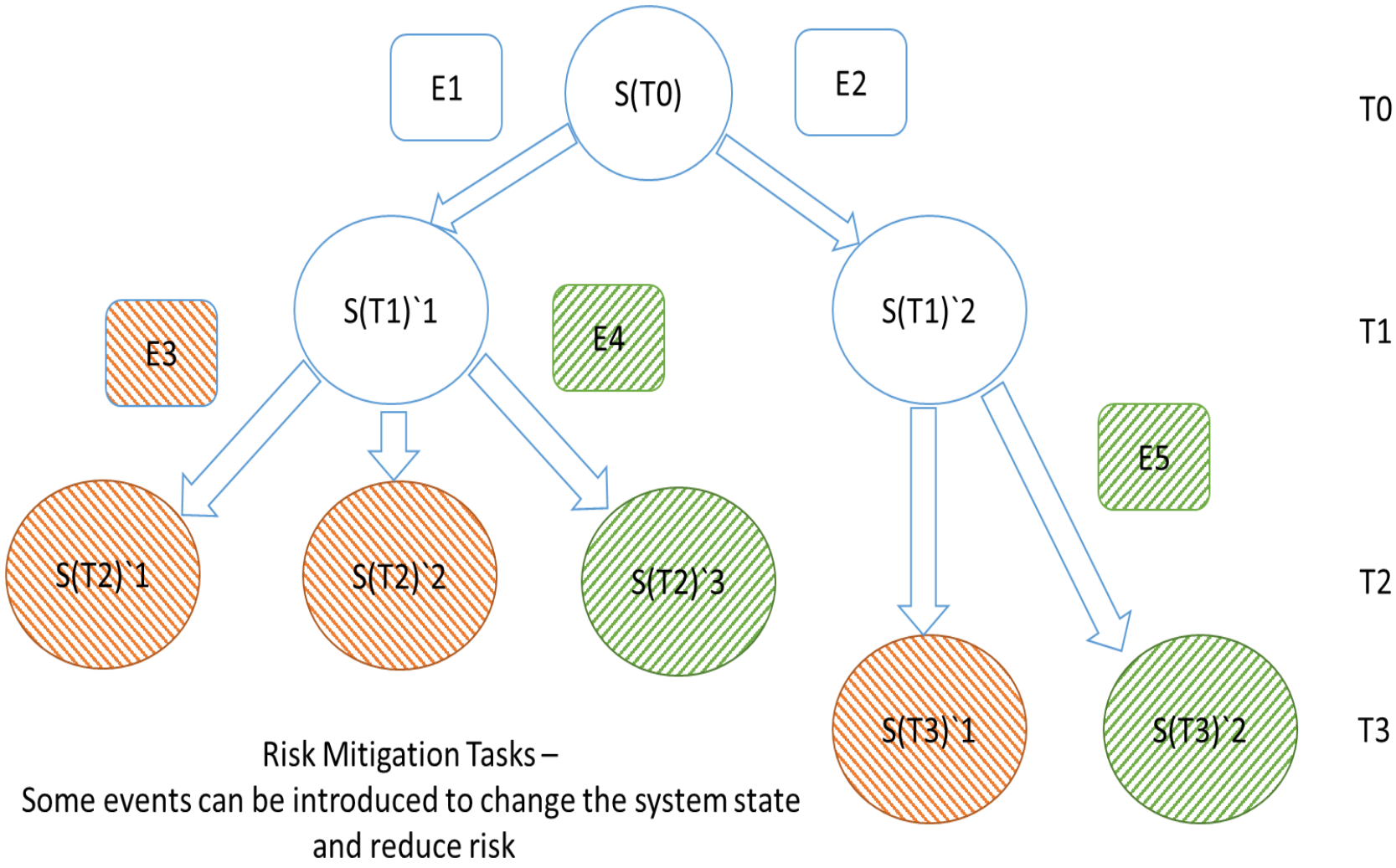


Figure 20. Risk mitigation tasks

Figure 21 represents the possible choices of results a user of the system. As the approach constructs different processes, the outcomes of these processes can be different forms of a resource granted to the user. In a software development scenario, when a developer requests the source code, it is not always known to AC what is the form of the source code the developer needs. For example, if a developer is currently compiling, then the source code might not be needed, but the compiled version of the source code will be sufficient. If a developer wants to make sure that the code is compilable or wants to change a single line of code, e.g., when fixing a common mistake with the help of debugging information, the developer might be satisfied with the obfuscated version of the source code. The risk of sharing the obfuscated version of the source code is typically higher than the risk of sharing the binary code version. If significant changes are required, the developer might need the complete source code and sharing this code leads to a higher risk than the risk of sharing an obfuscated version of code.

As the risks are different for the various forms of a requested resource, the proposed approach is capable of constructing risk-mitigation processes that grant access to each form of the resource. Therefore, when developers submit an AC request, they do not get an "yes" or "no" answer, but they rather get a selection of results and the processes to obtain these results. It is up to developer to select which form of the resource they need and he or she can select any of these process that matches their needs.

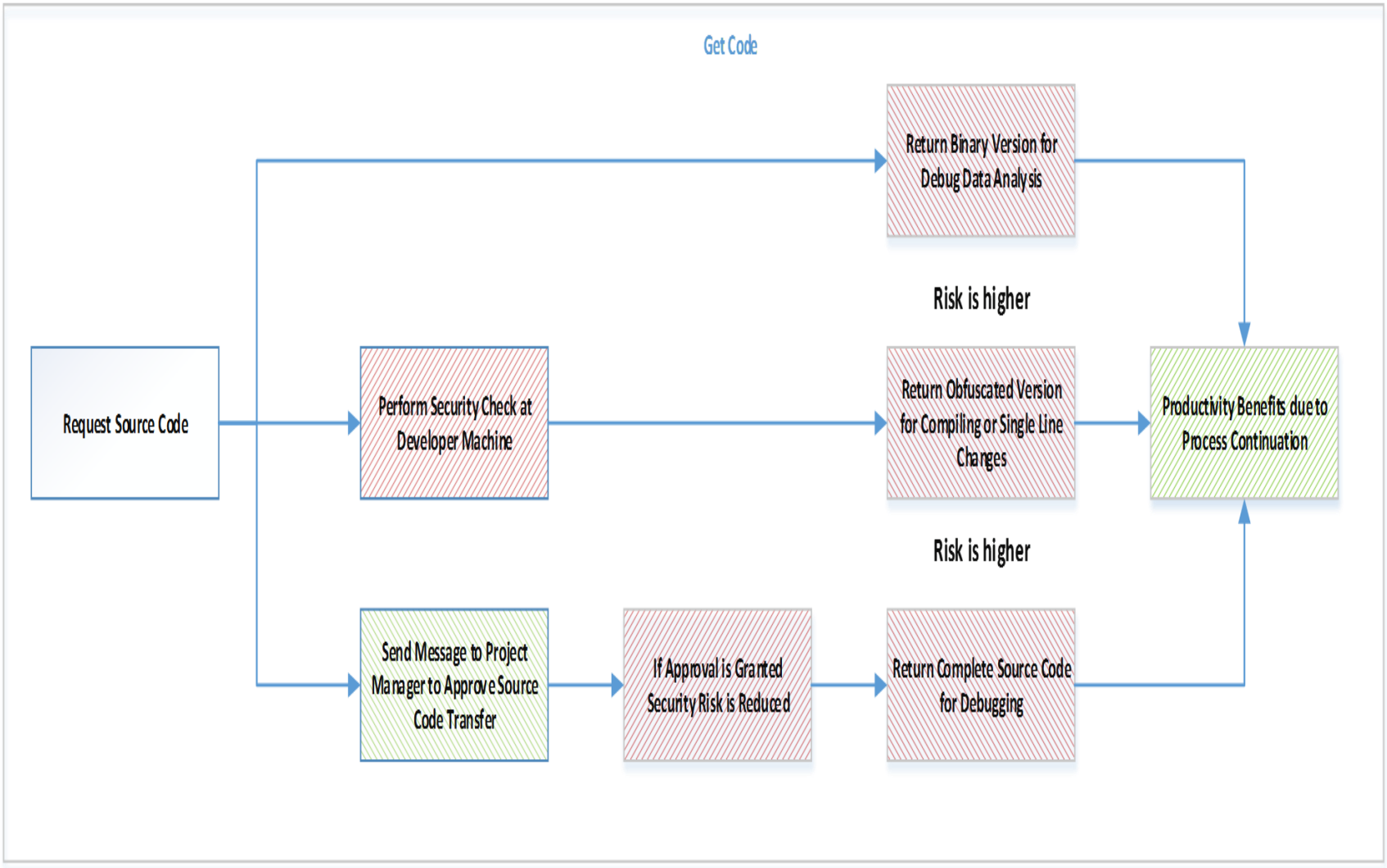


Figure 21. Possible RMPs and their results

3.2.2 Discussion

The approach was designed to work in a distributed decentralized environment in which traditional architectures are not applicable. For example, in the case of a distributed application when any number of users can join and leave as desired, it is not possible to enforce most of the centralized approaches decisions because a malicious user can report a different identity on each connection, posing as a new user or even as a large set of users. There is however, a need for orchestrated array of common tasks within such a system. Even though this situation brings lack of trust, however, any component (or users represented by these components) can choose to trust other specific components to perform mutually tasks they need. The proposed approach is designed to function in these circumstances, the approach assumes that any component can deviate from a set of constructed RMPs.

An obligation service of AC approaches can be used to extend the AC analysis beyond the simple allow or deny AC request resolution. However, the obligation service is neither capable of constructing a new RMP custom-tailored for a specific situation the system is currently in nor of evaluating the impact of the predefined RMP in detail. Further, RMP construction includes obligations represented in existing approaches as a part of the RMP functionality. In summary, the proposed approach is capable of providing access options in case any existing approach would deny access due to being unable to construct RMPs dynamically in response to resource or service request.

Note that it is possible in principle to translate existing traditional RBAC, ABAC, MAC or DAC policies into a set of domain models and components reserves such that access will be granted only if the traditional AC policies grant the access. However, if the traditional AC policies deny access, then the proposed approach might be able to construct RMPs to mitigate the risks to acceptable levels. In this sense, RMPs are used to provide an alternative that solves the problem when access is denied.

3.3 Formal Definitions

This section provides a formal definition of the main concepts used in this thesis, explains their role in the approach, and presents examples of their application. The following

collection includes definitions of attribute, resource, component, task, process graph, process, event, projection, risk and risk dimensions among others.

Before providing the formal definitions, a high-level description of some of these concepts is presented. Attributes are associated with specific objects and commonly used for quantitative risk analysis. An example of an attribute is the number of medical records a doctor has requested. Resources are physical and logical objects that are needed to complete a task. An example of a resource is the cash dispensed to customers in an ATM. A process graph is a model of the business processes represented in the form of a directed graph with tasks used as its nodes. Examples of process graphs are the process graphs specific to the health and software development domains that are provided in chapter 4. A process is a representation of the actual business process used in forecasting. Typically, many processes are created before an AC decision is made and the actual process starts. A process includes a reference to a process graph, the executing component(s) and the process context, which can be, for example, the process duration.

Further, events are used to represent occurrences in the FM, which processes the scheduled events to create new events and is used to project the modelled system into the future. A projection is the operation of taking an entity at specific point of time and transforming by simulating the changes it goes through over time. For example, while a doctor treats a patient in an emergency department it is possible to project that with a certain probability that in two hours a new patient will come to this department in need of some form of medical treatment. Risks have a probability associated with them. For example, when doctor is treating an emergency, there is a risk and an associated probability that treatment will be not effective and the patient might not survive. A risk dimension is a set of risks that can be grouped together in a specific application domain. For example, in a software development setting possible risk dimensions are security and productivity.

The detailed formal definitions of the main concepts used in this thesis are provided next.

Definition 1: Attribute $Atr \in A$; A is a set of attributes; $Atr = \{V, N\}$; V is the value of the attribute and N is the name of the attribute.

An attribute defined as a risk-related quantified parameter associated with any entity described in the approach. Changes are generally specified as attribute(s) change(s) or change(s) of sets of attributes. Sets are changed by adding or removing its members. An example of an attribute is the time to be updated, which can be associated with a specific document.

Definition 2: Record Rec ∈ RC; RC is a set of records; Rec = $\{A_{rec}, N\}$; $A_{rec} \subset A$ is a subset of attributes associated with Rec, and N is the name of the record.

A record is the risk-related attribute that can be copied to and moved from one component to other components. Each record can exist only as one instance, but can be referenced by many objects. For example, in a patient privacy setting a printed document and entry in a database are objects that can reference a medical record of a patient.

Definition 3: Resource Res ∈ R; R is a set of resources; Res = $\{A_r, F_r(A_i, A_e)\}$; $A_r \subset A$ is a subset of attributes associated with Res; F_r is a set of functions F associated with Res, each F has a predefined set of attributes as arguments, where $A_i \subset Ar$ is a subset of attributes associated with the resource (internal to resource), $A_e \subset A$ is a set of attributes (external to resource), where $A_e \cap A_r = \emptyset$.

A resource is something required to perform a business process, and its types vary depending on the application domain. For example, in the health domain, a medical record can be a resource necessary for doctor to perform a surgery. The return value F is not restricted and can have side effects that result in changes to other objects states. For example, in a patient privacy setting, a printer can be seen as a resource. However, if the printer is not available or responsive another resource can be utilized, that is, the doctor can either print a requested document using another printer or look through the document on a computer screen.

Definition 4: Component Com ∈ C; C is a set of components; Com = $\{R_c, PG_c, PGN_c\}$; $R_c \subset R$ is a subset of resources associated with the Com. A component can use these resources without the need of an explicit permission from other components; $PG_c \in PG$, is a process graph that is currently being executed by Com, where PG is a set of process graphs;

$PGN_c \in PGN$, is a current process graph node executed by Com, where PGN is a set of process graph nodes.

A component is an entity executing a process. An example in the health domain of a component is a doctor treating a patient. The pair of {PGc, PGNc} is contained in the focus element of the improved design model. An example of the component in a patient privacy example is a database holding all medical records. In a software development application, an example of a component is a firewall: the component designed to analyze the network traffic and allow packets to pass through according to specific firewall policies.

Definition 5: Task $Ta \in T$;T is a set of Tasks; $Ta = \{A_t, C_t, R_t, \{CN_{ti}(A_i, A_e)\text{->Bool}, F_{rti}(A_{rtii}, A_{rtie})\}_t\}$; $A_t \subset A$ is a subset of attributes associated with Ta; $C_t \subset C$ is a set of components executing Ta; $R_t \subset R$ is a set of resources needed to execute Ta; $\{CN_{ti}(A_i, A_e)\text{->Bool}, R_{ti}.F()\}_t$ is a set of pairs of functions associated with this task; CN_t is a set of condition functions associated with this task; $CN_{ti} \in CN_t$ is a condition function that takes a predefined set of attributes as arguments and produces a boolean value, where $A_i \subset A_t$ is a subset of attributes associated with the task (internal to the task), $A_e \subset A$ is a set of attributes (external to the Task), where $A_e \cap A_t = \emptyset$; $F_{rti} \in F$ associated with resource R_{ti}; $R_{ti} \in R_t$.

A task is a set of simple actions with context. Tasks are designed to execute the self-contained logic of the components; they also contain a routine to choose the next task when the current task is completed. When the task is completed, there is a mandatory set of attributes known as the result of the task that is used to select the next task included in process description to be executed.

Definition 6: process graph $PGr \in PG$; PG is a set of process graphs; $PGr = PGN_i$, where PGN_i is a set of process graph nodes associated with the process graph.

A process graph is a graph of tasks used to model process execution. In a patient privacy setting an example of process graph can be represented when a doctor tries to save a patient's life during an emergency and performs sequence of tasks that can be described as process graph nodes.

Definition 7: The process graph node $PGNo \in PGN$; PGN is a set of process graph nodes; $PGNo = \{A_{ptn}, \{CN_{ptnoi}(A_i,A_e)\text{->Bool}, T_{ptnoi}\}_{ptno}\}$; $A_{ptn} \subset A$ is a subset of attributes

associated with PGNo; $\{CN_{ptnoi}(A_i,A_e)\text{->Bool}, T_{ptnoi}\}_{ptno}$ is a set of pairs of functions associated with this process graph node; CN_{ptno} is also a set of condition functions associated with this process graph node; $CN_{ptnoi} \in CN_{ptno}$ is a condition function that takes a predefined set of attributes as arguments and produces a boolean value, where $A_i \subset A_{ptn}$ is a subset of attributes associated with the process graph node (internal to process graph node), $A_e \subset A$ is a set of attributes (external to process graph node), where $A_e \cap A_{ptn}=\emptyset$.; Tptno is a set of tasks associated with the process graph; $T_{ptnoi} \in T_{ptno}$.

Examples of process graph nodes are the nodes included in a graph consisting of three process graph nodes, A, B, C, where A is the starting task and B is executed if the execution result of A is a success, while C is executed if such execution is a failure. Then, the condition for A is that the task A was not executed previously; the condition for B is that task B was not executed previously and the last executed task executed was A, and the last task execution was a success; while the condition for C is that the task C was not executed previously and the last task execution was a failure. In summary, a task T can be executed when Cn is evaluated to be true. When a task is initially defined as a template, actual tasks can be instantiated using this template and executed.

Definition 8: Process $Pr \in P$; P is a set of processes; $Pr = \{A_p, C_p, \{CN_p(A_i, A_e)\text{->Bool}, T_{pi}\}_p\}$; $A_p \subset A$ is a subset of attributes associated with Pr; $C_p \subset C$ is a subset of components executing Pr; $\{CN_{pi}(A_i,A_e)\text{->Bool}, T_{pi}\}_p$ is a set of pairs of functions associated with this process; CN_p is a set of condition functions also associated with this process; $CN_{pi} \in CN_p$; CN_{pi} is a condition function that takes a predefined set of attributes as arguments and produces a Boolean value, where $A_i \subset A_p$ is a subset of attributes associated with the process (internal to the process), $A_e \subset A$ is a set of attributes (external to the process), and where $A_e \cap A_p=\emptyset$.; T_p is a set of tasks associated with this process; $T_{pi} \in T_p$; by definition, $Pr \equiv \{PT_r, C_p\}$.

A process is a set of tasks that have to resolve a problem in a specific situation. Each process has its own process graph, which is a graph of interconnected tasks. The components that execute a process perform tasks from the process graph and navigate the process graph by choosing the task to be executed after they finish a selected task.

Definition 9: System $S = \{C_s, R_s, P_s, O_s\}$; $C_s \subset C$ is a set of components that are part of the system; $R_s \subset R$ is a set of resources that are part of the system (i.e., they belong to the system components and/or are controlled by the AC system); $P_s \subset P$ is a set of currently ongoing processes in a system executed by system components; and O_s is a set of outside (not internal) components, tasks, resources, attributes and processes, $O_s \cap C_s = \emptyset$, $O_s \cap R_s = \emptyset$, $O_s \cap P_s = \emptyset$.

The system is a collection of components, processes and resources work together to perform one or more functions.

Definition 10: Event $Ev \in E$; E is a set of events; Ev denotes an event that happens at T_1, $S(t) = S(t_0)$ when $t \in [t_0, t1)$, $S(t_1) \neq S(t_0)$.

An event is something that occurs at a specific time. In the software development setting, the completion of a task named ProblemSpecification can be seen as an event that occurs at a time equal to 3140 seconds. Before the 3140 second mark, there has to be an interval in which nothing changes the system state. At the same time that this event happens, the system state would be changed by the impact of this event (and possibly also by the impact of other events if they happen at the same time). In a patient privacy setting, for example, the deadline for shredding a document can be 120 minutes after the doctor has requested the document for printing. Assuming an event is this deadline, it may imply that if in 120 minutes this document is not yet shredded a doctor and later a privacy analyst would be informed that the deadline for shredding the printed document has expired.

Definition 11: Operator $\Delta(A, B)$ is a change that can be applied to object A to get object B. This change involves altering the attribute values and adjusting sets by removing and adding their elements.

In general, this operator is used to capture what changes need to be made to transform an object A into an object B. In the case of a software development application:

Δ([{“C:\”,”driveName”}, {6000,”driveSpeed”}], [{“C:\”,”driveName”}, {5000,”driveSpeed”}]) = [1000,”driveSpeed”] assuming that the drive speed is an additive attribute.

In the case of a patient privacy application:

Δ([{“120”,”timeToShred”},{1105,”documentID”}],[{“1105”,”documentID”},{130,” timeToShred”}]) = [10,” timeToShred”], assuming that timeToShred is an additive attribute.

Definition 12: Operator -: A - B ≡ Δ (A, B).

Definition 13: Operator +: A + B ≡ A - (∅ - B).

Definition 14: Operator Diff (A, B) =Diff (Δ (A, B), ∅) is an analyst defined function (difference, distance) of similarity.

If applied to strings, the operator Diff driven by the analyst-defined function eliminates the values that are equal, subtracts similar values, and returns the distance between two strings. In a software development setting:

[{“C:\”,”driveName”}, {6000,”driveSpeed”}] - [{“C:\”,”driveName”} , {5000,”driveSpeed”}] = {6000,”driveSpeed”} - {5000,”driveSpeed”}] = {1000,”driveSpeed”} = 1000, given driveSpeed value is 1.

[{“120”,”timeToShred”}, {1105,”documentID”}] - [{“130”,”timeToShred”}, {1105,”documentID”}] = {120,”timeToShred”} - {“130”, ”timeToShred”}] = {10,”timeToShred”} = 600 seconds, given timeToShred is measured in minutes and calculated value is in seconds.

Definition 15: Operator x for objects $S(T_0)xE(T_1)=S(T_0)+(\lim_{T\to T1}(S(T))-S(T_0))$, defined for the system, $E_1(T_1)xE_2(T_1)$: $(S(T_0)xE_1(T_1))xE_2(T_1) \equiv (S(T_0)xE_2(T_1))xE_1(T1) \equiv S(T_0)x(E_1(T_1)xE_2(T_1))$, SxT_a: $T_a \in T$; assume $T_a = \{E_{ta1}(T_{eta1})xE_{ta2}(T_{eta2})x\ldots x\ E_{tak}(T_{etak})\}$; $E_{tai} \in E$; $T_{etai} \in T$; $E_{tai} = (CN_{ti}(A_i, A_e)\text{->Bool}) \wedge (F_{rti}(A_{rtii}, A_{rtie}))$.

Operator x for events - $E_1(T_1)xE_2(T_1) \equiv E_3(T_1)$: $(S(T_0)xE_1(T_1))xE_2(T_1)= S(T_0)xE_3(T_1)$.

The operator x changes the state of an object, it is applied to by executing an event (E (T_1)) at a specified time. The operator x applied to multiple events combines the application of the operator to each of the events.

Definition 16: Operator Ω: $S(T_1)\ \Omega\ E_x \equiv S(T_1) : (S(T_1) = S(T_0)x[E_i(T(E_i))] \wedge (\exists i{:}E_i=E_x), T_1>T(E_i)>T_0)$.

The operator Ω indicates that the system state at time T_1 is such that the event E has happened some time previously. For example, if we observed that at time 120 minutes a specific medical record was printed 10 times and at time 130 minutes it was printed 11 times, it means that between time 120 and 130 minutes there was an event that made this record to be printed.

Definition 17: Operator A (T) ≡ state of object A in time T.

With respect to operator A(T), S(T) = {C_s(T), R_s(T), P_s(T), O_s(T)}, while C, R and P are part of the system and can be monitored, O only refers to interactions/observations/projections regarding outside elements, and therefore, assumptions about it can be inaccurate. In a software development setting, an attribute representing a hard drive can be ["C:\","driveName"] at time 2500 seconds and, after manipulations, it can become ["D:\","driveName"] at time 2600 seconds. In a patient privacy setting, when an attribute {"120","timeToShred"} becomes {"130","timeToShred"} at time 7200 seconds it means that a deadline for shredding was extended by ten minutes and it is not yet time to resolve the problem by notifying a privacy analyst.

Definition 18: Operator `: $A(T_1)` \equiv \{A(T_1)`_1|\ldots|A(T_1)`_n\}$, where $A(T_1)`_i$ is a possible state number i of A at time T_1 and A has a total number n of possible states at time T_1.

The operator ` refers to possible outcomes. From $T_0$ to $T_1$ some events can happen and the ` Operator denotes possible object states at time T_1. Assume there are *n* possible event sets. These events will result in *n* different states of the object $A(T_1)` = \{A(T_1)`_1|\ldots|A(T_1)`_n\}$, where $A(T_1)`_i$ is the i-th possible state of A at time T_1 and A has a total number n of possible states at time T_1. For example, in a patient privacy setting, depending on the outcomes of the medical procedures as well as on the provided information, an emergency can be resolved or can persist. In a software development setting, considering a previously listed piece of code, depending on the results of a Random.getNext call, the variable partOneProblemDone can be set to true or not. Figure 22 presents a set of possible states of an object A at times T_0, $T_{0.3}$, $T_{0.4}$, $T_{0.5}$ and at T_1. It also shows the events that are leading to these states. Based on this diagram, at T_1, *n* equals 6 as there are six possible states at T_1.

Definition 19: Projection Operator $Pr(A,T_1,T_2) \equiv Pr(A(T_1),T_1,T_2) \equiv [A(T_2)`_i, L(A(T_2)`_i)]$ where i=1..K and K is a number of projected states of A at time T_2, $L(A(T_2)`_i)$

is a projected likelihood of $A(T_2)= A(T_2)`_i$, with $0 \leq$ likelihood $\leq 1$ and $\sum_K^{i=1} L(A(T_2)`_i) = 1$. T_1 is a time when state of object A is known and T_2 is a time for which the operator projects the state of the object A to be.

Projections are built to forecast future events and produce outcomes which are set of states and a probability associated with each of them. However, this set of states can be incomplete and their probabilities may not be accurate. This is caused by invalid projection assumptions or missing data. For example, if a coin in a coin toss is biased and has 0.2 probability of landing on its head and 0.8 probability of landing on its tail, assuming an analyst was not aware of the bias, the analysis would predict equaly probable results. However, the real outcome would likely be different, and the mean of the difference between the forecasted values and the actual values should be, for example, (0.8-0.5)*n, where n is a number of coin tosses. Figure 22 depicts the progress of the system states as different events happen.

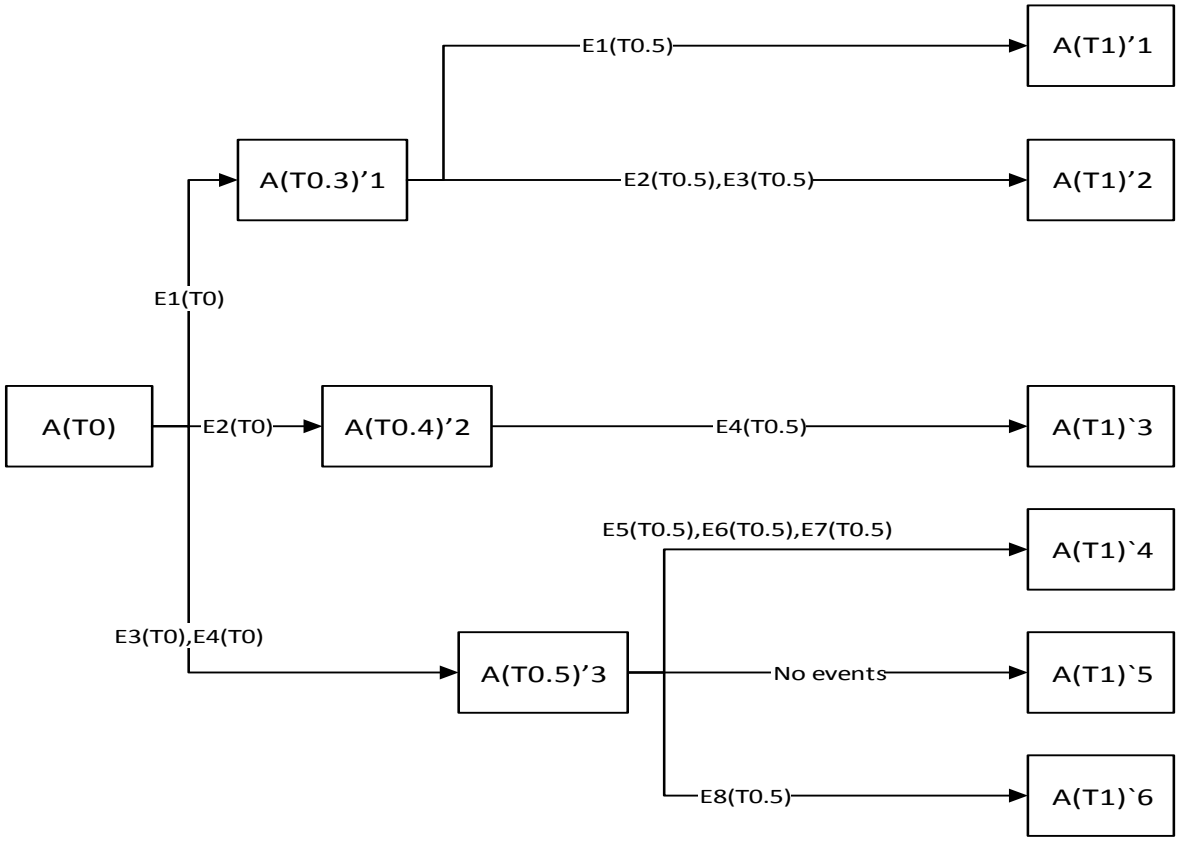


Figure 22.Object A: possible states and preceding events

Definition 20: Inaccuracy of Projection is:

$\sum$ (Diff (E (T_2), E (T_2)`$_i$))*(1- Probability (E($T_2$))`$_i$).

The inaccuracy of a projection is the weighted difference between the projected state of the object at a certain time and the actual observed state. The actual observed state has probability of 1, hence the difference is 1- Probability (ET_2)`$_i$. The weights include the projected probabilities. For example, in a patient privacy setting, assume that at time T=0 a document was printed with a privacy risk impact of 1.1 and the projection for T=120 was calculated at T=110 (the time in which the projection is calculated) as Pr({"documentShredded"}, 110, 120) = [{{"documentShredded", true}, 70%}, {{"documentShredded", false}, 30%}] and given that at time T=120 a document has not yet been shredded, the inaccuracy of the projection is 0.7*1.1, where 0.7 is the accuracy of prediction and 1.1 is a risk impact of the document being unauthorizedly disseminated.

Definition 21: Events queue is {$T_{current}$, E_i (T_i)}; $E_i \subset E$; $T_i > T_{current}$.

The proposed approach includes an events queue responsible for keeping all scheduled events and executing them on time. For example, in case of a patient privacy scenario, the contents of the events queue can be [{Notify Privacy Analyst: {" timeToShred" ≥ "currentTime"},720 s},{Extend Deadline: {"timeToShred" = "currentTime"+100},700 s}], where the deadline for shredding a document is extended. in case of a software development setting, the contents of the events queue can be [{Update File Size Event: {"fileSize" = "fileSize"+10kb},12 ms}, {Check File Size: { if "fileSize" >100kb create new event (Transfer Data) },15 ms}], which means that when time reaches 12 milliseconds, the attribute file size is increased by 10kb and when time reaches 15 milliseconds, a component checks the file size attribute and, if it exceeds 100kb, the component creates a new event to transfer the excess data.

Definition 22: Simulation(A,T_1,T_n) ≡ Projection(Projection (… Projection(Projection(A, T_1, T_2), T_2, T_3),…),T_{n-1},T_n), where simulation steps are [$\Delta(T_i,T_{i-1})$] and $T_1 \ldots T_n$ are the times when events occur. The simulation takes three parameters, A, which is an object to be projected into the future, T_1 is a start of the simulation and T_n is time when simulation ends. N is not defined and depending on how simulation unfolds N can vary.

The simulation is a consecutive set of projections.

Definition 23: Risk Evaluation RE (T) ≡ [F(S(T)) -> numeric], where F is an evaluation function that assesses the state of the system for which AC is provided.

In a patient privacy setting, at a time equal to 120 minutes, the simulation could project that the following risks are present: [{"Unauthorized Dissemination Risk", 5}, {"Poor Quality Treatment Risk", 0.1}] and given current ratio of these risks is 30:1 then RE(120 min)=5/30+0.1=0.266. In a software development setting, for a time equal to 20 minutes, the simulation projects that there could be two risks present, [{"Security Risk", 10}, {"Productivity Risk", -2}], and assuming that the current ratio of risks, security to productivity, is 0.5, then RE (20 min) =10*0.5+ (-2) =3.

Definition 24: Risk Impact of event Rim(E_r, T) ≡ Diff (RE(S (T)), RE(S (T) × E_r (T))), defined by Risk event E_r and possible time of occurrence T.

Risk Impact is a change in risk attributes, which an event creates in a system. In a patient privacy setting, for example, when a request for printing a patient file is granted, the risk impact of this event would be 1 in units of an Unauthorized Dissemination risk dimension. In a software development example, when a component is reading a sensitive file, the security risk increases, and thus, the risk impact of the event is positive. If, during an event, the risk attributes do not change, the risk impact is equal to zero.

Definition 25: Risk Likelihood of event Rli(E_r, Pr(S, T_1, T_2)) ≡ $\sum_{K}^{i=1} L\left(S(T_2)\right)`_i : S(T_2) \Omega E_r$. Risk likelihood of an event associated with a risk is the likelihood of this event happening. In a software development setting:

The projection at time 1 hour:

[{Task Solved Event`$_1$: {"productivity risk" = "productivity risk"-2}, 30%, 1 hour},

{Task Not Solved Event`$_2$: { create new event (Read More) }, 70%, 1 hour}]

Then the risk likelihood of a developer not being able to solve the task and having to read additional documentation is 0.7.

Definition 26: Risk Rs ∈ R; R is a set of risks; Rs(Er, Pr(S, T_1, T_2)) $= \int_{T2}^{T=T1} Rli(E_r, \Pr(S, T, T_2)) * Rim(E_r, T)$. An integral is used here, since number of steps

the simulation takes to get from time T_1 to time T_2 is not predefined and can vary based on the probabilistic model.

Each risk is associated with a specific risk dimension. The total productivity risk for the projections listed in the previous definition in case of a software development setting, for example, is: 0.3*(-2*1+0.2*(-1)) +0.7*(0.5*0.3+0.2*0.2+0.2*0.5) =0.093.

Definition 27: Risk Dimension RD={R_{RD}, RE_{RD}}; $R_{RD} \subset R$, where R is a set of risks, R_{RD} is a subset of risks associated with this RD and RE_{RD} is a risk evaluation function designed to capture all events and attributes related to this specific RD.

Risk dimension is a group of risks related to a specific aspect of the system operation in a specific application domain. In the software development domain, examples of risk domains are security and productivity. In a patient privacy domain, possible risk dimensions can be patient privacy and the quality of the patient treatment. The risk evaluation function associated with RD can either be formal and capture the utility related to RD or be an informal description of what risks are associated with this domain.

Definition 28: Risk Conversion RC(i, j, (S(T))=F(i, j, RD_i, RD_j, S(T), T) : RE(T) increased or not lowered by conversion (trade by two components of one RD QRM to another).

Risk conversion is a function than converts the value of a specific risk type to a fair value of another risk type. "Fair" in this description means that the conversion does not deteriorate the system state, which is measured using risk levels.

Definition 29: Risk Mitigation Method M(sEr,R(Er)) =Ta: ∃i: Diff(RE(SxTa), $RE(S))_i$>0.

A risk mitigation method is a way of introducing specific events that reduce risks. In a patient privacy setting, AC can refuse a requested access or can ask other medical personnel participating in emergency to verify the request. In a software development setting, a firewall is able to prevent unauthorized information dissemination. However, to properly configure a firewall, firewall policies must be altered. Thus, an event that properly alters the firewall policies reduces the risks of unauthorized information dissemination. The improved design

model is event-based, and the events represent an asynchronous execution of routines associated with tasks and infrastructure events.

The function of the approach is to find the set of $M_b \subset M$:

$\forall$ set $M_b' \neq M_b$, where $M_b' \subset M$, where two inequalities are true, and the first one relates to the utility, $RE(\Delta(\Pr(S(T_1)\Omega M_b, T_1, T_2), \Pr(S(T_1)\Omega M_b', T_1, T_2))) > 0$, where $\Pr(S(T)\Omega M_b, T_1, T_2)$ is a projected state of the system for which AC is provided at time T, such that the risk mitigation task M_b is applied during this projection. $\Delta(\Pr(S(T_1)\Omega M_b, T_1, T_2), \Pr(S(T_1)\Omega M_b', T_1, T_2))$ is a difference between two states of the system projections that are the result of the application of two different risk mitigation tasks. The simulation produces $\Delta(\Pr(S(T_1)\Omega M_b, T_1, T_2), S(T_1))$. However, as per definition, $\Delta(\Pr(S(T_1)\Omega M_b, T_1, T_2), \Pr(S(T_1)\Omega M_b', T_1, T_2)) =$
$\Delta(\Delta(\Pr(S(T_1)\Omega M_b, T_1, T_2), S(T_1)), \Delta(\Pr(S(T_1)\Omega M_b', T_1, T_2), S(T_1)))$.

This means that the difference between the results of the application of M_b and M_b' is the same as difference of the difference between the result of the application of M_b and the initial system state and the difference between the result of the application of M_b' and the initial system state. The difference between the two projections is calculated, and then the total utility is computed to determine the resulting risks as a numeric value. One way to determine the resulting risk value is to convert the risks to a single dimension. The second inequality relates to the likelihood of the critical risks.

$$\left(\int_{T2}^{T1} Rli(E_r, T, dT)\, dT, E_r : \sum_{N}^{i=1} RC(1, i, \Pr(S(T)\Omega M_b \Omega E_r, T_1, T))\right) < 0) \leq$$
$$\left(\int_{T2}^{T1} Rli(E_r, T, dT)\, dT, E_r : \sum_{N}^{i=1} RC(1, i, \Pr(S(T)\Omega M_b' \Omega E_r, T_1, T))\right) < 0)$$

Critical risks are risks that move the system (represented by the FM) from a state with positive reserves to a state with negative reserves. Assuming that all risks can be converted to some dimension (in this case, first dimension was chosen), then critical risks can be computed as a projection of the state of the system when it is converted to single dimension and its value drops below zero. The positive value of the sum of risks converted to a single dimension represents the reserves of the system or the ability of the system to withstand a certain amount of negatively impacting events. This formula takes into account that reserves in one dimension can be converted to another dimension, which would be necessary for the

system to attempt to survive severe negatively impacting events. For example, if a risk analysis is performed for a company in which productivity and financial aspects are important, then the failure condition could be a company declaring bankruptcy. However, if the company approaches bankruptcy, then certain reserves in productivity dimension (such as stockpiled products) can be turned into finances through sales, which is the conversion of the reserves in productivity dimension into reserves in finance dimension with reduced effectiveness or price. The critical risks are the risks that the chosen approach cannot alleviate. However, there is always some uncertainty about the accuracy of forecasting since the approach projects into an unknown future.

Once a simulation run of the FM is complete, a log is constructed that contains all the detailed information necessary to understand what the FM projection has predicted. The timeline of risks is a set of risk timelines for each risk dimension which plots the utility of each risk dimension versus time. As mentioned previously the utility is a number that represents the level of the risks and benefits.

Chapter 4

Case Studies

This chapter illustrates the applicability of the proposed approach through two case studies. The first case study describes a patient-privacy scenario and the second one relates to software development.

The first case study describes how the proposed approach can be used to handle medical emergencies when a doctor needs to retrieve medical records of some patients, while maintaining their privacy. The privacy is maintained by prohibiting transfer of private patient information in a digital form outside of a hospital and by enhancing all printed documents with a printed unique identifier. All shredders at the hospital are equipped with scanners identifying which documents are shredded. The approach considers two possible scenarios. The first scenario assumes that a doctor's credentials were stolen and that the emergency is falsified to obtain a large set of medical records. This scenario considers RMP 1 in which the AC approach allows access initially because initial data does not indicate that the credentials were stolen. However, if the initial data had indicated that the doctor's credentials were stolen, then the approach would have refused access. The second scenario considers a genuine emergency and two possible RMPs. The first of these RMPs (RMP 2) assumes that the requested access is allowed and the printed documents are shredded before timeout. The

second RMP (RMP 3) assumes that the access is denied. The risk analysis and forecasting were calculated manually following the approach to create three utility graphs, one for each of the three RMPs previously described in this paragraph. The utility graphs are provided and used to assess risks and benefits. The selection of the best RMP requires the probability that a doctor's credentials are stolen. For example, if the probability is low, then RMP 2 and RMP 3 are applicable, only but RMP 2 is selected because its utility is higher.

The second case study describes how the proposed approach can be used when a software developer needs to access a source code remotely. To construct the RMPs, the approach needs to configure a firewall between the developer, who uses an insecure network, and the secure corporate network to minimize risks associated with a possible source code leak. The source code contains a number of elements, each associated with two attributes: a monetary penalty and relevancy to a task. The penalty is applied if the element is stolen during transit in an insecure network. The relevancy to a software development task is estimated based on a source code dependency graph. The risk analysis and forecasting were automatically computed for 871 RMPs, where each RMP represents the allowed and denied combinations of code elements. 870 of these RMPs were elliminated based on the specific strategy that was used to assess all of them and choose the best RMP. The utility graphs that are provided show the utility for the best RMP. The best RMP represents the optimal balance between the penalty loss associated with the risk of leaking source code and the benefits gained because the developer completes the task using the relevant elements of the source code that are provided. Even though 871 RMPs is a large set, they can be forecasted in parallel, which can lower the computational time.

4.1 Patient-Privacy Case Study

Consider a scenario where the AC has to provide risk mitigation tasks that consider the risks related to two risk dimensions: patient privacy and treatment quality. A high degree of patient privacy implies that the information about a patient's condition and treatment must be released only to the attending doctors. However, knowledge about medical history can be used to improve the treatment quality of other patients with a set of similar conditions, thus potentially compromising privacy, since the access to records of patients with related

conditions can be abused by malicious entities to gain unauthorized access to their medical records.

A simplified business process of the patient treatment procedure is presented in Figure 23. A more detailed version of this process is presented at Appendix E -. Once a patient treatment starts, the attending physicians must identify themselves and either declare an emergency or proceed with regular treatment.

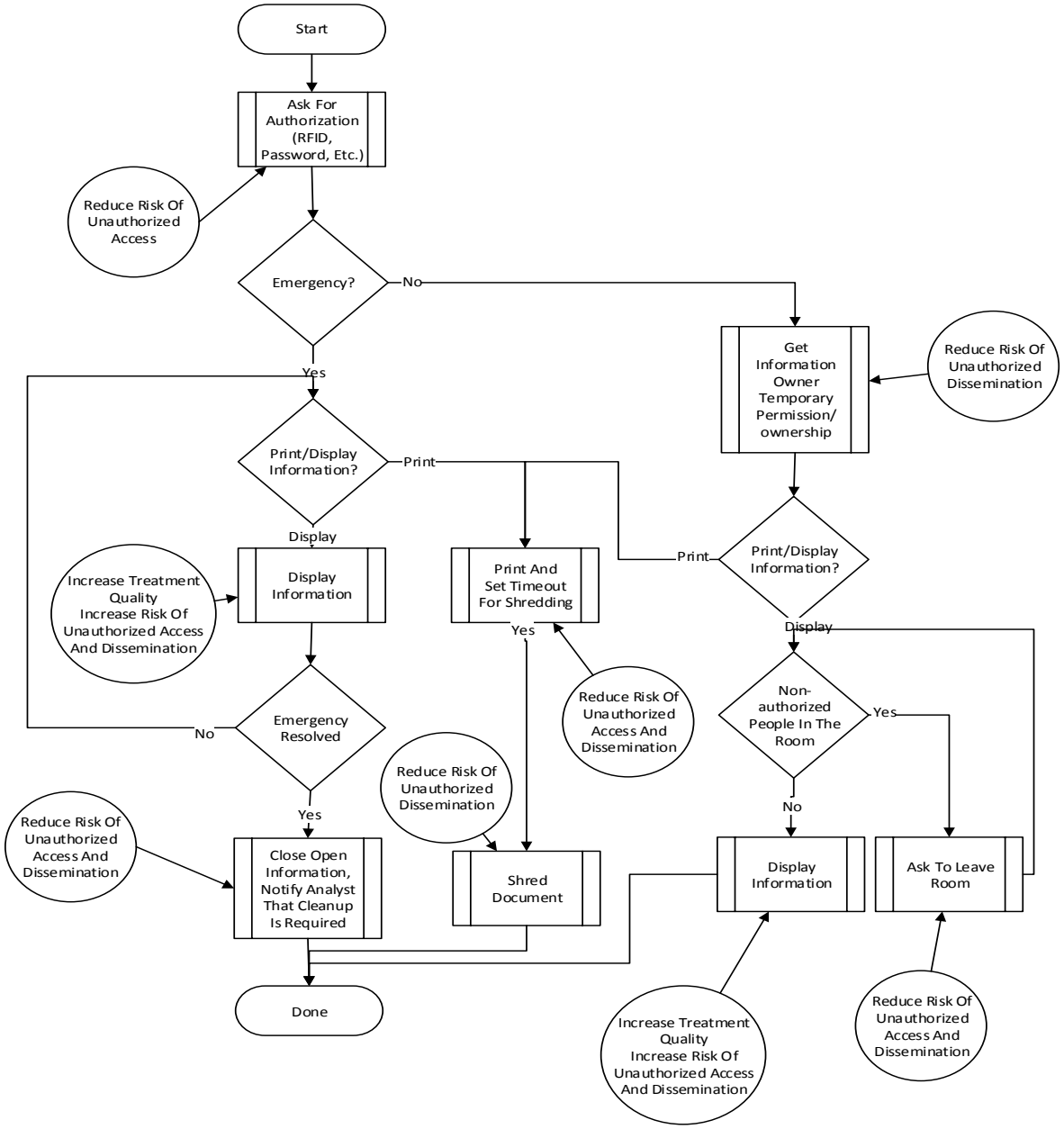


Figure 23. Patient-privacy simplified process

If an emergency is declared, the treatment quality become much more important compared to patient privacy, and some tasks, normally performed beforehand, can be completed after the emergency is resolved. To limit possible unauthorized information dissemination, each printed document related to a patient is printed with bar-code and information about each document is stored in the hospital database, including the credentials used to print, the conditions under which the document was printed (whether it was an emergency or a regular visit), the deadline for shredding this document, and the identity of the patient whose record was printed. Each hospital shredder is equipped with a scanner that scans bar code before shredding it, updates the database and sets the record of the document as shredded. A privacy analyst assesses the usage of medical records, and resolves anomalies that happen, for example, when a doctor accesses many unrelated medical records in a suspicious way. Staff and visitor identification cards in combination with facial recognition and computer vision provide location-based authorization for access to medical records. Patients and other people assigned as permanent owners of their information can either grant temporary access to or temporary ownership of the data. Temporary data ownership allows a person to grant temporary access to the data by other people and can be used, for example, by an attending physician to update other medical personal participating in treatment about specific details.

Permissions and ownerships are granted by filling out paper or electronic forms (even remotely with email notification) or by giving express permission. For example, AC, using the computer screen, notifies that a specific part of (or whole) medical record is requested to be accessed by specific medical personnel for a specified purpose and duration. During a regular visit, AC resolves access requests for medical records and additional requests based on the medical personnel needs, who then can receive information in a printed or displayed version. The computer vision, facial recognition and location based authorizations allows AC to grant access to displayed information only in the presence of authorized individuals while displayed. Printed information has to be shredded before shredding timeout expires, thus limiting possible information dissemination. Assuming that the hospital implements other security policies, such as restricting medical record transfer, prohibiting personal devices to capture images, and regulating printed information transfer, this AC provides a

comprehensive system to prevent possible breaches in patient privacy while allowing doctors to obtain the required information.

During emergencies, however, given that there is sufficient trust in the personnel declared emergency, assuming some correlation exists between the requested data and the ongoing procedure, and the restrictions are much lower, a typical utility graph of emergency is presented in Figure 24. Initially, the utility of the procedure is set to value representing the AC confidence in the doctor authorizing an emergency, which is later verified with other participating personnel. However, once the personnel start accessing the medical records, some of them are not obviously related to the particular emergency, which causes utility of the process to drop below zero. However, after the emergency is resolved, all displayed relevant information is closed and the utility starts to increase since the AC is able to provide the required information and the treatment quality with respect to information provisioning is as high as it could have been, the quality of the treatment is high, which gives a considerable boost to the utility. Finally, when all documents are shredded and the security and privacy analysis are concluded, the utility grows even higher.

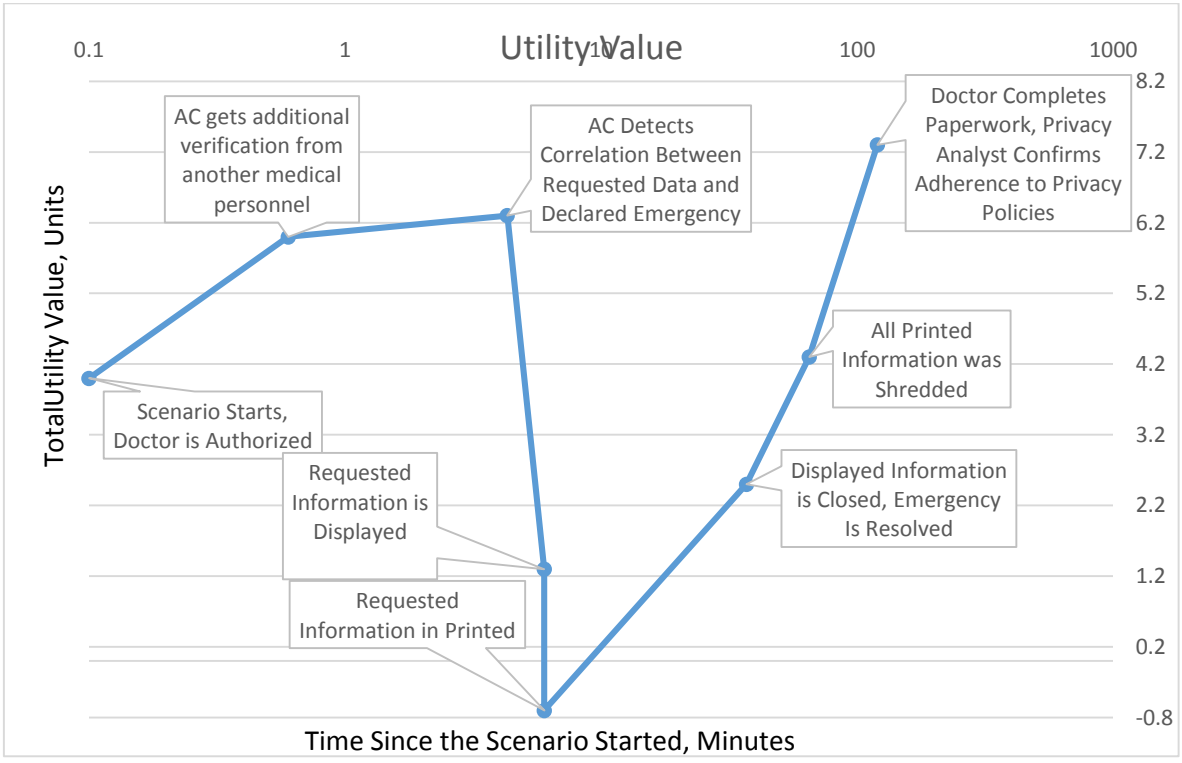


Figure 24. Utility diagram of the regular emergency scenario

The utility is calculated as a sum of all utilities related to specific domains. As it was mentioned earlier for this scenario, two domains and related types of utility are considered: privacy and patient treatment quality. Three risks are being considered, the risk of unauthorized medical information dissemination, the risk of providing poor quality medical treatment, and the risk of unauthorized access. Assume that current exchange ratio between unauthorized dissemination and poor quality is 30 to 1, while 1 unauthorized access can result on average in 20 unauthorized disseminations. The utility is measured in units of treatment quality, where positive utility means that the risk of poor treatment quality is low and the risk of unauthorized medical records dissemination is low as well.

In addition, the scenario involving an unauthorized access to medical records using stolen credentials is considered. Figure 25 presents a utility graph of this scenario. In this scenario, the utility starts with a similar value, and we assume that similar credentials were used in this and previously considered scenarios. After this, stolen medical credentials are used to verify an emergency. Then, the attackers request the printed and displayed information for a number of designated individuals. Later, we assume that AC was not able to find a proper correlation between the requested data and the declared emergency, since the target of attack was a number of unrelated medical records. However, because the medical emergency was declared and life could be lost if the information is not provided, AC allows access by displaying and printing the requested medical records, which significantly lowers the utility of the process. After a certain time, due to the lack of continued confirmation that an emergency is ongoing (for example, in case all doctors left the room with the patient and due to timeout a system believes that emergency is resolved), the displayed information is closed and the emergency is considered resolved. However, when the printed information was not shredded at a scheduled time and the security analyst discovers that the accounts used to declare an emergency and the requested data were falsified, the amount of the gained utility is low, since only risk mitigation tasks available to an AC system would be able to limit the negative consequences, for example by informing the victims of the attack the their medical records were stolen.

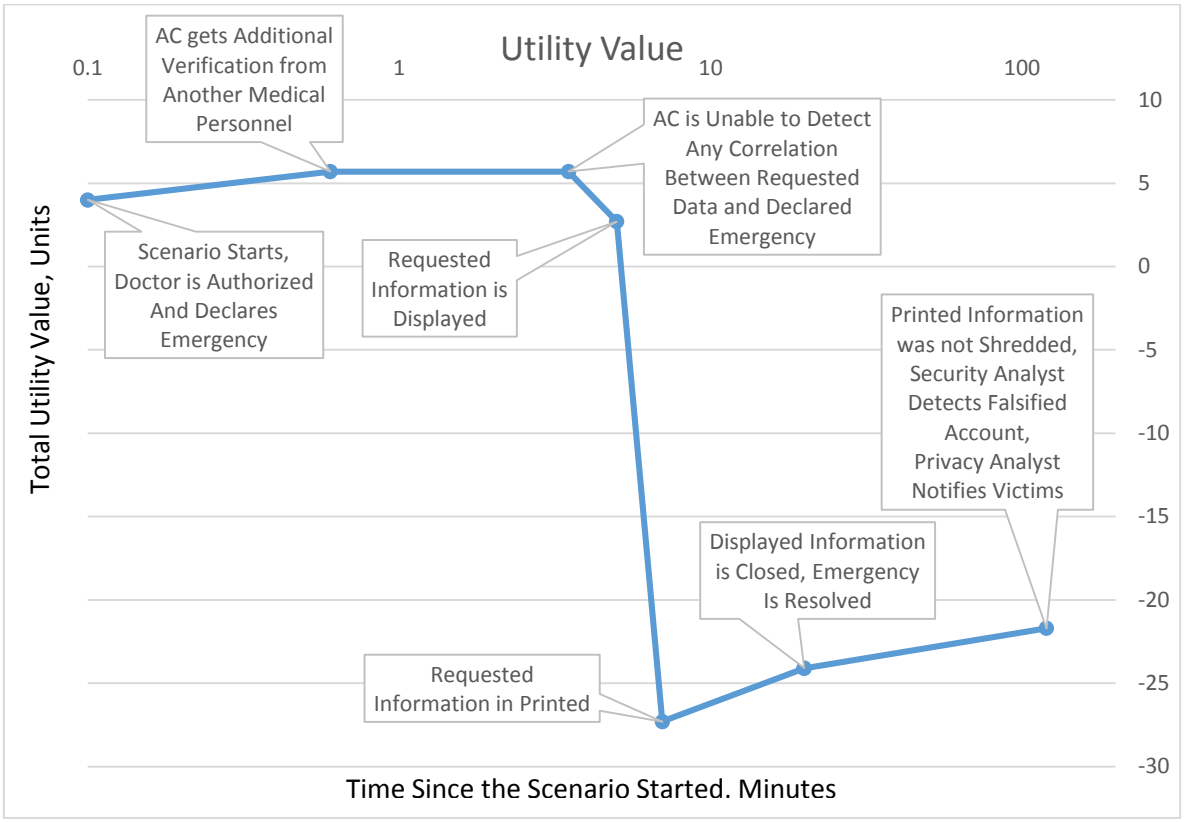


Figure 25. Utility diagram of the falsified credentials scenario

Further, another scenario was considered, where an incorrectly written correlation detection algorithm in conjunction with severe AC policies result in the doctor being denied access to the required information. The utility graph of this scenario is presented in Figure 26. The start of this scenario is similar to previous two, and the doctor declares an emergency and AC verifies it with the other medical personnel participating in the emergency. However, a poorly written correlation detection algorithm makes AC believe that this access attempt is some form of attack and due to severe AC policies, the access is prohibited, even though it is a medical emergency. At this point, the utility of the process drops sharply since the system for which AC is provided no longer can function properly and the doctors were denied access to the genuinely needed information in time. After this, the security analyst is notified of a potential ongoing attack and he or she discovers that the medical emergency is genuine and overrides the AC decision. However, as it is a medical emergency, the information is provided too late and is no longer relevant. Thus, AC refused to provide information in time and the medical emergency was not properly resolved. In some time, the printed information

would be shredded, but since the security analyst is participating in the situation directly and this is a person, the AC system puts the most trust in, the utility does not drop. However, it does not increase when information is shredded either. After this, when the privacy analyst confirms the adherence of actions to the privacy policies, a request is be submitted to correct the correlation detection algorithm. However, a service without sufficient information has already provided.

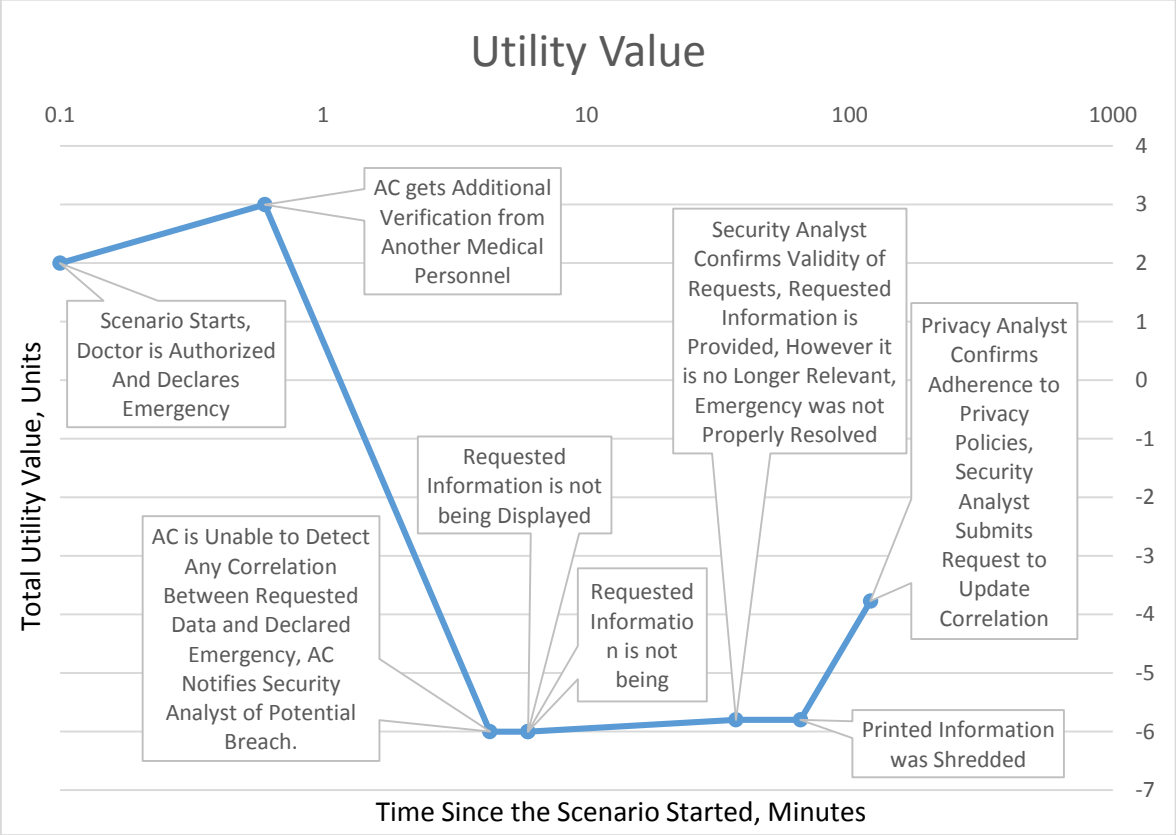


Figure 26. Utility diagram of denied access during the emergency scenario

To conclude this case, existing ACs either allow access if the credentials match a permissive policy, or deny access if there is a match to a prohibitive policy, or, if there is no match, reverts to a default policy, which is typically much simpler and still either permits or denies access. The second and third scenarios previously described respectively demonstrate that excessive permissions and prohibitions can result in undesired consequences. The proposed approach therefore relies on building a custom-tailored process for the specific situation of the system for which AC is provided. This process mitigates the risks, and, after comparing all possible options, the proposed approach selects the best applicable process,

according to the strategies designed by developers and the risk analysts that implement the approach.

4.2 Software Development Case Study

4.2.1 Overview

A software development case study was also designed to illustrate the applicability of the proposed approach to project risk levels into the future and to make AC decisions in an automated way. Table 2 presents the relationships between domains, subdomains, risks and dimensions that are represented in this case study. The case study is based on a scenario where a software developer makes the necessary changes to a product that has been developed for several years.

Table 2. Domains, risks and dimensions

Domain	**Subdomain**	**Risk**	**Dimension**
Security	Access Control	Risk of Code Being Stolen	Security, units
Business Process	Software Development	Risk of Delay	Productivity, units
Computer-Based Systems	Network & Resources	Risk of Poor Hardware and Software Performance	N/A

Figure 27 illustrates the process graph representing the typical tasks the software developer performs to maintain the software. This graph consists of twelve possible tasks and a set of transitions from one task state to another. Initially (1), the developer must find a machine with appropriate hardware and software for the software development to take place. For this task, the developer checks his or her location (room) for a suitable machine and designates the suitable machine as the one where he or she will work. The developer then obtains the specification of the task he or she needs to perform. In order to perform this step, the developer retrieves the record (the risk-related quantum of information) of this task.

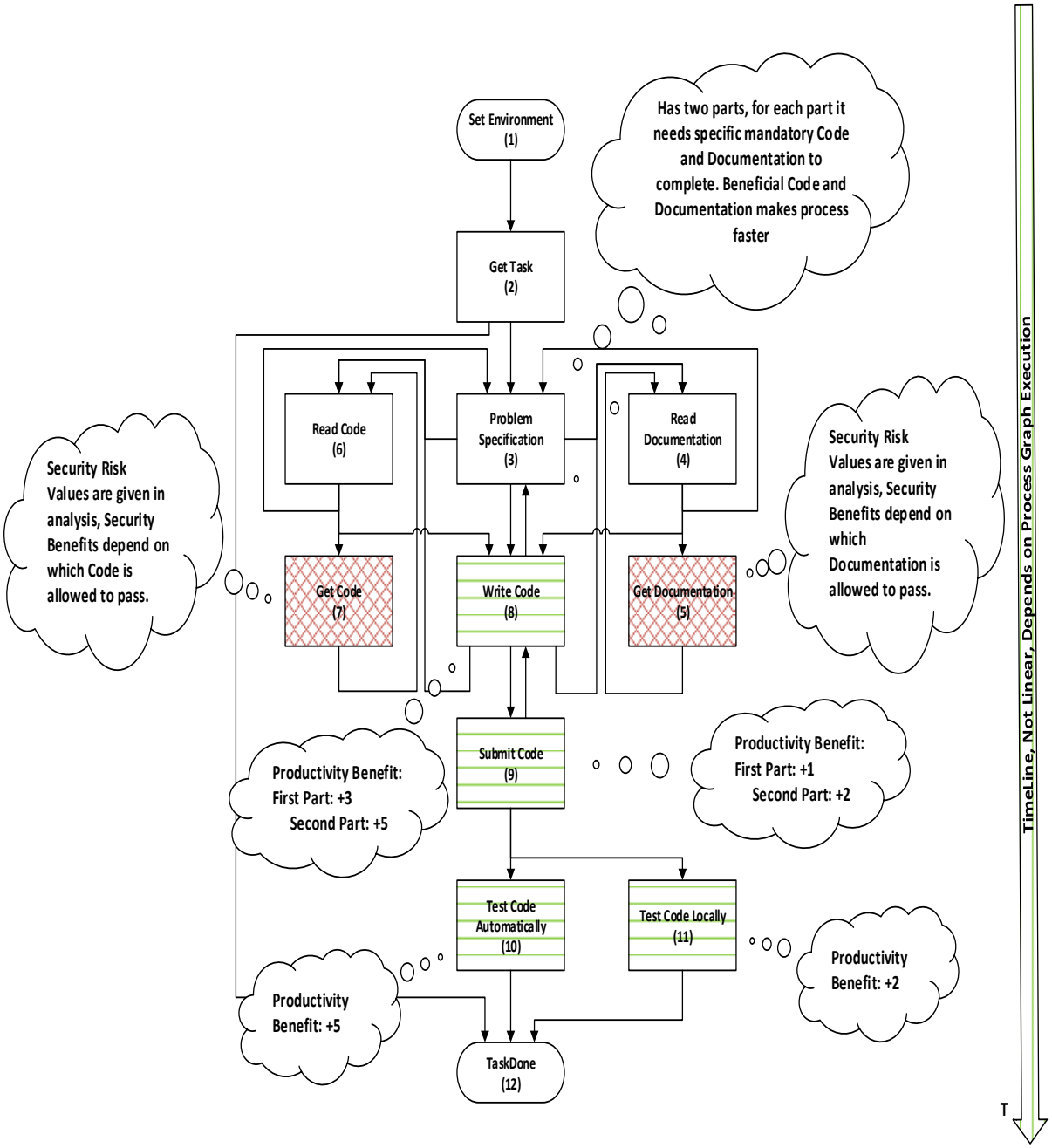


Figure 27. Software developer process graph

The network diagram of the case study is presented in Figure 28. The machine the developer is working on is indicated as I. The record is associated with two existing records in this scenario. First, the record is stored in the email record form received from the project manager at the external email server (II), while the second record is stored on the file transfer protocol server (III), which runs on the same machine as the build server (III).

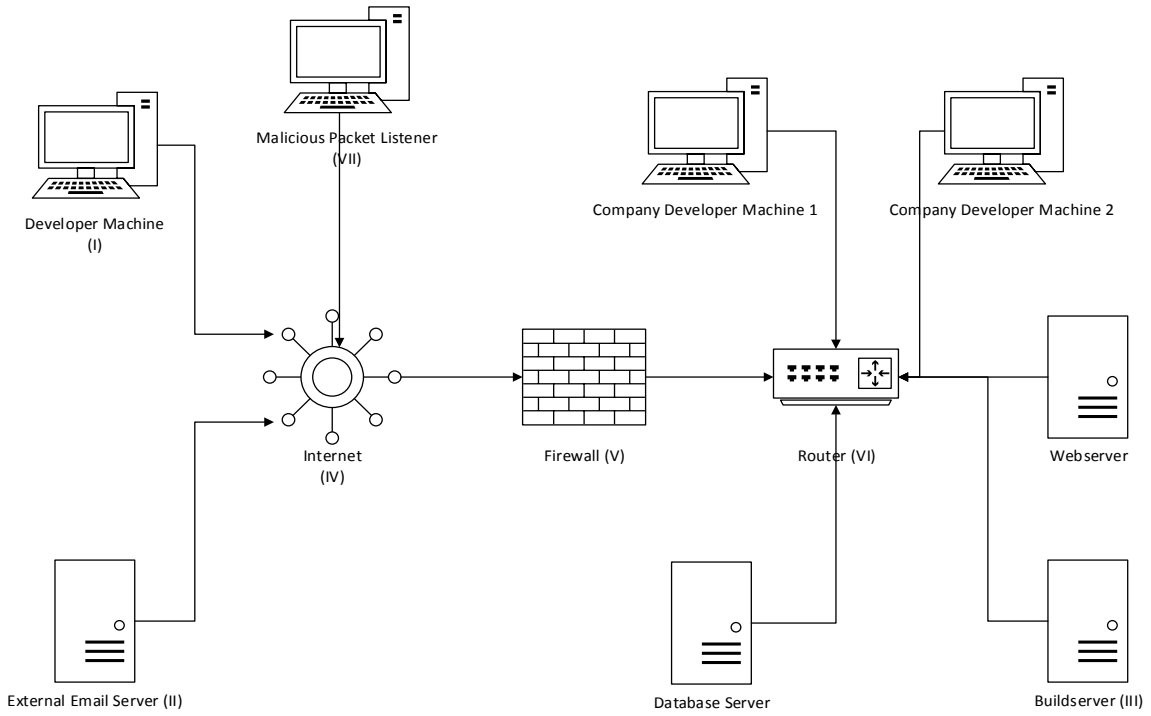


Figure 28. Network diagram of case study initial data

According to the scenario, the developer randomly selects one of the sources for the required record to be retrieved. The next step of this task is to retrieve the selected record. To retrieve the record, the developer opens a suitable component, e.g., the browser for an email record at external email server (II) and the FTP client for a file at build server (III). The component creates a message that is routed to its destination by the operating system. Both possible (because of two possible record location) messages pass through the internet (IV) first, where their contents is observed with predefined probability of 50 percent by a malicious packet listener (VII), which then attempts to recover any record associated with the message with predefined probability of 80 percent. All messages recovered by the malicious packet listener (VII) containing records that has a greater than zero security value result in a security risk that is accumulated in the risk collection. Multiple intercepted

messages containing the same record result in a higher probability of recovery by the malicious packet listener (VII). However, the risk for each record never exceeds 1, which means that in the worst case scenario, that record would be definitely known to the malicious packet listener (VII) by the end of scenario. The impact, however, depends on the sensitivity of the record. In this case study, the adversary only listens and does not attempt to extract information using any known to adversary information (e.g., account name and password for external email server II), which would result in dynamic security values of the records complicating the example and reducing its clarity. Dynamic security values of the records If the developer chooses to retrieve information from the ftp server (III), then after passing the internet (IV) the message arrives to the firewall (V). The firewall analyzes the contents of the message and can choose to stop it from continuing the traversal or to allow further message traversal. In the case where the message is stopped, depending on its policies, the firewall can send a message to the originating network node or ignore the message. If such a message is received by the component opening the communication (given that this new message from firewall is delivered), then the component will not send it again and the message sending routine returns with a response specifying that the route is blocked by the firewall (V). However, if the firewall (V) policies prohibit the firewall from delivering a response, then it does not notify the component that has originated the communication. In the case when there is no response, the originating component is programmed to repeat its query. Given that the message has successfully passed through the firewall (V) it will be carried to router (VI) and then to buildserver (III). Once the message arrives at the build server (III), the operating system of the build server (III) matches the target components port (name) and, if there is a match, the message is delivered to the ftp server that is running on the build server (III).

Most of the servers implemented in this scenario require an authorization, meaning that the first message from the originating component to the server is a query to authenticate. The server authenticates the component by locating an entry in file, database, or other data sources.

Once the corresponding user record is located, the server retrieves the role from it and creates a new session for this user. The response contains a session number and it is used to perform checks of whether this role has access to the requested action or resource. That

authentication is completed according to the RBAC. The new AC approach is designed, not as a substitute for current AC approaches, but rather to alter current AC approaches. However, it adds the ability to estimate the impact that AC decisions have on the system for which AC is provided and, by comparing the set of decisions, to choose the best applicable decision for the current specific situation. In other words, this approach is not intended to be executed for each AC decision in the system. However, at the critical points, such as the business process start or when the AC is about to refuse access, the AC must assess the risks with the context. These critical points can be identified as conditions in the process that lead to significant utility changes.

Once the session has been successfully created, the component begins the actual queries, including retrieving information and writing new information. In the case of the software development process, if the component representing the developer (browser or ftp client) is not able to retrieve (2) the task from both data sources, the process moves to the final state, task done (12). However, even though the task has been completed, there is no benefit, which means that, the system: a) wasted resources or b) increased the risk to the project when the delay is due to wasted time. Generally speaking, that approach is acceptable under the following circumstances: a) wasted resources are not critical when compared to risks that the AC would accept if the process had started; b) the risk utility of the project delay is less than the risk utility of the record being stolen; c) delaying the decision would create additional options for which the cumulative utility (combination of benefits and risks) would be higher compared to the utility of the current process (e.g., the developer moves to a location with higher security).

Given that the developer has found a proper machine in which to work (1) and is able to obtain permission to access the task specification (2), the next step is to find a theoretical solution (3) of the problem the developer is trying to solve. This step includes the analysis of the documentation (4), including log files and project documentation and the code (6). Each attempt of reading a file means that the developer will spend more time in the task. However, of course, after reading the code, the developer's understanding of each read piece of code and documentation will increase. The formula the software development domain model uses

is: NewValue = 1 - (1 - oldValue) * (0.55+Pr/10) where Pr is a probability that denotes the increased understanding during each reading cycle and belongs to the interval [0..1].

However, not all the machines that the developer can work on have the documentation or source code files presented in Table 3. Not having the documentation or source code leads to the necessity of retrieving documentation (5) and code (7) that, according to the scenario, is located on the build server (III). Since the developer knows the location of this information and has setup the accounts needed to access it, he or she can retrieve the required information. However, since there is a risk that this information is intercepted by a malicious packet listener (VII), there is a security risk associated with this action. Further, since the approach is building RMPs, part of behaviour of these processes is to choose which records the approach should prohibit from being sent out and therefore from being intercepted, there is also a security benefit related to these tasks (5, 7). The scenario is setup in a way that both the documentation and the code have four files that can be utilized for analysis. However, having and reading all mentioned files is not mandatory. While defining a problem specification (3) and writing code (8) a developer may need to read some of the file parts. Note that each of these tasks has two stages.

Table 3. Documentation and code attributes

File Part \ Attributes	Problem Specification	Write Code	Security Risk
Documentation, A	Mandatory for stage 1	Beneficial for stage 1	0.7
Method header, A	Mandatory for stage 1	Mandatory for stage 1	0
Method body, A	Beneficial for stage 1	Mandatory for stage 1	1
Documentation, B	Beneficial for stage 1		0.5
Method header, B	Beneficial for stage 1		0
Method body, B			2
Documentation, C	Beneficial for stage 2		0.1
Method header, C	Beneficial for stage 2		0
Method body, C			0.05
Documentation, D	Mandatory for stage 2	Beneficial for stage 2	0
Method header, D	Mandatory for stage 2	Mandatory for stage 2	0
Method body, D		Mandatory for stage 2	0
Development task descriptiption	Mandatory for process to start	Mandatory for process to start	0.2

Although each of the tasks previously mentioned, namely problem specification and writing code, has two stages, for a developer to start the task of writing code he or she needs to complete the problem specification task. The problem specification task does not have productivity benefits associated with it. However, there are productivity benefits associated with completing the task of writing code (8), because, even if the code was not submitted to the repository (9) as a next task in the software development process (Figure 27), the code has already been produced and, for this reason, the productivity benefits can be taken into account.

Each of these two stages stage of both tasks (that is, problem specification and writing code) has different mandatory and optional file parts needed to solve this task (Table 3). If the developer does not read any file part that is mandatory for the task, the probability of solving this task is 0. However, as the developer reads optional file parts, the probability of solving this task increases. Each probability depends on the previously specified value of the level of understanding of the required and optional code and documentation. In summary, the formula used in the software development domain model results in 0, if any mandatory piece of code or documentation was not read at all, and (Sum of read & understood file parts) / (number of required records), otherwise.

While a typical developer retrieves all the eight files that are relevant to the process, some of these files present a higher security risk and, taking source code analysis into account, the approach makes it possible for the AC to detect which files are more likely to be needed. In other words, it is not necessary to transfer the entire source code and the documentation. In fact, it is possible to split existing files into different pieces of information. For example, source code is structured in classes that typically include the documentation for the class, documentation for each method with references to input and output parameters, their classes and acceptable values, documentation for class attributes, class attributes, their default values, and method headers and bodies. In addition, each piece of information can exist in different forms. For instance, the source code can exist in plaintext, obfuscated, and compiled (binary) versions.

Each piece of information for the scenario can have a different usage. For example, if a certain file is needed only to compile the project, there is little to no difference from the

productivity standpoint between supplying the compiled version of file and the plaintext source file. However, from the security standpoint, the risk of transferring a plaintext source file is much greater than the risk of transferring the binary version of the same file. Traditionally, such a problem would be solved by grouping files into certain types (e.g., a set of classes responsible for processing a customer order and a set of files responsible for writing information to a hard drive). Since developers tend to specialize on a specific set of technologies, this grouping and assigning certain developers as responsible for these groups does make sense. However, specialization makes an approach with finer granularity that separate method headers from method bodies inapplicable.

Given that it was possible to complete at least one stage of writing the code, the developer can submit it (9) to the buildserver. This process obviously results in certain productivity benefits, and given that both stages were solved, it is also possible to run tests automatically (10), given that the buildserver is working properly, or run tests locally (11), given that the buildserver is not responding. Testing the solution also results in productivity benefits, since it brings the development to a conclusion (12) that creates a fully usable product. However, it also should be noted that there are costs associated with the development, which increase productivity risks linearly depending on the time the developer spent working on the solution to the problem.

The goal of the case study is to find automatically the optimal composition of files that should be transferred to the developer machine with a different context (location). As previously stated, different strategies and different states of a system for which AC is provided lead to different results. For example, if a strategy favours utility, then the file generally is transferred to the developer machine when needed unless there is an explicitly high security risk associated with this file and the utility of it to the developer is not particularly high. However, when the same strategy constrained by a closing deadline to avoid deadline penalties, the approach allows the developer to transfer files associated with an even higher security risk. When the goal of the strategy is to minimize critical risks, it might elect to restrict transfer even when the file is optionally needed but is not a mandatory requirement. For mandatory required files with a high security risk, the approach can choose to mitigate the risk by querying another developer or project manager responsible for

maintaining the file. As these strategies are changed at runtime, it is possible to have a highly dynamic reaction to certain events. For example, normally, components operate with a maximum utility strategy. However, if the risks associated with a certain dimension become critical, e.g., the approach detects that a successful attempt to gain unauthorized access to information would result in a security becoming a dimension with critical risks, or an approaching deadline would result in productivity becoming such a dimension. The components operating with this dimension (e.g., firewall for security or task management software for productivity) switch to a strategy with minimal critical risks. The switch allows components to adapt to changes seamlessly in the state of a system for which AC is provided. Current approaches to this problem focus all efforts on dealing with critical risks. In other words, if a security breach is detected, all communications can be severed until the situation is resolved. This risk mitigation task, however, does not work if two dimensions are critical at the same time. For example, if there is a deadline for project, severing communication to prevent security risk also prevents project progress. The total utility formula used in the software development domain model is: utility = (security benefits - security risk) * 1000 + (productivity benefits - productivity risks(0.0625 * time)) * 400, where 1000 is a current value of each unit of security benefit or -1000 is the negative value of each unit of the security risk. This means that the security risk associated with one unit of security risk is estimated to cause a system to lose 1000 dollars. The productivity benefit units are currently considered to be equivalent to 400 dollars, and each millisecond, used as a time measure unit in this case study, is associated with gaining 0.0625 units of productivity risks.

As previously stated, the proposed approach allows for the mutually beneficial work of components operating with different knowledge domains in terms of risks and benefits. Thus, each AC decision is custom-tailored to address specific situations for a set of components with different goals and strategies. It is not always possible to create a RMP that significantly reduces all risks, even in one dimension. To manage risks, the new approach uses QRMs that accumulate the risks and benefits in form of reserves, while operating with the maximal utility. To summarize, while there are no critical risks to mitigate, the new approach reduces them in a future. This future reduction of risks related to any of the risk dimensions, which contributes to increase the reserves, can be utilized when critical risks appear. The reserves can be converted to mitigate risks related to any risk dimension, and if

a certain dimension needs certain resources to resolve internal problems, it can borrow risks from other dimensions. However, once the dimension's problems are resolved, the components should return the borrowed risks. With respect to the software development case study, the utilities of the two risk dimensions, security and productivity, are combined by a static conversion rate because the scenario runs for a time that is less than a day and there is no external sources of risks to be considered. Thus, the risks and benefits of the developer's actions from both dimensions are correlated with the static ratio.

Given that the developer has successfully solved the first stage of the problem specification (3), he or she can begin to either write the first stage of the code (8) or solve the second stage of the problem specification (3). Both these tasks depend on whether the firewall permits the necessary and beneficial records listed in Table 3 to pass through. The firewall operates according to its policies and its policies are set by the policy decision point as a part of the scenario for simulation in the FM. For this scenario, the policy decision point can either allow all communications through the firewall or can block certain records. The goal of the simulation is for the policy decision point to find the optimal set of records that must be blocked for the optimal function of the system. The setup of the current scenario results in high risks associated with low utility for both security and productivity. Thus, the situation in which the new approach might elect to accept high risks (becoming critical) to obtain a higher utility or when the approach does not accept high risks at a penalty of not receiving a higher utility. Both situations are suitable for different strategies, which would propose a different course of actions.

4.2.2 Simulation Results

The optimal functioning for this scenario is defined by the utility of all the tasks. The highest utility of productivity is achieved when the developer is able to complete all the work in the shortest amount of time. In order to complete each of the two stages for the problem specification (3) and writing code (8), the developer needs all mandatory records listed in Table 3 to pass the firewall to his or her machine. However, access to the optional records listed also in Table 3 increases the probability of each attempt to complete these tasks. Thus, for the optimal functioning from the point of the productivity dimension, the firewall should allow passage of all records that are mandatory or beneficial.

The utility of the security dimension is dependent on the number of times the malicious packet listener is able to intercept the records with a security value that is greater than zero. These values are provided in Table 3. Thus, for the optimal functioning from the view of the security dimension, the firewall should not allow passage of any record that has a security risk value higher than zero. However, there are records that are both mandatory and beneficial and that have a security risk value higher than 0; therefore, the optimal behaviour for both dimensions cannot be achieved and a compromise between risks and benefits of both security and productivity must be found. From all existing combinations, the case where the firewall policy is set to prevent traversing of documentation B, method body B, documentation C, and method body C has the highest utility located in Table 4 and represents the values projected into the future security risk, productivity risk, total utility at a scenario when the firewall is not allowing passage of messages containing documentation B, method body B, documentation C, or method body C.

Table 4. Simulation results for documentation B, method body B, documentation C, and method body C

Time in Milli-seconds	1980051	6800151	17520312	17640331	17640332	18240332	18360372	18420382
Security Risk in Units	0.56	1.36	1.36	1.36	1.36	1.36	1.36	1.36
Produc-tivity Risk in Units	0	0	-3	-3	-4	-9	-11	-16
Total Utility in Units	-573.7	-1407.2	-281.6	-282.5	117.4	2113.3	2912.4	4912.0
Time in hours	0.55	1.88	4.86	4.90	4.90	5.06	5.10	5.11

In Figure 29 the security risk value over time is presented. The security risk is increasing because the firewall is set to allow passage of messages containing sensitive data and these messages are intercepted by a malicious packet listener. While the actual values start at zero at time zero, the final level of security risk is 1.36 units.

Figure 29. Security risk value over time

In Figure 30 the productivity risk value over time is presented. The values start at 0 and decrease to -16 units. The value decreases because the developer executes software development process and the benefits of his or her productivity thereby reducing the risk of a project delay, which is a part of productivity dimension. Initially, productivity increases slowly because the developer needs a significant amount of time to prepare for future development, which is to download and read the required documentation and source code files. However, after approximately 4.9 hours, the developer has read and analyzed most of the required data while performing the first stage of the problem specification task (3). After reading and analysis, the work progresses much faster, and, in next 20 minutes, the task is complete.

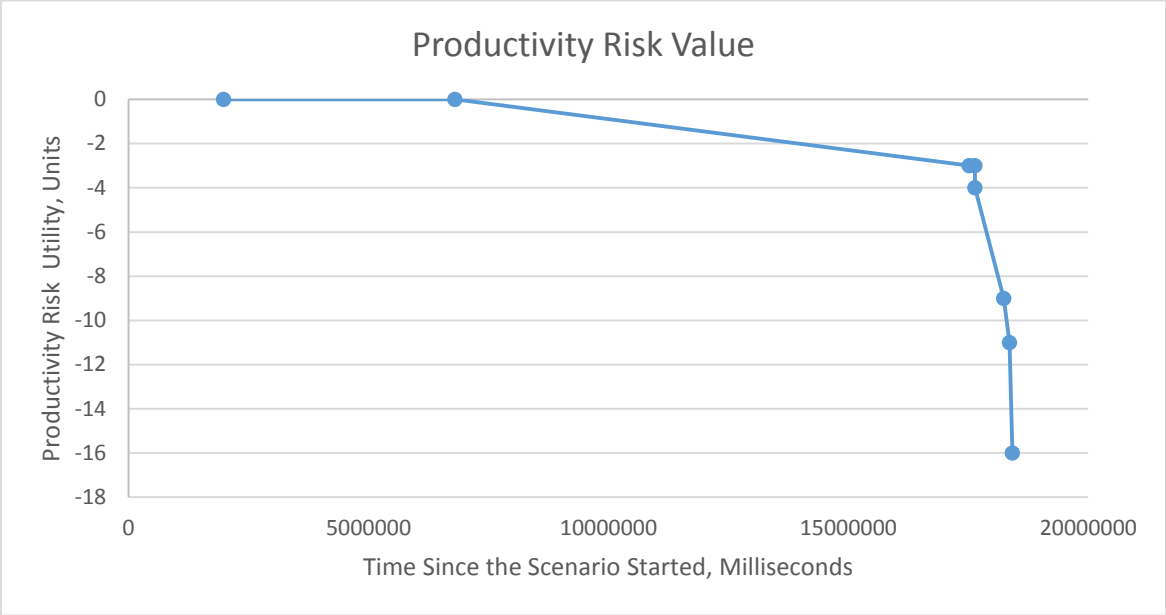


Figure 30. Productivity risk value over time

In Figure 31 the total utility value over time is presented. The value initially drops below 0 because of the security risk that the AC must accept to allow the developer to read all the required documentation and the source code. However, after 1.88 hours, the utility starts to slowly increase because the developer is able to solve the first stage of the problem specification task (3), and therefore, the benefit of productivity reduces the overall risk. After 4.9 hours, the value starts to sharply increase, exceeds zero and reaches 4912 units. Zero in this case is a relative level of risk when the developer starts the task. Obviously, since some work was done previously, zero is not an absolute level. However, since the set of RMP evaluated in the case study does not depend on other processes while being executed, it is possible to set this level of risk to zero. If other processes were involved, then the difference between the final projected level of risk and initial level of risk would be the utility value. However, in that case, the AC would have to separate this RMP from other processes by simulating other options of this RMP and obtaining the difference between utility results.

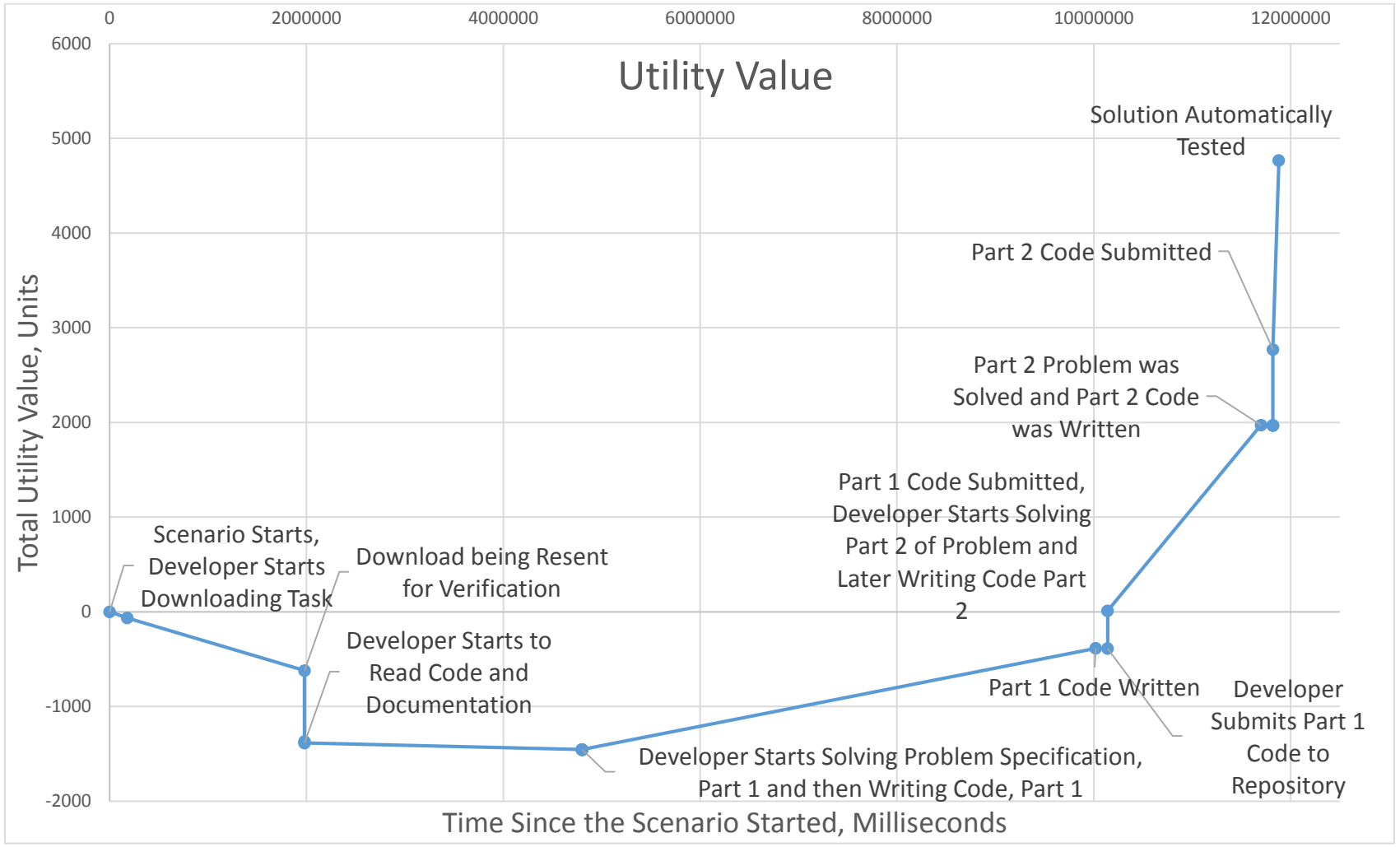


Figure 31. Total utility value over time

The final values of the utility for this scenario indicate that, according to the approach, it is generally useful to adopt this set of policies for the firewall while this process is ongoing. In Table 5 the simulation results are provided for the scenario in which the firewall does not allow the following records to pass through the unsecure network: method header B, documentation C, email from project manager are presented, where email from project manager contains the development task. This record is included because, even though it is not located within the secure part of network, it is a record that exists in the system for which AC is provided and this approach instantiation was built to verify all possible firewall policies.

Table 5. Simulation results for method header B, documentation C, email from project manager

Projected Time in Milliseconds	180029	2100069	3100109	4100149	11620259	11620279
Security Risk	0.16	0.96	2.56	2.6	3.16	3.56
Productivity Risk	0	0	0	0	0	0
Total Utility	-161.3	-974.6	-2581.5	-2628.5	-3240.7	-3640.7
Time, hours	0.1	0.6	0.9	1.1	3.2	3.2

Table 5 and Table 6 presents the simulation results over time.

Table 6. Simulation results for method header B, documentation C, email from project manager

Projected Time in Milli-seconds	18720350	18840369	18840370	20940370	21060410	21120420
Security Risk	3.56	3.56	3.56	3.56	3.56	3.56
Productivity Risk	-3	-3	-4	-9	-11	-16
Total utility	-2490.0	-2490.8	-2090.8	-105.4	693.7	2693.3
Time, hours	5.2	5.2	5.2	5.8	5.9	5.9

The initial configuration is not optimal because, even though this RMP was completed and there are 16 units of productivity benefits for the entire project, the previous firewall configuration had 4912 of total utility and this one has only 2693. However, the productivity utility is sixteen units of utility below zero because of the greater amount of time used by the developer for this project and the much higher security risk that occurred while the developer retrieved data from the buildserver.

In Figure 32 the total utility for another RMP is presented in graphical form. Even though there was an initial decrease in the utility value that has occurred because the AC was accepting security risks and not gaining productivity benefits, it was compensated by later growth. As well, the bottom of this graph is much lower than bottom of the previous one, which means that with low initial reserves these risks can become critical.

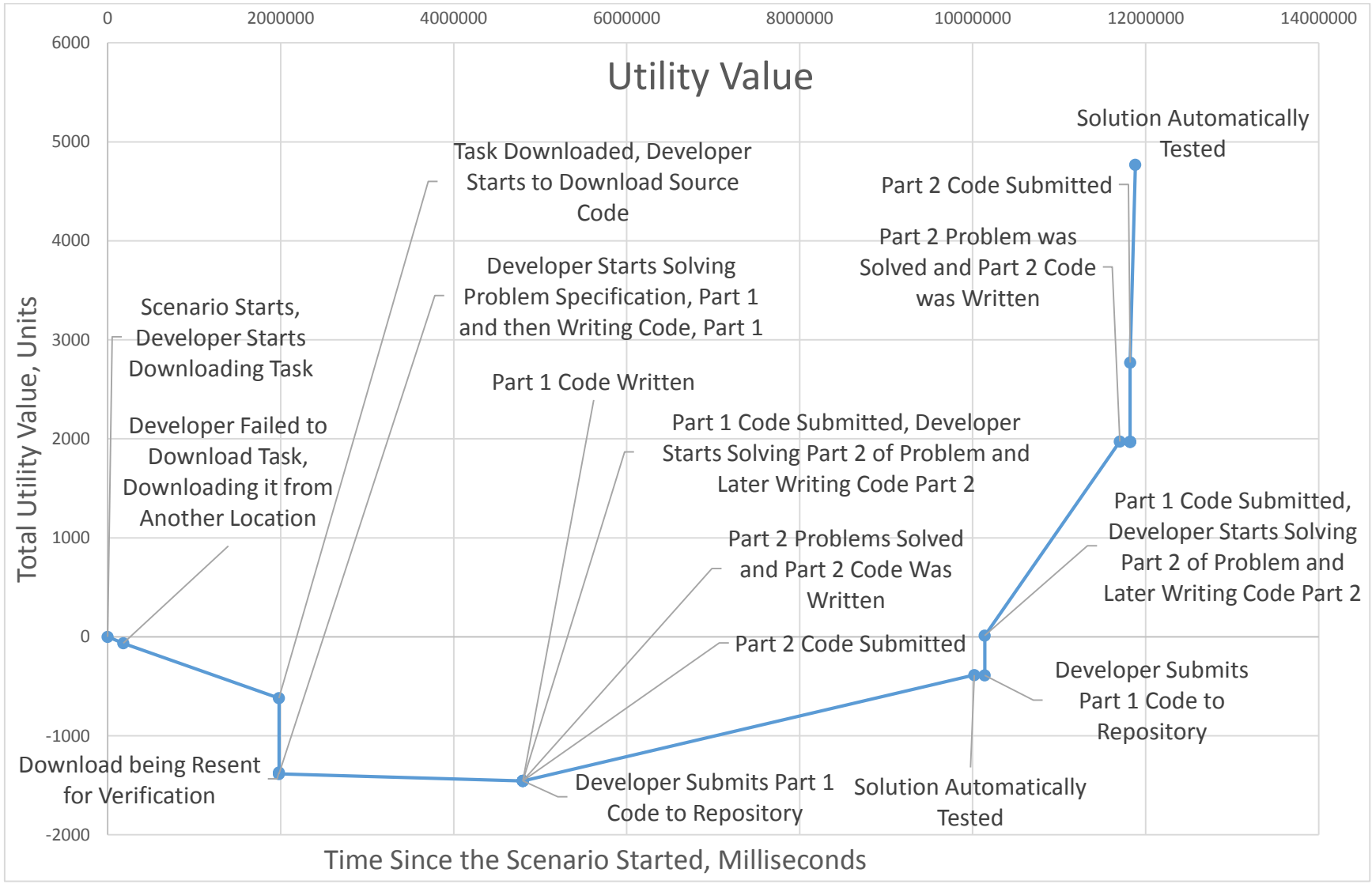


Figure 32. Total utility for another RMP

The total utility of the RMP can be quite different depending on the specific scenarios. The simulation of the FM that adopted the policies where method body B, documentation C, method header C, and method header D were blocked by the firewall, resulted in the utility being lower than zero, which means that this approach is detrimental to the company.

Table 7. Simulation results for method body B, documentation C, method header C, method header D

Time in Milli-seconds	180029	2100069	4100149	7800219	7800239	16100310	16220329	16220330
Security Risk	0.16	0.96	1	1.56	1.96	1.96	1.96	1.96
Produc-tivity Risk	0	0	0	0	0	-3	-3	-4
Total utility	-161.3	-974.6	-1028.5	-1614.2	-2014.2	-871.8	-872.6	-472.6
Time, hours	0.1	0.6	1.1	2.2	2.2	4.5	4.5	4.5

In Figure 33 the total utility value over time for method body B, documentation C, method header C, method header D is presented. The result, which is lower than zero, can be explained by the lack of productivity. The developer managed to achieve productivity benefit of 4 units, but he or she was not able to balance the security risks that the system incurred while the developer downloaded the needed data and tried to complete this task.

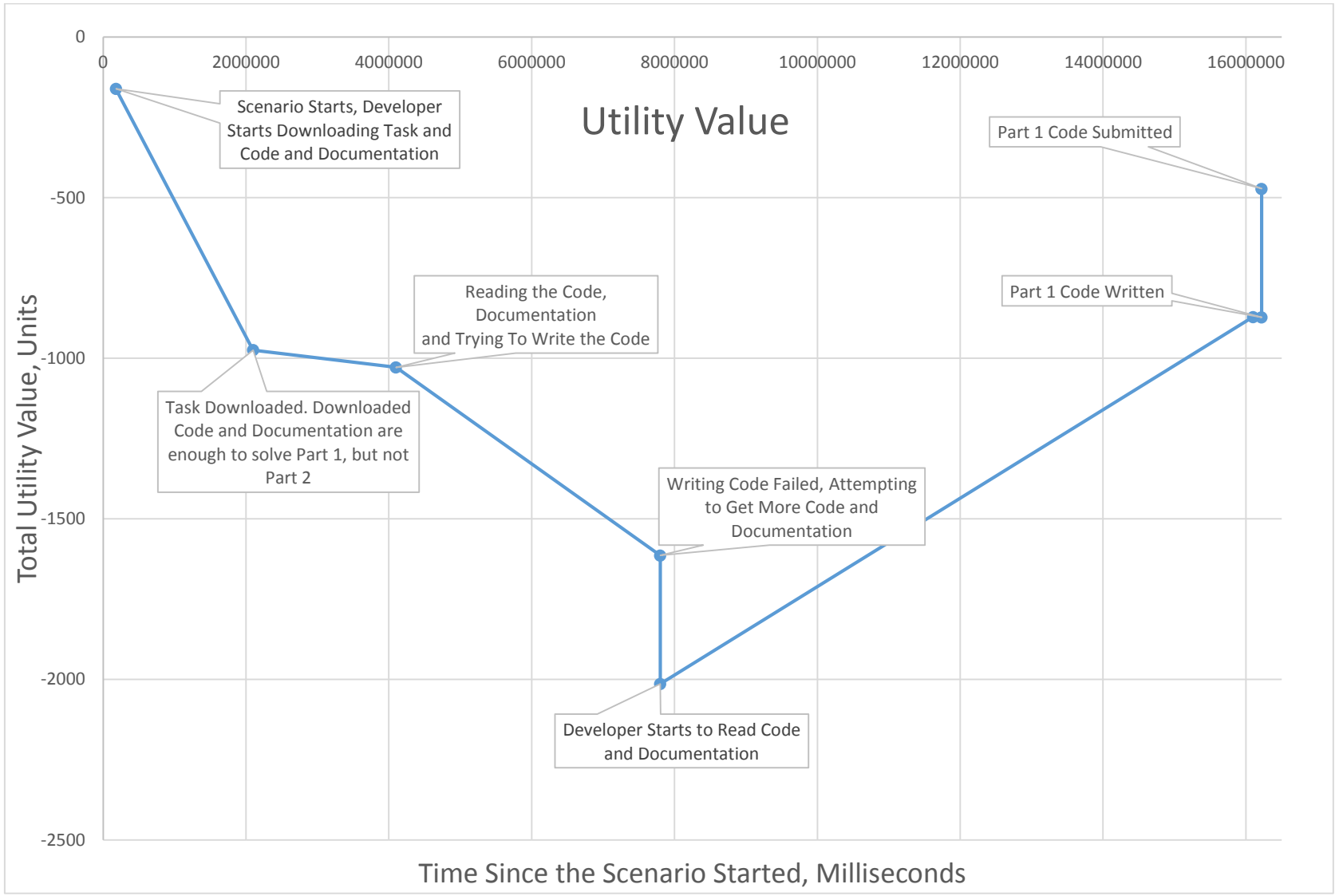


Figure 33. Total utility value over time for method body B, documentation C, method header C, method header D

4.2.3 Discussion of the Results

This case study presents a scenario in which a developer makes the necessary changes to a software product. The approach is required to mitigate risks by changing the firewall policies before the start of the process by adding one or more different policies that block specific records to prevent them from traversing into an insecure network. The best decision for the approach was to block the documentation B, the method body B, the documentation C and the method body C. An analysis of the material of Table 2 verifies that this RMP is the most logical among all possible records combinations that the firewall could block or allow passage. It is important to note that the risk analysis was performed automatically. Moreover, a software development process was adapted at runtime to take into account the risks and benefits incurred during the automated analysis. Furthermore, some AC policies, e.g., firewall rules, were generated at runtime based on the adaptation of the process. The trade-off between security and productivity was also assessed. In case the access to the required information was denied, all gained security benefits were reduced by increasing the productivity risk, and the processes that required the access to the information to be denied are disregarded. All simulations had a negative utility minimum initially due to developer's need to study the problem first before a solution could be found and benefits could be given because the developer needed to read the code and to read the code the developer needed to transfer information over an insecure network, and for this reason, this task increased the security risks. Thus, the novel approach demonstrated the ability to project the state of a system for which AC is provided into the future and, by analysing multiple projections, this approach made a logically correct decision.

4.2.4 Domain Models in the Case Study

The domain models used in this case study include the AC, software development and network models. These models are presented in the UML diagram in Figure 34 and Figure 35. These are simplified UML diagrams, which present only the data and the relationships among the classes. A detailed diagram is provided in Appendix B.

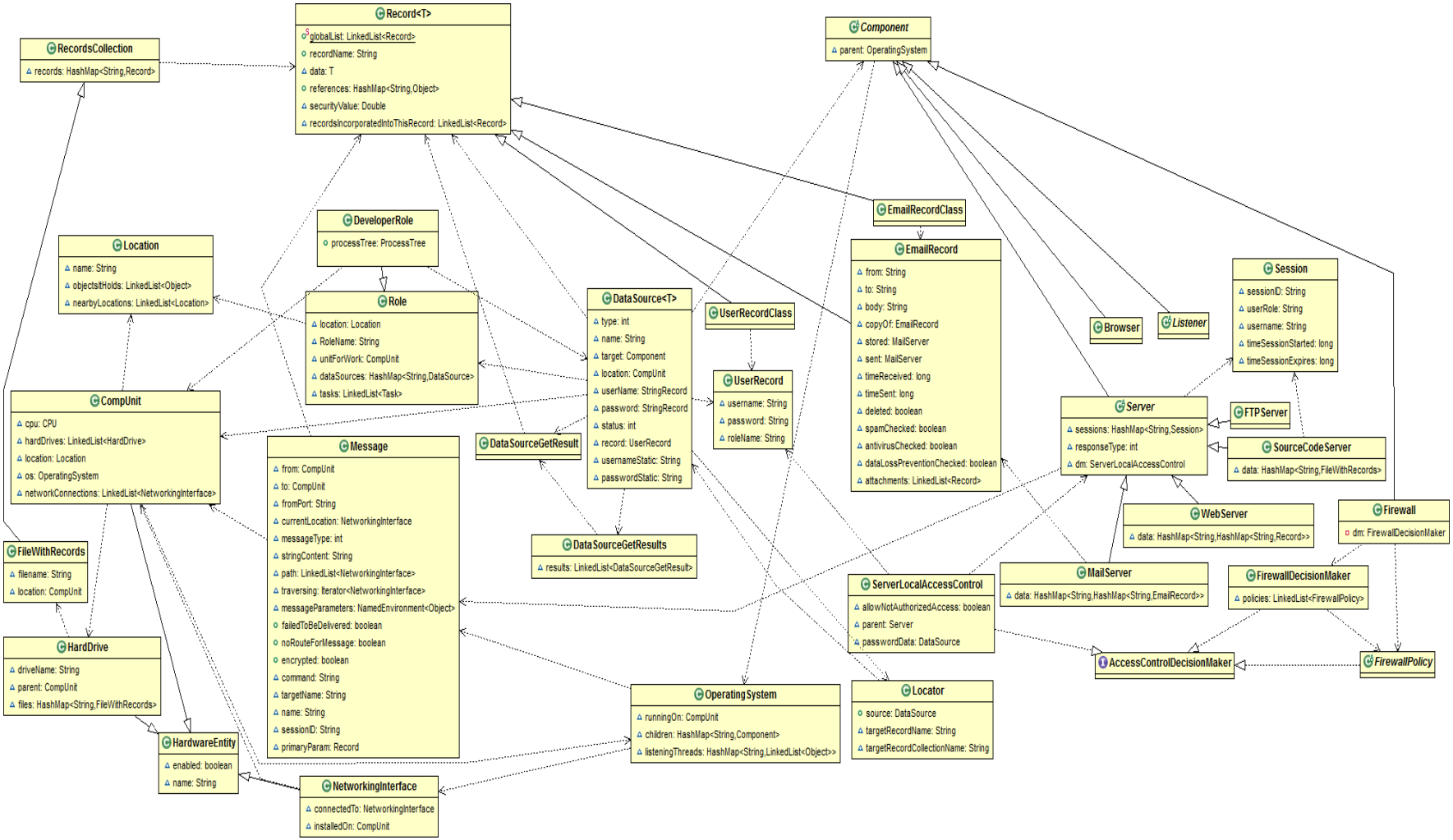


Figure 34. UML class diagram for the domain model in the software development case study

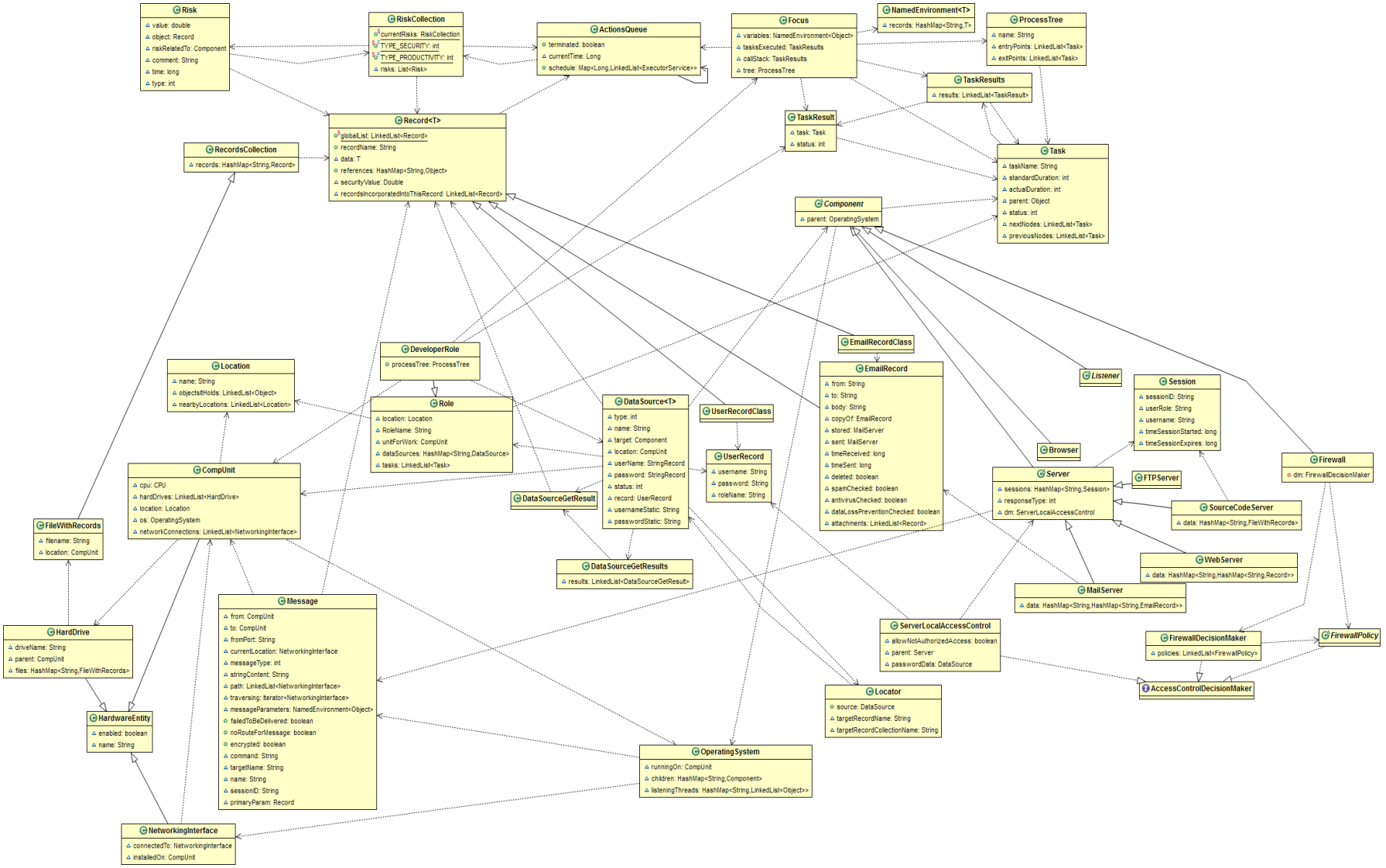


Figure 35. UML class diagram for the approach and the domain model in the software development case study

The AC model includes the severlocalaccesscontrol class, which contains data about the passwords used on the server (e.g., mail server). The software development model is represented using a developer role class and the process graph that describes the sequence of actions that a developer typically goes through to make changes to a software product. Network and resources are depicted for example in the operatingsystem class, which represents the components of an operating system that support access to files and communication to specific devices such as servers and the firewall.

4.3 Implementation

The improved design model was implemented in Java language version 1.7. The forecasting core model contains 12 classes and domain models contains 50 classes in 59 files and 4445 lines of code. The full listing of source code is presented in Appendix A. The main class starting execution is called RMS and it does not require parameters. The method main of this class is responsible for organizing the risk mitigation tasks before the scenario execution. The implementation produces logs specifying what is projected to occur in the system according to the FM as well as the resulting level of risks for all the dimensions, the time elapsed, and the timeline of risks in CSV format.

Chapter 5

Summary and Future Work

5.1 Summary

The thesis presents a new approach in risk analysis and mitigation. The new approach is an extension of the approach proposed in [126]. The presented approach is designed to manage risks in dynamic conditions, in which components are not required to follow directions from a centralized authority. Current risk analysis and mitigation approaches rely on the AC to resolve risks originating in the security dimension. In addition, RAdAC lacks implementation support. Both existing and emerging AC approaches (e.g., RAdAC) were shown to have limitations in their expressiveness and in their lack of specific features.

Each of these approaches uses predefined reactions to certain risks, which can be anticipated and countered. Existing AC methods also follow a predefined AC process. If this approach is observed by malicious entities, tasks that AC takes to mitigate risks can be anticipated and prevented. Furthermore, an existing AC fails to consider explicitly different dimensions in which current AC's work. This failure can result in unbalanced tasks where one dimension is favoured over others, and thus, risks increase.

Additionally, current AC approaches ignore risks that are considered negligible. However, when a number of tasks is executed, these risks become too significant to be

ignored. Finally, existing risk analyses are based on only human analysis and cannot dynamically adapt to a changing environment.

The presented approach is capable of performing the risk analysis simultaneously for a set of a predefined risk dimensions and is able to respond to risks by dynamically creating a set of RMPs for components to follow. The approach does not require components to follow constructed RMP. However, in case there is no centralized authority, tasks that need the combined efforts of a set of components would become possible only if all interacting components participate in a mutually beneficial RMP. The risk analysis in the proposed approach is performed by projecting the state of a system into the future via the simulation of the FM. The state of the system is updated and the FM can also be updated as the AC approach monitors events occurring in the system, and thus, the analysis is no longer predefined. All the risks, including the least significant, that the AC system encounters, accumulate in components gaining or loosing QRMs, which are required to motivate the cooperation of other components. A balance among different dimension risk needs is obtained by adjusting the exchange ratio of QRMs for different dimensions. Depending on probabilistic model, the simulation can depend on multiple runs and result in average values (e.g., averaging utility over multiple runs resulting in different outcomes) or can specify the potential deviation for the projected values.

The activities of the approach start when a component originates a resource or service request and ends when the AC request resolution finished with the creating a set of RMPs. RMPs are created by applying various relevant risk mitigation measures and projecting the state of the system into the future in which these decisions have been applied. After the projection of the state of the system into the future in which these measures are applied, the timeline of risks are compared and user-defined strategies select the best possible RMP. Once every strategy has constructed its required RMP, the set of RMP is returned to the component in the form of an AC decision. Upon receiving these RMP, and depending on intelligence of the particular component, the component can determine which one of proposed RMP suits its needs. The example for the approach offered in the thesis also demonstrates the calculations necessary to conduct risk analysis manually. Based on different relevant sets of data, the AC decision is created. This example shows also a data structure that is used to

collect data required for the risk analysis. Moreover, the example presents an algorithm for utility calculation for scenarios using risk analysis and the function that converts utilities to the utility chosen as a common basis.

Disclosing this information typically results in the danger of releasing sensitive information to a component that requests an AC decision. However, if no disclosure is possible, then a limited set of tasks (if any) is performed cooperatively by the components. Additionally, as the FM does not require the input of all the information possessed by the component and is able to project the component's state into the future with a limited set of historical and present data, the components can select which information to share with a third party component. Furthermore, existing approaches by restricting access disclose information about the access attempt. For example, if an attack attempt is identified by the AC, the AC can respond by allowing access in a virtualized replica of the system for which AC is provided with scrambled information, which results in identifying the attacker's targets and methods. While this technique is known as honeypots, current AC approaches are unable to initiate a honeypot dynamically. By granting or denying access to the requested resource, existing AC's reveal information about the beliefs of AC regarding component requesting access, which can be used to assess internal variables of AC and consequently modify attack.

This thesis also presents two case studies that were constructed to illustrate the applicability of the approach. A software development case study was designed to demonstrate the feasibility of automating the approach. In this case study, the AC is required to make a decision about the software development process that is about to start. The risk analysis is performed by projecting the state of the system into the future and comparing the risk timelines of possible decisions and the arguments for them. By comparing the decisions and arguments, the implementation of the approach resulted in an AC decision that was correct for the given data compared to a manual evaluation. Thus, the approach demonstrates its ability to perform risk analysis automatically.

5.2 Limitations

The proposed approach assumes that in the paths of the process graphs, the precondition of a previous task matches the postcondition of the subsequent task. This assumption is made because preconditions and postconditions were analyzed by the designers of the processes at design time. However, when the approach constructs RMPs, restrictions related to mutually and non-mutually exclusive tasks are taken into consideration. Therefore, these restrictions can be seen as simplified conditions that the sequences of tasks need to satisfy. In addition, for simplicity purposes, the approach assumes a normal distribution of the calculated probabilities. However, other probability distributions can, in principle, be used to represent other types of risk assessment.

5.3 Future work

Future work includes publishing scientific articles about the proposed approach, additional case studies, explicit trading measures, context-aware approaches, software agents, and anomaly discovery methods. The evaluation of the approach can be done by implementing the approach with industrial data and comparing projections and proposed RMPs to actual observations and tasks executed by an existing AC system. Additional case studies would further explain specific details of the approach design necessary to implement it correctly in other domains of knowledge. As well, additional case studies must provide data required to compare proposed approach with existing AC approaches. The sequence of task execution is predefined and enhancement of the approach would be to instead of providing graph of tasks to provide tasks and their pre and post conditions evaluation.

5.3.1 Additional Case Studies

The approach would benefit from having additional case studies. These case studies must include more complex models and scenarios, exist in different knowledge domains, be based on real-life data from industry.

5.3.2 Explicit Risk Trading Measures

Introducing explicit risk trading measures would benefit the RMP construction for both risk analysis and risk mitigation. Although the current FM operates with various QRMs from different risk dimensions and it considers both supply and demand, explicit trading was not implemented with the dynamic ratio of QRM conversion and other tools from both micro- and macroeconomics that could be applied to improve the projections built by the approach.

5.3.3 Context-Aware Models

The approach would benefit from having an explicit context-aware FM that takes into account various attributes of components, processes and the system for which AC is provided such as spatial and temporal. In addition, to improve the projection performed by the FM, the approach needs to monitor the state of the system and compare this state with the projection. If this projection at some point starts to differentiate significantly from the state of the system, the approach should be able to reevaluate the ongoing risks based on context and adjust some risk-related values or even select different algorithms for risk evaluation.

5.3.4 Introducing Software Agents and Anomaly Discovery Methods

The approach could benefit from introducing software agents, including a goal-oriented FM that would allow components to make more rational decisions about what type of strategy to use and how to modify and evaluate the provided decisions. Additionally, introducing these agents to the approach would allow the approach to become compliant with predefined goals, simplify the analysis of its performance and make real-time adjustments that would benefit different aspects of the AC system functionality, including manageability and reliability. Finally, the approach would benefit from an anomaly discovery [127] that could be integrated into it. This anomaly discovery would allow the approach to detect components that are harmful earlier in the RMP. The anomalies can include statistical anomalies preventing accurate projection, anomalies in components behavior and other types of anomalies disrupting function of AC.

Bibliography

1. Salehie, M., Pasquale, L., Omoronyia, I., Ali, R., & Nuseibeh, B. Requirements-driven adaptive security: Protecting variable assets at runtime. Requirements Engineering Conference (RE), 2012 20th IEEE International, pp. 111-120, 2012.
2. Zhao, Z., Hu, H., Ahn, G. J., & Wu, R. Risk-aware mitigation for MANET routing attacks. Dependable and Secure Computing, IEEE Transactions on, 9(2), pp. 250-260, 2012.
3. Hsu, I. C. Extensible access control markup language integrated with Semantic Web technologies. Information Sciences, 238, pp. 33-51, 2013.
4. Borders, K., & Prakash, A. Quantifying information leaks in outbound web traffic. In Security and Privacy, 30th IEEE Symposium on, pp. 129-140, 2009.
5. Milne, G. R., Rohm, A. J., & Bahl, S. Consumers' protection of online privacy and identity. Journal of Consumer Affairs, 38(2), pp. 217-232, 2004.
6. Barkley, J. Workflow management employing role-based access control. U.S. Patent No. 6,088,679. Washington, DC: U.S. Patent and Trademark Office, 2000.
7. De La Huerga, C. Security badge for automated access control and secure data gathering. U.S. Patent No. 5,960,085, Washington, DC: U.S. Patent and Trademark Office, 1999.
8. Shen, H., & Dewan, P. Access control for collaborative environments. In Proceedings of the 1992 ACM conference on Computer-supported cooperative work, pp. 51-58, ACM, 1992.
9. Sandhu, R. The future of access control: Attributes, automation and adaptation. IRI, 2013.
10. Miettinen, M., Heuser, S., Kronz, W., Sadeghi, A. R., & Asokan, N. ConXsense–Context Profiling and Classification for Context-Aware Access Control, ASIACCS, 2014.
11. Mirza, N. A. S., Abbas, H., Khan, F., & Al Muhtadi, J. Anticipating Advanced Persistent Threat (APT) countermeasures using collaborative security mechanisms. In Biometrics and Security Technologies (ISBAST), pp. 129-132, 2014.
12. Kral, D., & Urbanek, J. F. Terror Threats Life Cycles Controlling Using Crisis Management during Environmental Metamorphoses, IERI Procedia, pp. 141-147, 2014.
13. Shield, J., Hopkins, B., Beaumont, M., & North. Hardware Trojans–A Systemic Threat. Australasian Information Security Conference Vol. 27, p. 30, 2015.
14. Brooks, T. T., Caicedo, C., & Park, J. S. Security vulnerability analysis in virtualized computing environments. International Journal of Intelligent Computing Research, 3(1/2), pp. 277-291, 2012.
15. Umrao, S., Kaur, M., & Gupta, G. K. Vulnerability assessment and penetration testing. International Journal of Computer & Communication Technology, 3(6-8), pp. 71-74, 2012.

16. Yao, G., Guan, Q., & Ni, K. Test model for security vulnerability in web controls based on fuzzing. Journal of Software, 7(4), pp. 773-778, 2012.
17. Rahbari, H., Krunz, M., & Lazos, L. Security vulnerability and countermeasures of frequency offset correction in 802.11 a systems. INFOCOM, pp. 1015-1023, 2014.
18. Manoj, R. J., Chandrasekhar, A., Praveena, M. A., & Desai, G. AFTAC: Attribute, Feedback and Time Decay Based Access Control for Web Services. In International Conference on Computing, p. 119, 2012.
19. Hu, V. C., & Kent, K. A. Guidelines for access control system evaluation metrics. US Department of Commerce, National Institute of Standards and Technology, 2012.
20. Zhou, L., Varadharajan, V., & Hitchens, M. Achieving secure role-based access control on encrypted data in cloud storage. Information Forensics and Security, IEEE Transactions on, 8(12), pp. 1947-1960, 2013.
21. Sridhar, S., Hahn, A., & Govindarasu, M. Cyber–physical system security for the electric power grid. Proceedings of the IEEE, 100(1), pp. 210-224, 2012.
22. Suhendra, V., A Survey on Access Control Deployment, Security Technology Communications in Computer and Information Science, volume 259, pp. 11-20, 2011.
23. Abercrombie, R., Sheldon, F., Aldridge, H., Duren, M., Ricci, T., Bertino, E., Kulatunga, A., & Navaratne, U., Secure Cryptographic Key Management System (CKMS) Considerations for Smart Grid Devices, Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research, article 59, 2011.
24. Li, J., Chen, X., Li ,J., Jia, C., Ma, J., & Lou, W., Fine-Grained Access Control System Based on Outsourced Attribute-Based Encryption, Computer Security–ESORICS Lecture Notes in Computer Science, volume 8134, pp. 592-609, 2013.
25. Vijayakumar, H., Jakka, G., Rueda, S., Schiffman, J., & Jaeger, T., Integrity Walls: Finding Attack Surfaces from Mandatory Access Control Policies, Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 75-76, 2012.
26. Jin, X., Krishnan, R., & Sandhu, R., A Unified Attribute-Based Access Control Model Covering DAC, MAC And RBAC, Data and Applications Security and Privacy XXVI, Lecture Notes in Computer Science, volume 7371, pp. 41-55, 2012.
27. Shafiq, B., Vaidya, J. S., Ghafoor, A., & Bertino, E., A Framework for Verification and Optimal Reconfiguration of Event-Driven Role Based Access Control Policies, Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 197-208, 2012.
28. Komlenovic, M., Tripunitara, M., & Zitouni, T., An Empirical Assessment of Approaches to Distributed Enforcement in Role-Based Access Control (RBAC), Proceedings of the First ACM Conference on Data and Application Security and Privacy, pp. 121-132, 2011.
29. Tripunitara, M. V., & Carbunar, B., Efficient Access Enforcement in Distributed Role-Based Access Control (RBAC) Deployments, Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, pp. 155-164, 2009.
30. Kirkpatrick, M. S., Damiani, M. L., & Bertino, E., Prox-RBAC: a Proximity-Based Spatially Aware RBAC, Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 339-348, 2011.

31. Hu, S., Muthusamy, V., Li, G., & Jacobsen, H. A., Transactional Mobility in Distributed Content-Based Publish/Subscribe Systems, Distributed Computing Systems, ICDCS'09, 29th IEEE International Conference on, pp. 101-110, 2009.
32. Bertino, E., & Kirkpatrick, M. S., Location-Based Access Control Systems for Mobile Users: Concepts and Research Directions, Proceedings of the 4th ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS, pp. 49-52, 2011.
33. Marinovic, S., Craven, R., Ma, J., & Dulay, N., Rumpole: a Flexible Break-Glass Access Control Model, Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, pp. 73-82, 2011.
34. Park, J., Sandhu, R., & Cheng Y., A User-Activity-Centric Framework for Access Control in Online Social Networks, Internet Computing, volume 15, issue 5, pp. 62-65, 2011.
35. Karp, A. H., Haury, H., & Davis M. H., From ABAC to ZBAC: the Evolution of Access Control Models, Technical Report HPL-2009-30, Hewlett-Packard Laboratories, pp. 1-21, 2009.
36. Huang, J., Nicol, D. M., Bobba, R., & Huh, J. H., A Framework Integrating Attribute-Based Policies into Role-Based Access Control, Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 187-196, 2012.
37. Schefer-Wenzl, S., & Strembeck, M., Generic Support for RBAC Break-Glass Policies in Process-Aware Information Systems, Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1441-1446, 2013.
38. Ramli, C. D. P. K. Detecting Incompleteness, Conflicting and Unreachability XACML Policies using Answer Set Programming. arXiv preprint arXiv:1503.02732, 2015.
39. Hinrichs, T. L., Martinoia, D., Garrison, W. C., Lee, A. J., Panebianco, A., & Zuck, L. (2013, June). Application-sensitive access control evaluation using parameterized expressiveness. In Computer Security Foundations Symposium (CSF), pp. 145-160, 2013.
40. Garrison III, W. C., Qiao, Y., & Lee, A. J. On the suitability of dissemination-centric access control systems for group-centric sharing. In Proceedings of the 4th ACM conference on Data and application security and privacy, pp. 1-12, 2014.
41. Hinrichs, T. L., Martinoia, D., Garrison III, W. C., Lee, A. J., Panebianco, A., & Zuck, L. Application-Sensitive Access Control Evaluation using Parameterized Expressiveness (Extended Version), Computer Security Foundations Symposium (CSF), IEEE 26th, 2013.
42. MacDermott, A., Shi, Q., Merabti, M., & Kifiyat, K. Considering an elastic scaling model for cloud security. In Internet Technology and Secured Transactions (ICITST), 8th International Conference for, pp. 150-155, 2013.
43. Vimercati, S. D. C. D., Foresti, S., Jajodia, S., Paraboschi, S., Psaila, G., & Samarati, P. Integrating trust management and access control in data-intensive web applications. ACM Transactions on the Web (TWEB), 6(2), p. 6, 2012.
44. Jin, X., Krishnan, R., & Sandhu, R. S. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. DBSec, 12, pp. 41-55, 2012.

45. Fenz, S., Neubauer, T., Accorsi, R., & Koslowski, T. FORISK: Formalizing information security risk and compliance management. In Dependable Systems and Networks Workshop (DSN-W), 43rd Annual IEEE/IFIP Conference on, pp. 1-4, 2013.
46. Hale, M. L.,& Gamble, R. Secagreement: Advancing security risk calculations in cloud services. In Services (SERVICES), IEEE Eighth World Congress on, pp. 133-140, 2012.
47. Yarmand, M. H., Sartipi, K., & Down, D. G. Behavior-based access control for distributed healthcare systems. Journal of Computer Security, 21(1), pp. 1-39, 2013.
48. Raykova, M., Zhao, H., & Bellovin, S. M. (2012). Privacy enhanced access control for outsourced data sharing. In Financial cryptography and data security, pp. 223-238, Springer Berlin Heidelberg, 2012.
49. Lin, H. Y., Kubiatowicz, J., & Tzeng, W. G. A Secure Fine-Grained Access Control Mechanism for Networked Storage Systems. In Software Security and Reliability (SERE), IEEE Sixth International Conference on, pp. 225-234, 2012.
50. Futcher, L., & von Solms, R. A Risk-Based Approach to Formalise Information Security Requirements for Software Development. In Information Assurance and Security Education and Training, pp. 257-264, Springer Berlin Heidelberg, 2013.
51. Leontie, E., Bloom, G., Narahari, B., & Simha, R. No principal too small: Memory access control for fine-grained protection domains. In Digital System Design (DSD), 15th Euromicro Conference on, pp. 163-170, 2012.
52. Krautsevich, L., Lazouski, A., Martinelli, F., Mori, P., & Yautsiukhin, A. Integration of quantitative methods for risk evaluation within usage control policies. In Computer Communications and Networks (ICCCN), 2013 22nd International Conference on, pp. 1-8, 2013.
53. Aich, A., & Sen, A. Study on Cloud Security Risk and Remedy. International Journal of Grid & Distributed Computing, 8(2), 2015.
54. Shafiq, B., Vaidya, J. S., Ghafoor, A., & Bertino, E., A Framework for Verification and Optimal Reconfiguration of Event-Driven Role Based Access Control Policies, Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 197-208, 2012.
55. Lagutin, D., Visala, K., Zahemszky, A., Burbridge, T., & Marias, G. F., Roles and Security in a Publish/Subscribe Network Architecture, IEEE Symposium on Computers and Communications (ISCC), pp. 68-74, 2010.
56. Singh, A., Role Based Trust Management Security Policy Analysis, International Journal of Engineering Research and Applications (IJERA), volume 2, issue 6, pp. 126-155, 2012.
57. Alalfi, M. H., Cordy, J.R., & Dean, T. R., Automated Verification of Role-Based Access Control Security Models Recovered from Dynamic Web Applications, 14th IEEE International Symposium on Web Systems Evolution (WSE), pp. 1-10, 2012.
58. Ullah, S., Xuefeng, Z., & Feng, Z., TCloud: A Dynamic Framework and Policies for Access Control across Multiple Domains in Cloud Computing, International Journal of Computer Applications, volume 62, number 2, pp. 1-7, 2013.
59. Chen, X., Chen, C., Tao, Y., & Hu, J. A Cloud Security Assessment System Based on Classifying and Grading. Cloud Computing, IEEE, 2(2), pp. 58-67, 2015.

60. Bugiel, S., Heuser, S., & Sadeghi, A. R. Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. In Usenix security, pp. 131-146, 2013.
61. Vijayakumar, H., Jakka, G., Rueda, S., Schiffman, J., & Jaeger, T. Integrity walls: Finding attack surfaces from mandatory access control policies. In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 75-76, 2012.
62. Dan, N., Hua-Ji, S., Yuan, C., & Jia-Hu, G. Attribute based access control (abac)-based cross-domain access control in service-oriented architecture (soa). In Computer Science & Service System (CSSS), pp. 1405-1408, 2012.
63. Liang, F., Guo, H., Yi, S., & Ma, S. A multiple-policy supported attribute-based access control architecture within large-scale device collaboration systems. Journal of Networks, 7(3), pp. 524-531, 2012.
64. Crampton, J., Practical and Efficient Cryptographic Enforcement of Interval-Based Access Control Policies, ACM Transactions on Information and System Security (TISSEC), volume 14, issue 1, article 14, 2011.
65. Zhu, Y., Hu, H., Ahn, G. J., Huang, D., & Wang, S. Towards temporal access control in cloud computing. In INFOCOM, pp. 2576-2580, 2012.
66. Steel, C., & Nagappan, R. Core Security Patterns: Best Practices and Strategies for J2EE", Web Services, and Identity Management. Pearson Education India, 2006.
67. Stepien, B., Felty, A.,& Matwin, S. Challenges of Composing XACML Policies. In Availability, Reliability and Security (ARES), Ninth International Conference on, pp. 234-241, 2014.
68. Ardagna, C. A., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., Samarati, P., & Verdicchio, M. Expressive and deployable access control in open Web service applications. Services Computing, IEEE Transactions on, 4(2), pp. 96-109, 2011.
69. Lazouski, A., Martinelli, F., & Mori, P. A prototype for enforcing usage control policies based on XACML, pp. 79-92, Springer Berlin Heidelberg, 2012.
70. El Kateb, D., ElRakaiby, Y., Mouelhi, T., Rubab, I., & Le Traon, Y. Towards a Full Support of Obligations In XACML. In Risks and Security of Internet and Systems, pp. 213-221, Springer International Publishing, 2014.
71. Ulltveit-Moe, N., & Oleshchuk, V. Decision-cache based XACML authorisation and anonymisation for XML documents. Computer Standards & Interfaces, 34(6), pp. 527-534, 2012.
72. Ayed, D., Bichsel, P., Camenisch, J., & den Hartog, J. Integration of Data-Minimising Authentication into Authorisation Systems. In Trust and Trustworthy Computing, pp. 179-187, Springer International Publishing, 2014.
73. Huang, D. H., & Yang, Y. Q. Role-based risk adaptive access control model. In Applied Mechanics and Materials, Vol. 416, pp. 1516-1521, 2013.
74. Khambhammettu, H., Boulares, S., Adi, K., & Logrippo, L. A Framework for Threat Assessment in Access Control Systems. In Information Security and Privacy Research, pp. 187-198, 2012.
75. Bijon, K. Z., Krishnan, R., & Sandhu, R. Risk-aware RBAC sessions. In Information Systems Security, pp. 59-74, 2012.

76. dos Santos, D. R., Merkle Westphall, C., & Becker Westphall, C. A dynamic risk-based access control architecture for cloud computing. In Network Operations and Management Symposium (NOMS), pp. 1-9, 2014.
77. Molloy, I., Dickens, L., Morisset, C., Cheng, P. C., Lobo, J., & Russo, A., Risk-Based Security Decisions under Uncertainty, Proceedings of the Second ACM Conference on Data and Application Security and Privacy, pp. 157-168, 2012.
78. Allen, L., & Bali, T. G., Cyclicality in Catastrophic and Operational Risk Measurements, Journal of Banking & Finance, volume 31, issue 4, pp. 1191–1235, 2007.
79. Molloy, I, Cheng, P., & Rohatgi, P., Trading in Risk: Using Markets to Improve Access Control, Proceedings of the 2008 Workshop on New Security Paradigms, pp.107-125, 2009.
80. Sahinoglu, M., Stockton, S., Morton, S., Vasudev, P., & Eryilmaz, M. Metrics to Assess and Manage Software Application Security Risk. In Proceedings of the International Conference on Security and Management (SAM), The Steering Committee of The World Congress in Computer Science, (WorldComp), pp. 1, 2014.
81. Allen, L., & Saunders, A., Incorporating Systemic Influences into Risk Measurements: A Survey of the Literature, Journal of Financial Services Research, volume 26, issue 2, pp.161-191, 2004.
82. Han, W., Shen, C., Yin, Y., Gu, Y., & Chen, C., Poster: Using Quantified Risk and Benefit to Strengthen the Security of Information Sharing, Proceedings of the 18th ACM Conference on Computer and Communications Security, pp.781-784, 2011.
83. Molloy, I., Dickens, L., Morisset, C., Cheng, P. C., Lobo, J., & Russo, A., Risk-Based Security Decisions under Uncertainty, Proceedings of the Second ACM Conference on Data and Application Security and Privacy, pp. 157-168, 2012.
84. Kirkpatrick, M. S., Damiani, M. L., & Bertino, E., Prox-RBAC: a Proximity-Based Spatially Aware RBAC, Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 339-348, 2011.
85. Fugini, M., Teimourikia, M., & Hadjichristofi, G. A web-based cooperative tool for risk management with adaptive security. Future Generation Computer Systems, 2015.
86. Molloy, I, Cheng, P., & Rohatgi, P., Trading in Risk: Using Markets to Improve Access Control, Proceedings of the 2008 Workshop on New Security Paradigms, pp. 107-125, 2009.
87. Han, W., Sun, C., Shen, C., Lei, C., & Shen, S. Dynamic combination of authentication factors based on quantified risk and benefit. Security and Communication Networks, 7(2), pp. 385-396, 2014.
88. Allen, L., & Saunders, A., Incorporating Systemic Influences into Risk Measurements: A Survey of the Literature, Journal of Financial Services Research, volume 26, issue 2, pp. 161-191, 2004.
89. Västfjäll, D., Peters, E., & Slovic, P. The affect heuristic, mortality salience, and risk: Domain-specific effects of a natural disaster on risk-benefit perception. Scandinavian journal of psychology, 55(6), pp. 527-532, 2014.
90. Backes, M., Kopf B., & Rybalchenko, A., Automatic Discovery and Quantification of Information Leaks, 30th IEEE Symposium on Security and Privacy, pp 141-153, 2009.

91. Hart, M., Manadhata, P., & Johnson, R., Text Classification for Data Loss Prevention, Proceedings of the 11th international conference on Privacy enhancing technologies (PETS), pp. 18-37, 2011.
92. Shameli-Sendi, A., Shajari, M., Hassanabadi, M., Jabbarifar, M., & Dagenais, M. Fuzzy multi-criteria decision-making for information security risk assessment. The Open Cybernetics & Systemics Journal, 6(1), 2012.
93. Molloy, I., Dickens, L., Morisset, C., Cheng, P. C., Lobo, J., & Russo, A. Risk-based security decisions under uncertainty. In Proceedings of the second ACM conference on Data and Application Security and Privacy, pp. 157-168, 2012.
94. Dionne, G. Risk management: History, definition, and critique. Risk Management and Insurance Review, 16(2), pp. 147-166, 2013.
95. ISO 31000:2009 - Principles and Guidelines on Implementation. International Organization for Standardization, 2009.
96. ISO/IEC 31010:2009 - Risk Management - Risk Assessment Techniques. International Organization for Standardization, 2009.
97. ISO/IEC 73:2009 - Risk management—Vocabulary. International Organization for Standardization, 2009.
98. Metzger, A., Leitner, P., Ivanovic, D., Schmieders, E., Franklin, R., Carro, M., & Pohl, K. Comparing and combining predictive business process monitoring techniques. Systems, Man, and Cybernetics: Systems, IEEE Transactions on, 45(2), pp. 276-290, 2015.
99. Catalao, J. P. S., Pousinho, H. M. I., & Contreras, J. Optimal hydro scheduling and offering strategies considering price uncertainty and risk management. Energy, 37(1), pp. 237-244, 2012.
100. Wieland, A., & Wallenburg, C. M. Dealing with supply chain risks: Linking risk management practices and strategies to performance. International Journal of Physical Distribution & Logistics Management, 42(10), pp. 887-905, 2012.
101. Bolton, P., Chen, H., & Wang, N. Market timing, investment, and risk management. Journal of Financial Economics, 109(1), pp. 40-62, 2013.
102. Rockafellar, R. T., & Uryasev, S. The fundamental risk quadrangle in risk management, optimization and statistical estimation. Surveys in Operations Research and Management Science, 18(1), pp. 33-53, 2013.
103. Morin, J. H., Aubert, J., & Gateau, B. Towards cloud computing SLA risk management: issues and challenges. In System Science (HICSS), 45th Hawaii International Conference on pp. 5509-5514, 2012.
104. Grace, M. F., Leverty, J. T., Phillips, R. D.,& Shimpi, P. The Value of Investing in Enterprise Risk Management. Journal of Risk and Insurance, 82(2), pp. 289-316, 2015.
105. Ayyub, B. M. Risk analysis in engineering and economics, CRC Press, 2014.
106. Melnikov, A. Risk analysis in finance and insurance, CRC Press, 2011.
107. Broder, J. F., & Tucker, G. Risk analysis and the security survey. Elsevier, 2011.
108. ISO/DIS 31000 Risk management — Principles and guidelines on implementation. International Organization for Standardization, 2009.
109. NIST SP 800-30 Rev. 1 Guide for Conducting Risk Assessments, 2012.

110. Feng, N., & Xie, J. A Bayesian networks-based security risk analysis model for information systems integrating the observed cases with expert experience. Scientific Research and Essays, 7(10), pp. 1103-1112, 2012.
111. Poolsappasit, N., Dewri, R., & Ray, I. Dynamic security risk management using bayesian attack graphs. Dependable and Secure Computing, IEEE Transactions on, 9(1), pp. 61-74, 2012.
112. Cailliau, A., & Van Lamsweerde, A. A probabilistic framework for goal-oriented risk analysis. In Requirements Engineering Conference (RE), 2012 20th IEEE International, pp. 201-210, 2012.
113. Sadgrove, M. K. The complete guide to business risk management. Ashgate Publishing, Ltd, 2015.
114. Plon, S. E., Cooper, H. P., Parks, B., Dhar, S. U., Kelly, P. A., Weinberg, A. D., ... & Hilsenbeck, S. Genetic testing and cancer risk management recommendations by physicians for at-risk relatives.Genetics in Medicine, 13(2), pp. 148-154, 2011.
115. Bandaly, D., Satir, A., Kahyaoglu, Y., & Shanker, L. Supply chain risk management–I: Conceptualization, framework and planning process. Risk Management, 14(4), pp. 249-271, 2012.
116. Brucker, A. D., Hang, I., Lückemeyer, G., & Ruparel, R. SecureBPMN: Modeling and enforcing access control requirements in business processes. In Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 123-126, 2012.
117. Cognini, R., Falcioni, D., Polini, A., Polzonetti, A., & Re, B. HawkEye: a tool for collaborative business process modelling and verification. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 785-786, 2013.
118. Küster, T., Lützenberger, M., Heßler, A., & Hirsch, B. Integrating process modelling into multi-agent system engineering. Multiagent and Grid Systems, 8(1), pp. 105-124, 2012.
119. Rima, A., & Vasilecas, O. Business Process Simulation: Requirements for Business and Resource Models. Science–Future of Lithuania/Mokslas–Lietuvos Ateitis, 7(3), pp. 291-295, 2015.
120. Cherfi, S. S. S., Ayad, S., & Comyn-Wattiau, I. Aligning business process models and domain knowledge: a meta-modeling approach. In Advances in Databases and Information Systems, pp. 45-56, 2013.
121. Aguilar-Saven, R. S. Business process modelling: Review and framework. International Journal of production economics, 90(2), pp. 129-149, 2004.
122. Behnam, S. A., & Amyot, D. (2013). Evolution mechanisms for goal-driven pattern families used in business process modelling. International Journal of Electronic Business, 10(3), pp. 254-291, 2013.
123. del–Río–Ortega, A., Gutiérrez, A. M., Durán, A., Resinas, M., & Ruiz–Cortés, A. Modelling Service Level Agreements for Business Process Outsourcing Services. In Advanced Information Systems Engineering, pp. 485-500, 2015.
124. Dumas, M., La Rosa, M., Mendling, J., Mäesalu, R., Reijers, H. A., & Semenenko, N. Understanding business process models: the costs and benefits of structuredness. In Advanced Information Systems Engineering, pp. 31-46, 2012.

125. Solomon, A., & Litoiu, M. Business process performance prediction on a tracked simulation model. In Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems, pp. 50-56, 2011.
126. Savinov, S., Alencar, P. A Risk Management Approach for Distributed Event-Based Systems, Technical report, University of Waterloo, pp.1-40, 2014.
127. Fitzgerald, W. M., Turkmen, F., Foley, S. N., & O'Sullivan, B. Anomaly analysis for physical access control security configuration. In Risk and Security of Internet and Systems (CRiSIS), 7th International Conference on, pp. 1-8, 2012.

Appendices

Appendix A - Source Code of the Classes Implementing the Approach

This appendix consists of the source code developed to automate the proposed approach for the software development case study.

A.1 Main.java

```
package CA.engine;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Logical.DataSource;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.*;
import CA.Entities.Logical.RecordsCollection;
import CA.Entities.Physical.Location;
import CA.Entities.Role.DeveloperRole;
import CA.Entities.Software.*;
import CA.NotModel.RiskCollection;
import CA.Processes.Focus;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.LinkedList;


/**
 * Created by root on 08/04/2015.
 */
public class Main {

private static final Logger logger = LogManager.getLogger(Main.class);

public static Location internet;
public static Location serverRoom;
public static Location officeRoom;
public static Location roomOutsideOffice;
public static CompUnit userComputer;
public static CompUnit internetCU;
public static CompUnit notebookInExternalLocation;
```

```
public static CompUnit hardwareFirewall;
public static CompUnit hardwareRouter;
public static CompUnit webServer;

public static CompUnit devMachine1;
public static CompUnit devMachine2;
public static CompUnit buildServer;
public static CompUnit databaseServer;
public static CompUnit externalMailServer;
public static WebServer logicalWebServer;
public static Browser logicalUserBrowser;
public static Firewall logicalFirewall;
public static MailServer logicalExternalMailServer;
public static FileWithRecords externalMailServerPasswordFile;
public static FileWithRecords sourceCodeServerPasswordFile;
public static Record projectTask;
public static FTPServer ftpServerOnBuildServer;
public static SourceCodeServer sourceCodeServer;
public static FileWithRecords projectSourceCodeFileA;
public static FileWithRecords projectSourceCodeFileB;
public static FileWithRecords projectSourceCodeFileC;
public static FileWithRecords projectSourceCodeFileD;
public static StringRecord docA;
public static StringRecord headerA;
public static StringRecord methodA;
public static StringRecord docB;
public static StringRecord headerB;
public static StringRecord methodB;
public static StringRecord docC;
public static StringRecord headerC;
public static StringRecord methodC;
public static StringRecord docD;
public static StringRecord headerD;
public static StringRecord methodD;

public static StringRecord methodANew;
public static StringRecord headerDNew;
public static StringRecord methodDNew;

public static Focus developerFocus;

public DeveloperRole dr;

public void prepareEnvironment(){

        Record.globalList.clear();
docA = new StringRecord("This method is responsible for creating database
structure","docA");
headerA = new StringRecord("CREATE PROCEDURE createTables @City
varchar(30)","headerA");
methodA = new StringRecord("CREATE TABLE new_tbl SELECT * FROM
orig_tbl","methodA");
docB = new StringRecord("This method is responsible for populating database
structure","docB");
headerB = new StringRecord("CREATE PROCEDURE createTables @City
varchar(30)","headerB");
methodB = new StringRecord("CREATE TABLE new_tbl SELECT * FROM
orig_tbl","methodB");
docC = new StringRecord("This method is responsible for running server side
```

```
","docC");
headerC = new StringRecord("int main()","headerC");
methodC = new StringRecord("cout <<\"Hello World!\"<<  endl;","methodC");
docD = new StringRecord("This method is responsible for starting client
application","docD");
headerD = new StringRecord("public static void main(String[] args)","headerD");
methodD = new StringRecord("System.exit(0)","methodD");

methodANew = null;
headerDNew = null;
methodDNew = null;



internet = new Location("Internet");
serverRoom = new Location("ServerRoom");
officeRoom = new Location("OfficeRoom");
roomOutsideOffice = new Location("HotelRoom");

userComputer = new CompUnit("UserComputer");
internetCU = new CompUnit("Internet",internet);
notebookInExternalLocation = new
CompUnit("NotebookOfDeveloper",roomOutsideOffice);

        Listener listener = new Listener(internetCU.getOs()) {
public void processMessage(Message m){
                LinkedList<Record> records = new LinkedList<>();
for(String s:m.getMessageParameters().getKeySet()){
                    Object obj = m.getParameter(s);
if (obj instanceof Record)
                        records.add((Record) obj);
if (obj instanceof RecordsCollection){
                        RecordsCollection rc = (RecordsCollection) obj;
for(String n:rc.getRecordNames()){
                            records.add(rc.getRecord(n));
                        }
                    }

                }
for(Record r:records){
if (r==null) continue;
if (r.getSecurityValue()==0) continue;

RiskCollection.addSecurityRisk(0.4d,ActionsQueue.currentQueue.getCurrentTime(),r)
;
                }

}
        };
internetCU.getOs().addChild(listener,"riskListener");



hardwareFirewall = new CompUnit("HardwareFirewall",serverRoom);
hardwareRouter = new CompUnit("HardwareRouter",serverRoom);
webServer = new CompUnit("WebServer",serverRoom);

devMachine1 = new CompUnit("DevMachine1",officeRoom);
devMachine2 = new CompUnit("DevMachine2",officeRoom);
```

```
buildServer = new CompUnit("BuildServer",serverRoom);
databaseServer = new CompUnit("DatabaseServer",serverRoom);
externalMailServer = new CompUnit("ExternalMailServer");
logicalWebServer = null;
logicalUserBrowser = new Browser(userComputer.getOs());
logicalFirewall = new Firewall(hardwareFirewall.getOs());
projectTask = new StringRecord("Create new database table with
name:test","projectTask");
projectTask.setSecurityValue(0.2d);

externalMailServerPasswordFile = new
FileWithRecords("c:\\passwords.txt",externalMailServer);
externalMailServerPasswordFile.addRecord(new
StringRecord("password","developer1passwordAtExternalMailServer"),"developer1");
logicalWebServer = new
WebServer(webServer.getOs(),externalMailServerPasswordFile);
logicalExternalMailServer = new
MailServer(externalMailServer.getOs(),externalMailServerPasswordFile);

sourceCodeServerPasswordFile = new
FileWithRecords("c:\\sourcepasswords.txt",buildServer);
sourceCodeServerPasswordFile.addRecord(new
StringRecord("anotherpassword","developer1PasswordAtSourceCodeServer"),"developer
1");
sourceCodeServer = new
SourceCodeServer(buildServer.getOs(),sourceCodeServerPasswordFile);



projectSourceCodeFileA = new
FileWithRecords("c:\\create_database.sql",buildServer);
docA.setSecurityValue(0.7d);
projectSourceCodeFileA.addRecord(docA, "DocA1");
projectSourceCodeFileA.addRecord(headerA,"HeaderA1");
methodA.setSecurityValue(1d);
projectSourceCodeFileA.addRecord(methodA,"MethodA1");

sourceCodeServer.addRecords(projectSourceCodeFileA);

projectSourceCodeFileB = new
FileWithRecords("c:\\populate_data.sql",buildServer);
docB.setSecurityValue(0.5d);
projectSourceCodeFileB.addRecord(docB,"DocB1");
projectSourceCodeFileB.addRecord(headerB,"HeaderB1");
methodB.setSecurityValue(2d);
projectSourceCodeFileB.addRecord(methodB,"MethodB1");
sourceCodeServer.addRecords(projectSourceCodeFileB);

projectSourceCodeFileC = new FileWithRecords("c:\\run_program.cpp",buildServer);
projectSourceCodeFileC.addRecord(docC,"DocC1");
projectSourceCodeFileC.addRecord(headerC,"HeaderC1");
methodC.setSecurityValue(0.05d);
projectSourceCodeFileC.addRecord(methodC,"MethodC1");
sourceCodeServer.addRecords(projectSourceCodeFileC);

projectSourceCodeFileD = new
FileWithRecords("c:\\mobile_application.java",buildServer);
projectSourceCodeFileD.addRecord(docD,"DocD1");
projectSourceCodeFileD.addRecord(headerD,"HeaderD1");
projectSourceCodeFileD.addRecord(methodD,"MethodD1");
```

```
sourceCodeServer.addRecords(projectSourceCodeFileD);

        EmailRecord emailRecord = new EmailRecord("emailFromProjectManager");
        emailRecord.addAttachment(projectTask);

        FileWithRecords fwr = new FileWithRecords("c:\\task.txt",buildServer);
        fwr.addRecord(projectTask,"Project task");
ftpServerOnBuildServer = new FTPServer(buildServer.getOs(),null);

logicalExternalMailServer.addRecord("John Doe","projectTask",emailRecord);

createTestNetwork();
//        DeveloperRole dr = new DeveloperRole(officeRoom);
dr = new DeveloperRole(roomOutsideOffice);
        UserRecord ur = new UserRecord("John Doe","mypassword","user");
        ur = new UserRecord("John Doe","mypassword","user");
        DataSource ds = new
DataSource(DataSource.TYPE_EMAIL_SERVER,"Developer1ExternalMailAccount",externalM
ailServer, ur);
dr.addDataSource(ds,"externalMailAccount");
externalMailServerPasswordFile.addRecord(new
UserRecordClass(ur,"Developer1UserRecordAtExternalMailServer"),ur.getUsername());


        UserRecord userRecord = new UserRecord("Developer1", "newpassword",
"developer");
        DataSource dsNew = new
DataSource(DataSource.TYPE_VERSION_CONTROL_SERVER,"Developer1VersionControlAccoun
t",
sourceCodeServer.getParent().getRunningOn(), userRecord);
dr.addDataSource(dsNew,"versionControlServerAccount");
sourceCodeServerPasswordFile.addRecord(new
UserRecordClass(userRecord,"Developer1UserRecordAtSourceCodeServer"),userRecord.g
etUsername());

    }


public void startScenario(){
developerFocus = new Focus(dr.processTree);
        ActionsQueue.currentQueue.start(developerFocus);
        ActionsQueue.currentQueue.progress();
    }

public static void createTestNetwork(){
userComputer.connect(internetCU);
externalMailServer.connect(internetCU);
internetCU.connect(hardwareFirewall);
hardwareFirewall.connect(hardwareRouter);
hardwareRouter.connect(webServer);
hardwareRouter.connect(devMachine1);
hardwareRouter.connect(devMachine2);
hardwareRouter.connect(buildServer);
hardwareRouter.connect(databaseServer);
notebookInExternalLocation.connect(internetCU);

    }

}
```

A.2 ActionsQueue.java

```
package CA.engine;

import CA.NotModel.Risk;
import CA.NotModel.RiskCollection;
import org.apache.logging.log4j.LogManager;

import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import static CA.lib.Lib.*;

/**
 * Created by root on 08/04/2015.
 */
public class ActionsQueue {


private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(ActionsQueue.class);
public static ActionsQueue currentQueue = new ActionsQueue();

public boolean terminated = false;
    Long currentTime = 0l;//1 tick=1ms
public Map<Long, LinkedList<ExecutorService>>schedule;

public ActionsQueue() {
schedule = Collections.synchronizedSortedMap(new TreeMap<Long,
LinkedList<ExecutorService>>());
    }

public Future<?> start(Runnable r) {
        ExecutorService executor = Executors.newSingleThreadExecutor();
final Future f = executor.submit(r);
        Thread t = new Thread("error watching thread"){
public void run(){
try {
f.get();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (ExecutionException e) {
                    e.printStackTrace();
                }
            }
        };
        t.start();
return f;
}

/**
     *
     * @param scheduledTime time at which even is scheduled
     * @param monitor procedure will put in this list executor responsible for
processing request,
     *                it is waiting for notification. If any object is added to
it, i.e. Message then executor
     *                will not be invoked
```

```
     */
public void addNewEventWithInterruptionAtFirstSchedule(long scheduledTime,
LinkedList<Object> monitor) {
        ExecutorService executor = Executors.newSingleThreadExecutor();

        LinkedList<ExecutorService> scheduled = schedule.get(scheduledTime);
if (scheduled == null) {
            scheduled = new LinkedList<ExecutorService>();
            scheduled.add(executor);
schedule.put(scheduledTime, scheduled);
        } else
scheduled.add(executor);

        monitor.add(executor);//monitor holds executor

while (getCurrentTime() != scheduledTime && monitor.size() <= 1)//waiting to be
either on time  if no
            // interruptions occurred or to be interrupted
synchronized (executor) {
try {
logger.log(VERBOSE, "waiting on :" + executor);
                    executor.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
if (monitor.size()>1){
logger.log(VERBOSE, "waiting on :" + executor);
            removeEvent(scheduledTime,executor);
synchronized (monitor){
                monitor.notifyAll();
            }
        }
    }


/**
     *
     * @param scheduledTime time at which even is scheduled
     * @param monitor procedure will put in this list executor responsible for
processing request,
     *                it is waiting for notification. If any object is added to
it, i.e. Message then executor
     *                will not be invoked
     */
public void addNewEventWithInterruption(long scheduledTime, LinkedList<Object>
monitor) {
        ExecutorService executor = Executors.newSingleThreadExecutor();

        LinkedList<ExecutorService> scheduled = schedule.get(scheduledTime);
if (scheduled == null) {
            scheduled = new LinkedList<ExecutorService>();
            scheduled.add(executor);
schedule.put(scheduledTime, scheduled);
        } else
scheduled.add(executor);

        monitor.add(executor);//monitor holds executor

while (getCurrentTime() != scheduledTime && monitor.size() <= 1)//waiting to be
```

```
either on time  if no
                                                                        //
interruptions occurred or to be interrupted
synchronized (executor) {
try {
logger.log(VERBOSE, "waiting on :" + executor);
                    executor.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
if (monitor.size()>1){
logger.log(VERBOSE, "waiting on :" + executor);
            removeEvent(scheduledTime,executor);
synchronized (monitor){
                monitor.notifyAll();
            }
        }
    }

public boolean removeEvent(Long scheduledTime, ExecutorService executor){
if (!schedule.containsKey(scheduledTime)) return false;
        LinkedList<ExecutorService> list = schedule.get(scheduledTime);
        list.remove(executor);
synchronized (executor){
            executor.notify();//is it needed?
}
if (list.size()==0) schedule.remove(scheduledTime);
return true;
    }

public void addNewEvent(long scheduledTime) {

        ExecutorService executor = Executors.newSingleThreadExecutor();

        LinkedList<ExecutorService> scheduled = schedule.get(scheduledTime);
if (scheduled == null) {
            scheduled = new LinkedList<ExecutorService>();
            scheduled.add(executor);
schedule.put(scheduledTime, scheduled);
logger.log(VERBOSE,this+":no");
        } else {
            scheduled.add(executor);
logger.log(VERBOSE, ":yes");
        }

boolean waiting = false;
while (getCurrentTime() != scheduledTime)
synchronized (executor) {
try {
logger.log(VERBOSE, "waiting on :" + executor);
                    waiting = true;
                    executor.wait();

                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
if (waiting)
logger.trace(this + ":done waiting");
```

```
    }


public void progress() {

try {
            Thread.sleep(40);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        Set<Long> timeScheduled = schedule.keySet();

        Iterator<Long> it = timeScheduled.iterator();
long time = 0;

if (it.hasNext()) {
do {
                time = it.next();
if (time >= currentTime) break;
logger.error("skipping events, they were scheduled in the past");
            } while (it.hasNext());


            LinkedList<ExecutorService> threadsToActivate =
schedule.remove(time);
currentTime = time;
logger.debug("New time:" + currentTime);
for (ExecutorService t : threadsToActivate) {
synchronized (t) {
logger.log(VERBOSE, "notifying waiting on :" + t);
                    t.notify();
                }
            }
if (ActionsQueue.currentQueue.getCurrentTime()>10000){
logger.trace("security:" +
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECURITY));
logger.trace("productivity:" +
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_PRODUCTIVITY));
            }
        }
try {
            Thread.sleep(5);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }


if (!terminated)
            progress();
    }

public long getCurrentTime() {
return currentTime;
    }


public void printEventQueue(){
        Set<Long> timeScheduled = schedule.keySet();
if (timeScheduled.isEmpty()) {
```

```
logger.info("no events");
return;
        }
        String log = "Events we have:";
for (Long l : timeScheduled)
            log+=":" + l+"\n";

logger.info(log);
    }
}
```

A.3 Random.java

```
packageCA.engine;

import java.security.SecureRandom;

/**
 * Created by root on 12/04/2015.
 */
public class Random {
static Random defaultInstance = new Random();

//    SecureRandom sr = new SecureRandom("12.04.2015:2:37".getBytes());//working
case
SecureRandom sr = new SecureRandom("12.04.2015:2:379".getBytes());

public static float getNext(){
return defaultInstance.sr.nextFloat();
    }

public static int getNext(int val){
return defaultInstance.sr.nextInt(val);
    }

public static String getNewID(){
        String s = "";
for(int i=0;i<24;i++)
        s+=defaultInstance.sr.nextInt(10);
return s;
    }

public static void reInit(){
defaultInstance = new Random();
    }

}
```

A.4 AccessControlDecisionMaker.java

```
packageCA.Entities.AccessControl;

import CA.Entities.Logical.Message;

/**
 * Created by root on 11/06/2015.
 */
public interface AccessControlDecisionMaker {
public Boolean allowAccess(Message m);
}
```

A.5 FirewallDecisionMaker

```
packageCA.Entities.AccessControl;

import CA.Entities.Logical.Message;
```

```
import java.util.LinkedList;

/**
 * Created by root on 11/06/2015.
 */
public class FirewallDecisionMakerimplements AccessControlDecisionMaker {

    LinkedList<FirewallPolicy>policies = new LinkedList<>();
@Override
public Boolean allowAccess(Message m) {
        LinkedList<Boolean> result = new LinkedList<>();
for(FirewallPolicy fp:policies)
            result.add(fp.allowAccess(m));
return makeDecision(result);
    }

public void addPolicy(FirewallPolicy fp){
policies.add(fp);
    }

public boolean makeDecision(LinkedList<Boolean> result){
return makeDecisionOneReject(result);
    }

public boolean makeDecisionOneReject(LinkedList<Boolean> result){
for(Boolean b:result)
if (b!=null&&!b) return false;
return true;
    }

public boolean makeDecisionOneAccept(LinkedList<Boolean> result){
for(Boolean b:result)
if (b!=null&&b) return true;
return false;
    }


}
```

A.6 FirewallPolicy.java

```
packageCA.Entities.AccessControl;

import CA.Entities.Logical.Message;

/**
 * Created by root on 11/06/2015.
 */
public abstract class FirewallPolicyimplements AccessControlDecisionMaker{

@Override
public abstract Boolean allowAccess(Message m);
}
```

A.7 ServerLocalAccessControl.java

```
package CA.Entities.AccessControl;

import CA.Entities.Logical.DataSource;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.UserRecord;
import CA.Entities.Logical.Records.UserRecordClass;
import CA.Entities.Software.Server;
import CA.Entities.Software.Session;
import CA.engine.ActionsQueue;
import CA.lib.Pair;

/**
 * Created by root on 26/04/2015.
 */
public class ServerLocalAccessControl implements AccessControlDecisionMaker{
boolean allowNotAuthorizedAccess = false;
    Server parent;
    DataSource passwordData;

public ServerLocalAccessControl(String passwordFileName, Server parent){
this.parent = parent;
passwordData = new DataSource(parent.getParent().getFile(passwordFileName));
    }

public ServerLocalAccessControl(FileWithRecords passwordFile, Server parent){
this.parent = parent;
if (passwordFile==null) {
allowNotAuthorizedAccess = true;
return;
        }
passwordData = new DataSource(passwordFile);
    }

public ServerLocalAccessControl(boolean allowAccessWithoutAuthorization){
allowNotAuthorizedAccess = allowAccessWithoutAuthorization;
    }

public Boolean allowAccess(Message m){
if (allowNotAuthorizedAccess) return true;
if (m.getCommand().equals(Server.authorize)) return true;
```

```
        Session s = parent.getSession(m.getSessionID());
if (s!=null&&s.getTimeSessionExpires()>=
ActionsQueue.currentQueue.getCurrentTime())
return true;
return false;
}

public boolean passwordsMatch(String account, String passwordToTest){
if (passwordData==null){
return false;
        }
        Pair pair = passwordData.getRecord(account,
parent.getParent().getRunningOn());
if (pair.a==null) return false;
UserRecord rr = ((UserRecordClass) pair.a).getValue();
        String recordPassword = rr.getPassword();
return recordPassword.equals(passwordToTest);
    }

public String getAccountRole(String account){
        Pair pair = passwordData.getRecord(account,
parent.getParent().getRunningOn());
if (pair==null||pair.a==null) return null;
        UserRecord rr = ((UserRecordClass) pair.a).getValue();
return rr.getRoleName();
    }

public boolean isAllowNotAuthorizedAccess() {
return allowNotAuthorizedAccess;
    }
}
```

A.8 Workstation.java

```
packageCA.Entities.Configuration;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Software.OperatingSystem;

/**
 * Created by root on 08/04/2015.
 */
public class WorkStationextends CompUnit {
    OperatingSystem os = new OperatingSystem(this);

public WorkStation(String name) {
super(name);
    }
}
```

A.9 CompUnit.java

```
package CA.Entities.Hardware;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.RecordsCollection;
import CA.Entities.Physical.Location;
import CA.Entities.Software.OperatingSystem;
import CA.lib.Entity;

import java.util.LinkedList;

/**
 * Created by root on 08/04/2015.
 */
public class CompUnit extends HardwareEntity implements Entity {
    CPU cpu = new CPU();
    LinkedList<HardDrive>hardDrives = new LinkedList<HardDrive>();

    Location location = null;

public OperatingSystem getOs() {
return os;
    }

    OperatingSystem os;

public LinkedList<HardDrive> getHardDrives() {
return hardDrives;
    }

public NetworkingInterface getSpecificNetworkInterface(String
nameOfCompUnitItConnectsTo){
for(NetworkingInterface ni:networkConnections)
if
(ni.getConnectedTo().getInstalledOn().name.toLowerCase().startsWith(nameOfCompUni
tItConnectsTo.toLowerCase())) return ni;
return null;
    }



public LinkedList<NetworkingInterface> getNetworkConnections() {
```

```
return networkConnections;
    }

    LinkedList<NetworkingInterface>networkConnections = new
LinkedList<NetworkingInterface>();

public void connect(CompUnit newConnection){
        NetworkingInterface ni = new NetworkingInterface(this);
        NetworkingInterface ni2 = new NetworkingInterface(newConnection);
        ni.connectTo(ni2);
    }

public CompUnit(String name,Location location){
this(name);
        setLocation(location);
    }

public CompUnit(String name) {
this.name = name;
os = new OperatingSystem(this);
        setLocation(new Location("Room of:"+name));
        HardDrive hd = new HardDrive("c:",this);
this.addHardDrive(hd);
    }

public String toString(){
return name;
    }

public Location getLocation() {
return location;
    }

public void setLocation(Location location) {
if (this.location!=null){
this.location.removeObjectFromLocation(this);
        }
this.location = location;
        location.addObjectToLocation(this);
    }

public void addHardDrive(HardDrive hd){
hardDrives.add(hd);
    }

public void createFile(String name, RecordsCollection infoToPutInFile){
        FileWithRecords fwr = new FileWithRecords(name, this);
for(String n:infoToPutInFile.getRecordNames()){
            fwr.addRecord(infoToPutInFile.getRecord(n),n);
        }
    }
}
```

A.10 CPU.java

```
packageCA.Entities.Hardware;

import CA.lib.Entity;

/**
```

```
 * Created by root on 08/04/2015.
 */
public class CPU extends HardwareEntity implements Entity {
public float capacity=100;
public float usage;
}
```

A.11 HardDriver.java

```
packageCA.Entities.Hardware;

import CA.Entities.Logical.RecordsCollection;
import CA.Entities.Logical.FileWithRecords;
import CA.lib.Entity;

import java.util.HashMap;
import java.util.Map;

/**
 * Created by root on 08/04/2015.
 */
public class HardDriveextends HardwareEntity implements Entity {
    String driveName;
    CompUnit parent;
    HashMap<String, FileWithRecords>files = new HashMap<String,
FileWithRecords>();

public HardDrive(String driveName,CompUnit parent) {
this.driveName = driveName;
this.parent = parent;
    }

public FileWithRecords getRecords(String fullName){
return files.get(fullName.substring(3));
    }

public void addRecord(RecordsCollection r, String name){
        FileWithRecords fileWithRecords = null;
if (!(r instanceof FileWithRecords)) {
             fileWithRecords = new FileWithRecords(name, parent);
        } else{
            fileWithRecords = (FileWithRecords) r;
        }
files.put(name,fileWithRecords);
    }

public FileWithRecords removeFile(String name) {
return files.remove(name);
    }

public boolean removeReference(String name, RecordsCollection r) {
        RecordsCollection o = files.remove(name);
if (o == null) {
for (Map.Entry<String, FileWithRecords> e : files.entrySet()) {
if (r == e.getValue()) {
files.remove(e.getKey());
return true;
                }
            }
return false;
```

```
        } else {
return true;
        }
    }

public boolean removeReference(RecordsCollection r) {
for (Map.Entry<String, FileWithRecords> e : files.entrySet()) {
if (r == e.getValue()) {
files.remove(e.getKey());
return true;
            }
        }
return false;
    }

public String getDriveName() {
return driveName;
    }

public void setDriveName(String driveName) {
this.driveName = driveName;
    }
}
```

A.12 HardwareEntity.java

```
packageCA.Entities.Hardware;

import CA.lib.Entity;

/**
 * Created by root on 08/04/2015.
 */
public class HardwareEntity implements Entity{
boolean enabled = true;
    String name;
}
```

A.13 NetworkingInterface.java

```
packageCA.Entities.Hardware;

import CA.Entities.Logical.Message;
import CA.lib.Entity;

import java.util.LinkedList;

/**
 * Created by root on 08/04/2015.
 */
public class NetworkingInterface extends HardwareEntity implements Entity {
    NetworkingInterface connectedTo;
    CompUnit installedOn;

public void connectTo(NetworkingInterface ni){
connectedTo = ni;
        ni.connectedTo=this;
    }

public NetworkingInterface(CompUnit installedOn) {
this.installedOn = installedOn;
```

```
        installedOn.networkConnections.add(this);
    }

public LinkedList<NetworkingInterface> buildPath(CompUnit origin, CompUnit
target){
        LinkedList<NetworkingInterface> l  = new
LinkedList<NetworkingInterface>();
return recursiveBuildPath(origin,target, l);
    }

private LinkedList<NetworkingInterface> recursiveBuildPath(CompUnit origin,
CompUnit target, LinkedList<NetworkingInterface> pathSoFar){

if (enabled) {
if (connectedTo.installedOn.equals(target)) {
                pathSoFar.add(this);
return pathSoFar;
            }

for(NetworkingInterface ni:pathSoFar){//we just circled back, this path is no
good
if (ni.installedOn.equals(connectedTo.installedOn)) return null;
            }
            LinkedList<LinkedList<NetworkingInterface>> res = new
LinkedList<LinkedList<NetworkingInterface>>();

            LinkedList<NetworkingInterface> newPath= new
LinkedList<NetworkingInterface>(pathSoFar);
            newPath.add(this);

for(NetworkingInterface ni:connectedTo.installedOn.networkConnections){
                LinkedList<NetworkingInterface> tmp =
ni.recursiveBuildPath(origin, target, newPath);
if (tmp!=null&&tmp.size()!=0) res.add(tmp);
            }
if (res.size()==0) return null; else
            return res.getFirst();//we actually can choose better path
} else return null;
    }

public String toString(){
return installedOn+" to "+connectedTo.installedOn;
    }

public NetworkingInterface getConnectedTo() {
return connectedTo;
    }

public CompUnit getInstalledOn() {
return installedOn;
    }
}
```

A.14 RAM.java

```
packageCA.Entities.Hardware;

import CA.lib.Entity;

/**
```

```
 * Created by root on 08/04/2015.
 */
public class RAM extends HardwareEntity implements Entity {
public float capacity;
public float occupied;
}
```

A.15 DataSource.java

```
package CA.Entities.Logical;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Logical.Records.StringRecord;
import CA.Entities.Logical.Records.UserRecord;
import CA.Entities.Role.Role;
import CA.Entities.Software.*;
import CA.NotModel.Locator;
import CA.engine.Main;
import CA.lib.Pair;
import org.apache.logging.log4j.LogManager;

import java.util.LinkedList;


/**
 * Created by root on 15/04/2015.
 */
public class DataSource<T extends Record> {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(DataSource.class);
public static final int TYPE_FILE = 0;
public static final int TYPE_DATABASE = 1;
public static final int TYPE_EMAIL_SERVER = 2;
public static final int TYPE_WEB_SERVER = 3;
public static final int TYPE_VERSION_CONTROL_SERVER = 4;

public static final int SUCCESS = 6;
public static final int ACCESS_DENIED = 1;
public static final int PASSWORD_WRONG = 2;
public static final int COULD_NOT_REACH = 3;
public static final int TIMEOUT = 4;
public static final int GENERAL_FAILURE = 5;
public static final int NOT_STARTED = 0;
public static final int STARTED = 7;
public static final int RETRYING = 8;
public static final int NOT_FOUND = 9;


int type;
    String name;
    Component target = null;//this is for updating records locations only
CompUnit location = null;

    StringRecord userName = null;
    StringRecord password = null;

int status = NOT_STARTED;
    UserRecord record = null;
```

```
    String usernameStatic = null;
    String passwordStatic = null;

public String getUsername() {
if (record == null) return usernameStatic;
return record.getUsername();
    }

public String getPassword() {
if (record == null) return passwordStatic;
return record.getPassword();
    }

public DataSource(int type, String name, CompUnit location, UserRecord record) {

this.type = type;
this.name = name;
this.location = location;
this.record = record;
    }

public DataSource(int type, String name, CompUnit location, String userName,
String password) {

this.type = type;
this.name = name;
this.location = location;
this.passwordStatic = password;
this.usernameStatic = userName;
    }

public DataSource(FileWithRecords file) {
type = TYPE_FILE;
name = file.getFilename();
location = file.getLocation();
    }

public DataSourceGetResult getRecord(String recordName, CompUnit
queryOriginLocation){
return new
DataSourceGetResult(getRecordP(recordName,queryOriginLocation,null,null));
    }

public DataSourceGetResult getRecord(String recordName, CompUnit
queryOriginLocation, String subRecordName){
return new
DataSourceGetResult(getRecordP(recordName,queryOriginLocation,null,subRecordName)
);
    }

public DataSourceGetResult writeRecord(String recordName, CompUnit
queryOriginLocation, Record recordToWrite, String subrecordName){
return new
DataSourceGetResult(getRecordP(recordName,queryOriginLocation,recordToWrite,subre
cordName));
    }

public Pair<Record, Integer> getRecordP(String recordName, CompUnit
queryOriginLocation, Record recordToWrite, String subrecordName){
        Pair<Record,Integer> result = new Pair<>();
```

```
        Pair<RecordsCollection,Integer> res =
getRecordCollection(recordName,queryOriginLocation,recordToWrite,subrecordName);
        result.b = res.b;
if (res.a==null) return result;
        result.a=res.a.getRecord(recordName);
if (result.a==null&&subrecordName!=null){
            result.a = res.a.getFirstRecord();
        }
return result;
    }

public static DataSourceGetResults getRecordWithErrorHandling(Role role, CompUnit
queryOriginLocation, Record record){
        LinkedList<Locator> shuffledLocators =
Locator.getShuffledRecordsLocations(role, record);
return getRecordWithErrorHandling(queryOriginLocation,shuffledLocators);
    }


public static DataSourceGetResults getRecordWithErrorHandling(CompUnit
queryOriginLocation, LinkedList<Locator> shuffledLocators){
        DataSourceGetResults result = new DataSourceGetResults();

for(Locator l:shuffledLocators){
            DataSource ds = l.getSource();
            Pair<Record, Integer> p = null;
if (ds.type==TYPE_FILE)
                p = ds.getRecord(l.getTargetRecordName(),
queryOriginLocation,l.getTargetRecordName());
else p = ds.getRecord(l.getTargetRecordName(), queryOriginLocation);
if (p.b==DataSource.ACCESS_DENIED){
                result.add(p);
continue;//that was bad attempt, try next locator
}
if (p.b==DataSource.SUCCESS){
                result.add(p);
return result;
            }
//if no response yet, repeat once more
p  = ds.getRecord(l.getTargetRecordName(), queryOriginLocation);
if (p.b==DataSource.ACCESS_DENIED){
                result.add(p);
continue;//that was bad attempt, try next locator
}
if (p.b==DataSource.SUCCESS){
result.add(p);
return result;
            }
        }
        result.add(new Pair<>(null,GENERAL_FAILURE));
return result;
    }

public Pair<RecordsCollection, Integer> getRecordCollection(String recordName,
CompUnit queryOriginLocation, Record recordToWrite, String subrecordName) {
        RecordsCollection resultingRecordCollection = null;
        Pair<RecordsCollection, Integer> result = new Pair<RecordsCollection,
Integer>();
if (type == TYPE_FILE) {
if (queryOriginLocation == location) {
```

```
if (recordToWrite==null) {
if (!location.getOs().getAccessControlDecision(OperatingSystem.READ_FILE, name))
{
                        result.b = ACCESS_DENIED;
return result;
                    }
                    resultingRecordCollection =
location.getOs().getFile(name);//access control is asked inside this function
call
result.a = resultingRecordCollection;
                    result.b = SUCCESS;
return result;
                } else {
if (!location.getOs().getAccessControlDecision(OperatingSystem.WRITE_FILE, name))
{
                        result.b = ACCESS_DENIED;
return result;
                    }

                    resultingRecordCollection =
location.getOs().getFile(name);//access control is asked inside this function
call
if (resultingRecordCollection==null){
                        FileWithRecords fwr =
location.getOs().createNewFile(name);
                        resultingRecordCollection = fwr;
                    }
                    resultingRecordCollection.addRecord(recordToWrite);
                    result.a = resultingRecordCollection;
                    result.b = SUCCESS;
return result;
                }
            } else {
                Message m = new Message(location, Message.FTP, Server.authorize,
FTPServer.name);
                m.setParameter(Server.authorizeParamUsername, getUsername());
                m.setParameter(Server.authorizeParamPassword, getPassword());
                queryOriginLocation.getOs().sendMessage(m, false);
                Message response =
queryOriginLocation.getOs().listenAtSpecificPort("FTPClient");
if (response == null) {
                    result.b = TIMEOUT;
return result;
                }
                String responseCommand = response.getCommand();
if (responseCommand.equals(Server.authorizeSuccess)) {
if (recordToWrite==null) {
                        m = new Message(location, Message.FTP,
FTPServer.listCommand, FTPServer.name);
                        m.setSessionID(response.getSessionID());
                        m.setParameter(FTPServer.listParameter, name);
if (subrecordName!=null)

m.setParameter(FTPServer.listParameterRecord,subrecordName);
                        queryOriginLocation.getOs().sendMessage(m, false);
                        response =
queryOriginLocation.getOs().listenAtSpecificPort("FTPClient");
if (response == null) {
                            result.b = TIMEOUT;
return result;
```

```
                        }
if (response.getCommand().equals(FTPServer.listResponse)) {
if (subrecordName!=null) {
                                result.a = new RecordsCollection();

result.a.addRecord(response.getParameterRecord(FTPServer.listParameter));
                                }
else
result.a = response.getParameterRecordCollection(FTPServer.listParameter);
                                result.b = SUCCESS;
                            } else {
                                result.b = GENERAL_FAILURE;
                            }
                        } else{
                            m = new Message(location, Message.FTP,
FTPServer.saveCommand, FTPServer.name);
                            m.setSessionID(response.getSessionID());
                            m.setParameter(FTPServer.saveParameter, name);
                            m.setParameter(FTPServer.saveParameterRecord,
recordToWrite);
                            queryOriginLocation.getOs().sendMessage(m, false);
                            response =
queryOriginLocation.getOs().listenAtSpecificPort("FTPClient");
if (response == null) {
                                result.b = TIMEOUT;
return result;
                            }
if (response.getCommand().equals(FTPServer.saveResponse)) {
                                result.a =
response.getParameterRecordCollection(FTPServer.saveParameter);
                                result.b = SUCCESS;
                            } else {
                                result.b = GENERAL_FAILURE;
                            }
                        }
                    } else {
                        result.b = GENERAL_FAILURE;

                    }
return result;
                }
//return result;
}
if (type == TYPE_DATABASE) {
                Message m = new Message(location, Message.SMTP, Server.authorize,
DatabaseServer.name);
                m.setParameter(Server.authorizeParamUsername, getUsername());
                m.setParameter(Server.authorizeParamPassword, getPassword());
                queryOriginLocation.getOs().sendMessage(m, false);
                Message response =
queryOriginLocation.getOs().listenAtSpecificPort("DatabaseQueryOriginator");
if (response == null) {
                    result.b = TIMEOUT;
return result;
                }
                String responseCommand = response.getCommand();
if (responseCommand.equals(Server.authorizeSuccess)) {
if (recordToWrite!=null) {
                        m = new Message(location, Message.SMTP,
DatabaseServer.listCommand, DatabaseServer.name);
```

```
                    m.setSessionID(response.getSessionID());
                    m.setParameter(DatabaseServer.listParameter, recordName);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("DatabaseQueryOriginator");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(DatabaseServer.listResponse)) {
                        result.a =
response.getParameterRecordCollection(DatabaseServer.listParameter);
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
                    }
                } else{
                    m = new Message(location, Message.SMTP,
DatabaseServer.saveCommand, DatabaseServer.name);
                    m.setSessionID(response.getSessionID());
                    m.setParameter(DatabaseServer.saveParameter, recordName);
                    m.setParameter(DatabaseServer.saveParameterRecord,
recordToWrite);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("DatabaseQueryOriginator");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(DatabaseServer.saveResponse)) {
                        result.a =
response.getParameterRecordCollection(DatabaseServer.saveParameter);
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
                    }
                }
            } else {
                result.b = GENERAL_FAILURE;

            }
return result;
        }
if (type == TYPE_EMAIL_SERVER) {

Message m = new Message(location, Message.SMTP, Server.authorize,
MailServer.name);
            m.setParameter(Server.authorizeParamUsername, getUsername());
            m.setParameter(Server.authorizeParamPassword, getPassword());
            queryOriginLocation.getOs().sendMessage(m, false);
            Message response =
queryOriginLocation.getOs().listenAtSpecificPort("EmailClient");
if (response == null) {
                result.b = TIMEOUT;
return result;
            }
            String responseCommand = response.getCommand();
if (responseCommand.equals(Server.authorizeSuccess)) {
                m = new Message(location, Message.SMTP, MailServer.listCommand,
```

```
MailServer.name);
                m.setSessionID(response.getSessionID());
                m.setParameter(MailServer.listParameter, recordName);
                queryOriginLocation.getOs().sendMessage(m, false);
                response =
queryOriginLocation.getOs().listenAtSpecificPort("EmailClient");
if (response == null) {
                    result.b = TIMEOUT;
return result;
                }
if (response.getCommand().equals(MailServer.listResponse)) {

                    RecordsCollection rc = new RecordsCollection();

rc.addRecord(response.getParameterRecord(MailServer.listParameter),recordName);
                    result.a = rc;
//response.getParameterRecordCollection(MailServer.listParameter);
result.b = SUCCESS;
                } else {
                    result.b = GENERAL_FAILURE;
                }
            } else {
                result.b = GENERAL_FAILURE;

            }
return result;
        }

if (type == TYPE_WEB_SERVER) {
            Message m = new Message(location, Message.HTTP, WebServer.authorize,
WebServer.name);
            m.setParameter(Server.authorizeParamUsername, getUsername());
            m.setParameter(Server.authorizeParamPassword, getPassword());
            queryOriginLocation.getOs().sendMessage(m, false);
            Message response =
queryOriginLocation.getOs().listenAtSpecificPort("Browser");
if (response == null) {
                result.b = TIMEOUT;
return result;
            }
            String responseCommand = response.getCommand();
if (responseCommand.equals(Server.authorizeSuccess)) {
if (recordToWrite==null) {
                    m = new Message(location, Message.HTTP,
WebServer.listCommand, WebServer.name);
                    m.setSessionID(response.getSessionID());
                    m.setParameter(MailServer.listParameter, recordName);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("Browser");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(MailServer.listResponse)) {
                        result.a =
response.getParameterRecordCollection(WebServer.listParameter);
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
```

```
                    }
                } else{

                    m = new Message(location, Message.HTTP,
WebServer.saveCommand, WebServer.name);
                    m.setSessionID(response.getSessionID());
                    m.setParameter(WebServer.saveParameter, recordName);
                    m.setParameter(WebServer.saveParameterRecord, recordName);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("Browser");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(WebServer.saveResponse)) {
                        result.a =
response.getParameterRecordCollection(WebServer.saveParameter);
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
                    }
                }
            } else {
                result.b = GENERAL_FAILURE;

            }
return result;

        }

if (type == TYPE_VERSION_CONTROL_SERVER) {
            Message m = new Message(location, Message.HTTP, Server.authorize,
SourceCodeServer.name);
            m.setParameter(Server.authorizeParamUsername, getUsername());
            m.setParameter(Server.authorizeParamPassword, getPassword());
            queryOriginLocation.getOs().sendMessage(m, false);
            Message response =
queryOriginLocation.getOs().listenAtSpecificPort("CVS client");
if (response == null) {
                result.b = TIMEOUT;
return result;
            }
            String responseCommand = response.getCommand();
if (responseCommand.equals(Server.authorizeSuccess)) {
if (recordToWrite==null) {
                    m = new Message(location, Message.HTTP,
SourceCodeServer.listCommand, SourceCodeServer.name);
                    m.setSessionID(response.getSessionID());
                    m.setParameter(MailServer.listParameter, recordName);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("CVS client");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(MailServer.listResponse)) {
                        result.a =
response.getParameterRecordCollection(SourceCodeServer.listParameter);
```

```
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
                    }
                } else{

                    m = new Message(location, Message.HTTP,
SourceCodeServer.saveCommand, SourceCodeServer.name);
                    m.setSessionID(response.getSessionID());

m.setParameter(SourceCodeServer.saveParameterRecordName,subrecordName);
                    m.setParameter(SourceCodeServer.saveParameterFileName,
recordName);
                    m.setParameter(SourceCodeServer.saveParameterRecord,
recordToWrite);
                    queryOriginLocation.getOs().sendMessage(m, false);
                    response =
queryOriginLocation.getOs().listenAtSpecificPort("CVS client");
if (response == null) {
                        result.b = TIMEOUT;
return result;
                    }
if (response.getCommand().equals(SourceCodeServer.saveResponse)) {
                        result.a =
response.getParameterRecordCollection(SourceCodeServer.saveParameterRecord);
                        result.b = SUCCESS;
                    } else {
                        result.b = GENERAL_FAILURE;
                    }
                }
            } else {
                result.b = GENERAL_FAILURE;

            }
return result;

        }

return new Pair(null, GENERAL_FAILURE);
    }

public CompUnit getLocation() {
return location;
    }

public int getType() {
return type;
    }
}
```

A.16 DataSourceGetResult.java

```
packageCA.Entities.Logical;

import CA.Entities.Logical.Records.Record;
import CA.lib.Pair;

/**
 * Created by root on 14/06/2015.
 */
```

```
public class DataSourceGetResult extends Pair<Record,Integer> {

public DataSourceGetResult(Pair<Record,Integer> copy){
if (copy!=null) {
a = copy.a;
b = copy.b;
        }   else throw new NullPointerException();
    }

public Record getRecord(){
return this.a;
    }

public Integer getStatus(){
return this.b;
    }
}
```

A.17 DataSourceGetResults.java

```
packageCA.Entities.Logical;

import CA.Entities.Logical.Records.Record;
import CA.lib.Pair;

import java.util.LinkedList;

/**
 * Created by root on 14/06/2015.
 */
public class DataSourceGetResults  {

    LinkedList<DataSourceGetResult>results = null;

public DataSourceGetResults(){
results = new LinkedList<>();
    };

public DataSourceGetResults(LinkedList<DataSourceGetResult> copy){
results = copy;
    }

public boolean wasCallSuccess(){
if (results==null) return false;
if (results.size()==0) {
throw new NullPointerException();//not supposed to happen
//            return false;
}
        DataSourceGetResult dataSourceGetResult = results.getLast();
if (dataSourceGetResult==null) {
throw new NullPointerException();//not supposed to happen
//            return false;
}
return dataSourceGetResult.getStatus()==DataSource.SUCCESS;
    }

public void add(DataSourceGetResult result){
results.add(result);
    }
```

```
public void add(Pair<Record, Integer> p){
results.add(new DataSourceGetResult(p));
    }

public Record getLastRecord(){
if (results!=null&&results.size()>0) return results.getLast().a;
return null;
    }
}
```

A.18 FileWithRecords.java

```
packageCA.Entities.Logical;

import CA.Entities.Hardware.CompUnit;

/**
 * Created by root on 26/04/2015.
 */
public class FileWithRecordsextends RecordsCollection{
    String filename;
    CompUnit location;

public String getFilename() {
return filename;
    }

public CompUnit getLocation() {
return location;
    }

public FileWithRecords(String filename,CompUnit location) {
super();
this.filename = filename;
this.location = location;
        location.getOs().createNewFile(this);
    }
}
```

A.19 Message.java

```
package CA.Entities.Logical;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Hardware.NetworkingInterface;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Logical.Records.StringRecord;
import CA.Entities.Software.Component;
import CA.engine.Random;
import CA.lib.Entity;
import CA.lib.NamedEnvironment;
import org.apache.logging.log4j.LogManager;

import java.util.Iterator;
import java.util.LinkedList;

/**
 * Created by root on 08/04/2015.
 */
public class Message implements Entity {
```

```
private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Message.class);

    CompUnit from;
    CompUnit to;
    String fromPort;
    NetworkingInterface currentLocation;
int messageType;
    String stringContent;
    LinkedList<NetworkingInterface>path;
    Iterator<NetworkingInterface>traversing;
    NamedEnvironment<Object>messageParameters = new NamedEnvironment<Object>();
public boolean failedToBeDelivered = false;
public boolean noRouteForMessage = false;
public boolean encrypted = false;
    String command;
    String targetName;
    String name = ((Float)(Random.getNext())).toString();
    String sessionID = null;
    Record primaryParam = null;

public static final int HTTP = 1;
public static final int HTTPS = 2;
public static final int FTP = 3;
public static final int SFTP = 4;
public static final int ICMP = 5;
public static final int SSH = 6;
public static final int SMTP = 7;


public boolean traverse(){
if (currentLocation==null) {
traversing = path.iterator();
        }
if (!traversing.hasNext()) {
return false;
        }


currentLocation = traversing.next();

logger.trace("current message location:"+currentLocation);
return true;
    }

public CompUnit getFrom() {
return from;
    }

public void setFrom(CompUnit from) {
this.from = from;
    }

public CompUnit getTo() {
return to;
    }

public void setTo(CompUnit to) {
this.to = to;
    }
```

```
public int getMessageType() {
return messageType;
    }

public void setMessageType(int messageType) {
this.messageType = messageType;
switch(messageType){
case HTTPS:{
encrypted=true;
break;
            }
case SFTP:{
encrypted=true;
break;
            }
        }
    }

public Message() {
    }

public Message(CompUnit to) {
        setTo(to);
    }

public Message(CompUnit to, int messageType) {
this(to);
        setMessageType(messageType);
    }

public Message(CompUnit to, int messageType, String command) {
this(to, messageType);
        setCommand(command);
    }

public Message(CompUnit to, int messageType, String command, String targetName) {
this(to, messageType, command);
        setTargetName(targetName);
    }

public Message(CompUnit to, int messageType, String command, String targetName,
String fromPort) {
this(to, messageType, command, targetName);
        setFromPort(fromPort);
    }

public NetworkingInterface getCurrentLocation() {
return currentLocation;
    }

public void setCurrentLocation(NetworkingInterface currentLocation) {
this.currentLocation = currentLocation;
    }

public LinkedList<NetworkingInterface> getPath() {
return path;
    }

public void setPath(LinkedList<NetworkingInterface> path) {
```

```
this.path = path;
traversing = path.iterator();
currentLocation = path.getFirst();
    }

public Iterator<NetworkingInterface> getTraversing() {
return traversing;
    }

public void setTraversing(Iterator<NetworkingInterface> traversing) {
this.traversing = traversing;
    }

public String getStringContent() {
return stringContent;
    }

public void setStringContent(String stringContent) {
this.stringContent = stringContent;
    }

public String getCommand() {
return command;
    }

public void setCommand(String command) {
this.command = command;
    }

public String getTargetName() {
return targetName;
    }

public void setTargetName(String targetName) {
this.targetName = targetName;
    }

public boolean isParameterPresent(String name){
return messageParameters.isObjectPresent(name);
    }

public String getParameterString(String name){
return (String) messageParameters.getObject(name);
    }

public Record getParameterRecord(String name){
return (Record) messageParameters.getObject(name);
    }

public RecordsCollection getParameterRecordCollection(String name){
return (RecordsCollection) messageParameters.getObject(name);
    }

public void setParameter(String name, String value){
messageParameters.addObject(value, name);
    }

public void setParameter(String name, StringRecord value){
messageParameters.addObject(value.getValue(), name);
        value.addReference("Message:"+name,this);
```

```
    }

public void setParameter(String name, Record value){
messageParameters.addObject(value, name);
        value.addReference("Message:"+name,this);
//        int a=0;
//        a++;
}

public void setParameter(String name, RecordsCollection value){
messageParameters.addObject(value, name);
        value.addReferences("Message:"+name,this);
    }

public NamedEnvironment<Object> getMessageParameters() {
return messageParameters;
    }

public Object getParameter(String name){
return messageParameters.getObject(name);
    }

public boolean removeParameter(String name){
return messageParameters.removeObject(name);
    }

public String getFromPort() {
return fromPort;
    }

public void setFromPort(String fromPort) {
this.fromPort = fromPort;
    }

public String getSessionID() {
return sessionID;
    }

public void setSessionID(String sessionID) {
this.sessionID = sessionID;
    }
}
```

A.20 RecordsCollection.java

```
packageCA.Entities.Logical;

import CA.Entities.Logical.Records.Record;
import CA.engine.Random;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Created by root on 15/04/2015.
 */
public class RecordsCollection {
    HashMap<String,Record>records = new HashMap<String, Record>();
```

```
public Record getRecord(String recordName){
return records.get(recordName);
    }

public void addRecord(Record r, String name){
records.put(name,r);
        r.addReferenceWithRandomizedName(this);
    }

public void addRecord(Record r){
        String name = Random.getNewID();
        addRecord(r,name);
    }

public void addReferences(String prename, Object referringObject){
for(String s:records.keySet())
records.get(s).addReference(prename+Random.getNewID(),referringObject);
    }

public boolean removeReference(String name, Record r) {
        Record o = records.remove(name);
if (o == null) {
for (Map.Entry<String, Record> e : records.entrySet()) {
if (r == e.getValue()) {
records.remove(e.getKey());
return true;
                }
            }
return false;
        } else {
return true;
        }
    }

public boolean removeReference(Record r) {
for (Map.Entry<String, Record> e : records.entrySet()) {
if (r == e.getValue()) {
records.remove(e.getKey());
return true;
            }
        }
return false;
    }

public String getNameOfRecord(Record r){
for(String s:records.keySet()){
if (records.get(s)==r) return s;
        }
return null;
    }

public Set<String> getRecordNames(){
return records.keySet();
    }

public Record getFirstRecord(){
        Set<String> names = records.keySet();
for(String s:names)
return records.get(s);
return null;
```

```
    }
}
```

A.21 DoubleRecord.java

```
packageCA.Entities.Logical.Records;

/**
 * Created by root on 15/04/2015.
 */
public class DoubleRecord extends Record<Double> {
public DoubleRecord(Double newData,String recordName) {
super(newData, recordName);
    }
}
```

A.22 EmailRecord.java

```
packageCA.Entities.Logical.Records;

import CA.Entities.Software.MailServer;

import java.util.LinkedList;

/**
 * Created by root on 27/04/2015.
 */
public class EmailRecord extends Record {
    String from;
    String to;
    String body;
    EmailRecord copyOf = null;
    MailServer stored;
    MailServer sent;
long timeReceived;
long timeSent;



boolean deleted = false;
boolean spamChecked = false;
boolean antivirusChecked = false;
boolean dataLossPreventionChecked = false;
    LinkedList<Record>attachments = null;

public EmailRecord(String recordName) {
super(null,recordName);
    }

public String getFrom() {
return from;
    }

public void setFrom(String from) {
this.from = from;
    }

public String getTo() {
return to;
    }
```

```
public void setTo(String to) {
this.to = to;
    }

public String getBody() {
return body;
    }

public void setBody(String body) {
this.body = body;
    }

public EmailRecord getCopyOf() {
return copyOf;
    }

public void setCopyOf(EmailRecord copyOf) {
this.copyOf = copyOf;
        copyOf.incorporateRecord(this);
    }

public MailServer getStored() {
return stored;
    }

public void setStored(MailServer stored) {
this.stored = stored;
    }

public MailServer getSent() {
return sent;
    }

public void setSent(MailServer sent) {
this.sent = sent;
    }

public long getTimeReceived() {
return timeReceived;
    }

public void setTimeReceived(long timeReceived) {
this.timeReceived = timeReceived;
    }

public long getTimeSent() {
return timeSent;
    }

public void setTimeSent(long timeSent) {
this.timeSent = timeSent;
    }

public boolean isSpamChecked() {
return spamChecked;
    }

public void setSpamChecked(boolean spamChecked) {
this.spamChecked = spamChecked;
    }
```

```
public boolean isAntivirusChecked() {
return antivirusChecked;
    }

public void setAntivirusChecked(boolean antivirusChecked) {
this.antivirusChecked = antivirusChecked;
    }

public boolean isDataLossPreventionChecked() {
return dataLossPreventionChecked;
    }

public void setDataLossPreventionChecked(boolean dataLossPreventionChecked) {
this.dataLossPreventionChecked = dataLossPreventionChecked;
    }

public LinkedList<Record> getAttachments() {
return attachments;
    }

public void addAttachment(Record attachment) {
if (attachments==null) attachments = new LinkedList<>();
this.attachments.add(attachment);
this.incorporateRecord(attachment);
        attachment.addReferenceWithRandomizedName(this);
    }

public void addReference(String name, Object reference) {
if (references.containsValue(reference)) return;//infinite recursion prevention
if attaching email record to itself at some point
super.addReference(name,reference);
for(Record r: attachments)
            r.addReference(name,reference);
    }


}
```

A.23 EmailRecordClass.java

```
packageCA.Entities.Logical.Records;

/**
 * Created by root on 27/04/2015.
 */
public class EmailRecordClass extends Record<EmailRecord> {
public EmailRecordClass(EmailRecord er,String recordName){
super(er,recordName);
    }
}
```

A.24 IntegerRecord.java

```
packageCA.Entities.Logical.Records;

/**
 * Created by root on 15/04/2015.
 */
public class IntegerRecord extends Record<Integer> {
public IntegerRecord(Integer newData,String recordName) {
```

```
super(newData,recordName);
    }
}
```

A.25 Record.java

```
packageCA.Entities.Logical.Records;

import CA.engine.ActionsQueue;
import CA.engine.Random;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

/**
 * Created by root on 15/04/2015.
 */
public class Record<T> {
public static LinkedList<Record>globalList = new LinkedList<>();
public String recordName;
T data;
/**
     * Existing references to this record
     */
public HashMap<String, Object>references = new HashMap<String, Object>();

/**
     * Damage to security if stolen
     */
Double securityValue = 0d;

    LinkedList<Record>recordsIncorporatedIntoThisRecord = new LinkedList<>();

public void addReference(String name, Object reference) {
if (references.containsValue(reference)) return;//infinite recursion prevention
if record incorporates itself
references.put(name, reference);
for(Record r: recordsIncorporatedIntoThisRecord)
            r.addReference(name,reference);
    }

public HashMap<String, Object> getReferences() {
return references;
    }

public LinkedList<Record> getRecordsIncorporatedIntoThisRecord() {
return recordsIncorporatedIntoThisRecord;
    }

public void addReferenceWithRandomizedName(String prename,Object reference){
        addReference(prename+ Random.getNewID(),reference);
    }

public void addReferenceWithRandomizedName(Object reference){
        addReferenceWithRandomizedName("",reference);
    }

public Record(T newData, String recordName){
this.data = newData;
```

```
this.recordName = recordName;
globalList.add(this);
    }

public boolean removeReference(String name, Object reference) {
        Object o = references.remove(name);
if (o == null) {
for (Map.Entry<String, Object> e : references.entrySet()) {
if (reference == e.getValue()) {
references.remove(e.getKey());
return true;
                }
            }
return false;
        } else {
return true;
        }
    }

public boolean removeReference(Object reference) {
for (Map.Entry<String, Object> e : references.entrySet()) {
if (reference == e.getValue()) {
references.remove(e.getKey());
return true;
            }
        }
return false;
    }

public T getValue(){
return data;
    }

public void incorporateRecord(Record r){
this.recordsIncorporatedIntoThisRecord.add(r);
        r.addReference("Incorporated at:"+
ActionsQueue.currentQueue.getCurrentTime(),this);
    }

public boolean hasReference(Object obj){
for(Map.Entry<String,Object> en:references.entrySet()){
if (en.getValue()==obj) return true;
        }
return false;
    }


public Double getSecurityValue() {
return securityValue;
    }

public void setSecurityValue(Double securityValue) {
this.securityValue = securityValue*2;//2 is a current "price"
}

public String toString(){
if (recordName!=null) return recordName;
if (data!=null) return data.toString()+":"+super.toString();
return super.toString();
```

```
    }
}
```

A.26 StringRecord.java

```
packageCA.Entities.Logical.Records;

/**
 * Created by root on 15/04/2015.
 */
public class StringRecord extends Record<String> {
public StringRecord(String value,String recordName){
super(value,recordName);
    }
}
```

A.27 UserRecord.java

```
packageCA.Entities.Logical.Records;

import CA.Entities.Role.Role;

/**
 * Created by root on 26/04/2015.
 */
public class UserRecord {
    String username;
    String password;
    String roleName;

public String getUsername() {
return username;
    }

public void setUsername(String username) {
this.username = username;
    }

public String getPassword() {
return password;
    }

public void setPassword(String password) {
this.password = password;
    }

public String getRoleName() {
return roleName;
    }

public void setRoleName(String roleName) {
this.roleName = roleName;
    }

public UserRecord(String username, String password, String roleName) {
this.username = username;
this.password = password;
this.roleName = roleName;
    }
```

```
}
```

A.28 UserRecordClass.java

```
packageCA.Entities.Logical.Records;

/**
 * Created by root on 26/04/2015.
 */
public class UserRecordClass extends Record<UserRecord>{
public UserRecordClass(UserRecord record,String recordName){
super(record,recordName);
    }
}
```

A.29 Location.java

```
packageCA.Entities.Physical;

import CA.lib.Entity;

import java.util.LinkedList;

/**
 * Created by root on 19/04/2015.
 */
public class Locationimplements Entity {
    String name;
    LinkedList<Object>objectsItHolds = new LinkedList<Object>();
    LinkedList<Location>nearbyLocations = new LinkedList<Location>();

public Location(String name) {
this.name = name;
    }

public LinkedList<Object> getObjectsItHolds() {
return objectsItHolds;
    }

public LinkedList<Location> getNearbyLocations() {
return nearbyLocations;
    }

public void addObjectToLocation(Object object){
objectsItHolds.add(object);
    }

public boolean removeObjectFromLocation(Object object){
return objectsItHolds.remove(object);
    }
}
```

A.30 Customer.java

```
packageCA.Entities.Role;

import CA.Entities.Physical.Location;
import CA.Processes.ProcessTree;

/**
```

```
 * Created by root on 09/04/2015.
 */
public class CustomerRole extends Role {

static ProcessTree processTree;
static{
processTree = new ProcessTree("CustomerProcessTree");

    }

public CustomerRole(Location location) {
super(location);
    }
}
```

A.31 DeveloperRole.java

```
package CA.Entities.Role;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Logical.DataSource;
import CA.Entities.Logical.DataSourceGetResults;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Logical.Records.StringRecord;
import CA.Entities.Logical.RecordsCollection;
import CA.Entities.Physical.Location;
import CA.NotModel.Locator;
import CA.NotModel.RiskCollection;
import CA.Processes.*;
import CA.engine.Main;
import RMS.RMS;
import CA.engine.Random;
import CA.lib.Pair;
import org.apache.logging.log4j.LogManager;

import java.util.LinkedList;

import static CA.Processes.TaskResult.TASK_STATUS_FAILURE;
import static CA.Processes.TaskResult.TASK_STATUS_SUCCESS;

/**
 * Created by root on 09/04/2015.
 */
public class DeveloperRole extends Role {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(DeveloperRole.class);
public ProcessTree processTree;

static int a = 0;


public void setEnvironmentVariables(Focus f) {

        RecordsCollection phaseOneMandatoryProblem = new RecordsCollection();
        phaseOneMandatoryProblem.addRecord(Main.docA, "docA");
        phaseOneMandatoryProblem.addRecord(Main.headerA, "headerA");
        f.setVariable("phaseOneMandatoryProblem", phaseOneMandatoryProblem);
```

```
RecordsCollection phaseOneBeneficialProblem = new RecordsCollection();
phaseOneBeneficialProblem.addRecord(Main.methodA, "methodA");
phaseOneBeneficialProblem.addRecord(Main.docB, "docB");
phaseOneBeneficialProblem.addRecord(Main.headerB, "headerB");
f.setVariable("phaseOneBeneficialProblem", phaseOneBeneficialProblem);

RecordsCollection phaseTwoMandatoryProblem = new RecordsCollection();
phaseTwoMandatoryProblem.addRecord(Main.docD, "docD");
phaseTwoMandatoryProblem.addRecord(Main.headerD, "headerD");
f.setVariable("phaseTwoMandatoryProblem", phaseTwoMandatoryProblem);

RecordsCollection phaseTwoBeneficialProblem = new RecordsCollection();
phaseTwoBeneficialProblem.addRecord(Main.docC, "docC");
phaseTwoBeneficialProblem.addRecord(Main.headerC, "headerC");
phaseTwoBeneficialProblem.addRecord(Main.methodD, "methodD");
f.setVariable("phaseTwoBeneficialProblem", phaseTwoBeneficialProblem);

RecordsCollection phaseOneMandatoryWrite = new RecordsCollection();
phaseOneMandatoryWrite.addRecord(Main.headerA, "headerA");
phaseOneMandatoryWrite.addRecord(Main.methodA, "methodA");
f.setVariable("phaseOneMandatoryWrite", phaseOneMandatoryWrite);

RecordsCollection phaseOneBeneficialWrite = new RecordsCollection();
phaseOneBeneficialWrite.addRecord(Main.docA, "docA");
f.setVariable("phaseOneBeneficialWrite", phaseOneBeneficialWrite);

RecordsCollection phaseTwoMandatoryWrite = new RecordsCollection();
phaseTwoMandatoryWrite.addRecord(Main.headerD, "headerD");
phaseTwoMandatoryWrite.addRecord(Main.methodD, "methodD");
f.setVariable("phaseTwoMandatoryWrite", phaseTwoMandatoryWrite);

RecordsCollection phaseTwoBeneficialWrite = new RecordsCollection();
phaseTwoBeneficialWrite.addRecord(Main.docD, "docD");
f.setVariable("phaseTwoBeneficialWrite", phaseTwoBeneficialWrite);

Double docARead = 0d;
Double headerARead = 0d;
Double methodARead = 0d;
Double docBRead = 0d;
Double headerBRead = 0d;
Double methodBRead = 0d;
Double docCRead = 0d;
Double headerCRead = 0d;
Double methodCRead = 0d;
Double docDRead = 0d;
Double headerDRead = 0d;
Double methodDRead = 0d;

f.setVariable("docA", docARead);
f.setVariable("headerA", headerARead);
f.setVariable("methodA", methodARead);
f.setVariable("docB", docBRead);
f.setVariable("headerB", headerBRead);
f.setVariable("methodB", methodBRead);
f.setVariable("docC", docCRead);
f.setVariable("headerC", headerCRead);
f.setVariable("methodC", methodCRead);
f.setVariable("docD", docDRead);
f.setVariable("headerD", headerDRead);
f.setVariable("methodD", methodDRead);
```

```
        f.setVariable("docARecord", Main.docA);
        f.setVariable("headerARecord", Main.headerA);
        f.setVariable("methodARecord", Main.methodA);
        f.setVariable("docBRecord", Main.docB);
        f.setVariable("headerBRecord", Main.headerB);
        f.setVariable("methodBRecord", Main.methodB);
        f.setVariable("docCRecord", Main.docC);
        f.setVariable("headerCRecord", Main.headerC);
        f.setVariable("methodCRecord", Main.methodC);
        f.setVariable("docDRecord", Main.docD);
        f.setVariable("headerDRecord", Main.headerD);
        f.setVariable("methodDRecord", Main.methodD);

        f.setVariable("partOneProblemDone", false);
        f.setVariable("partTwoProblemDone", false);
        f.setVariable("partOneCodeDone", false);
        f.setVariable("partTwoCodeDone", false);
    }

public double calculateProblemSolutionProbabilityPhaseOne(Focus f, CompUnit
location, Task t, String mandatoryRecordCollectionName, String
optionalRecordCollectionName) {
double count = 0;
double res = 0;
        RecordsCollection rcM = (RecordsCollection)
f.getVariable(mandatoryRecordCollectionName);
for (String s : rcM.getRecordNames()) {
            DeveloperRole role = (DeveloperRole) t.getParent();
            Record r = rcM.getRecord(s);
            Locator.isRecordHere(role, r, location);
            Double d = (Double) f.getVariable(s);
if (d != null) {
if (d <0) d = 0d;
if (d >1) d = 1d;
                res += d;
if (d == 0) return 0;
            }
            count++;
        }

        RecordsCollection rcB = (RecordsCollection)
f.getVariable(optionalRecordCollectionName);
for (String s : rcB.getRecordNames()) {
            DeveloperRole role = (DeveloperRole) t.getParent();
            Record r = rcB.getRecord(s);
            Locator.isRecordHere(role, r, location);
            Double d = (Double) f.getVariable(s);
if (d != null) {
if (d <0) d = 0d;
if (d >1) d = 1d;
                res += d;
            }
            count++;
        }
return res / count;
    }

/**
     * update Record as if it was read
```

```
     *
     * @param f
* @param location
* @param t
* @param recordName
* @return
*/
public boolean updateReadRecord(Focus f, CompUnit location, Task t, String
recordName) {
        DeveloperRole role = (DeveloperRole) t.getParent();
        Record r = (Record) f.getVariable(recordName + "Record");
boolean recordIsThere = Locator.isRecordHere(role, r, location);
if (recordIsThere) {
            Double d = (Double) f.getVariable(recordName);
if (d <0) d = 0d;
if (d >1) d = 1d;
if (d == null || d == 0) {
                d = 0.45d;
                d += Random.getNext() / 10;//0.5+-0.05
} else {
double mul = 0.55d;
                mul += Random.getNext() / 10;//1-(1-d)*(0.6+-0.05)
d = 1 - (1 - d) * mul;
            }
            f.setVariable(recordName, d);
        }
return recordIsThere;
    }

public DeveloperRole(Location location) {
super(location);

processTree = new ProcessTree("DeveloperProcessTree");

        Task setEnvironment = new Task("SetEnvironment", 10, this) {//done
public void taskActions(Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();

                setEnvironmentVariables(f);

if (role.unitForWork == null) {
for (Object o : role.location.getObjectsItHolds()) {
if (o instanceof CompUnit) {
unitForWork = (CompUnit) o;//use the random one
setStatus(TASK_STATUS_SUCCESS);
return;
                        }
                    }
                }
if (unitForWork == null) //first find any nearby CompUnit
setStatus(TASK_STATUS_FAILURE);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
if (taskList.wasCallSuccess()) {//if ok, then continue
return findTask(nextNodes, "getTask");
                }
return findTask(nextNodes, "TaskDone");
            }
```

```
            ;
        };

        Task getTask = new Task("GetTask", 1000 * 60 * 3, this) {//done
public void taskActions(Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();

                DataSourceGetResults results =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, Main.projectTask);
if (results.wasCallSuccess()) {
                    setStatus(TASK_STATUS_SUCCESS);
                    FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\ProjectTask");
                    fwr.addRecord(results.getLastRecord(), "ProjectTask");
return;
                } else {
                    setStatus(TASK_STATUS_FAILURE);
                }
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
if (taskList.wasCallSuccess()) {//if ok, then continue
return findTask(nextNodes, "ProblemSpecification");
                }
if (taskList.count() >2 &&
                        !taskList.wasCallSuccess(0) &&
taskList.wasLastTaskNamed("getTask", 0) &&
                        !taskList.wasCallSuccess(1) &&
taskList.wasLastTaskNamed("getTask", 1))
return findTask(nextNodes, "taskDone");

return findTask(nextNodes, "getTask");//otherwise just repeat
}
        };//done

Task problemSpecification = new Task("ProblemSpecification", 15 * 60 * 1000,
this) {//done
public void taskActions(Focus f) {
//task consists of two parts,
                // first one needs docA, headerA beneficial: methodA, docB,
headerB
                // second one needs docD, headerD beneficial:docC, headerC,
methodD
DeveloperRole role = (DeveloperRole) getParent();
                Boolean b1 = (Boolean) f.getVariable("partOneProblemDone");
                Boolean b2 = (Boolean) f.getVariable("partTwoProblemDone");
if (!b1) {//if part one was not done
double d = calculateProblemSolutionProbabilityPhaseOne(f, role.unitForWork, this,
"phaseOneMandatoryProblem", "phaseOneBeneficialProblem");
if (d >0) {
double r = Random.getNext();
if (d > r) {
logger.info("first problem solved");
                            f.setVariable("partOneProblemDone", true);
                            setStatus(TASK_STATUS_SUCCESS);//problem just got
solved
return;
                        } else {
```

```
logger.info("solving problem failed with probability:"+d);
                            setStatus(TASK_STATUS_SUCCESS);//problem was not
solved, but there is a chance that it would
return;
                        }
                    } else {
                        setStatus(TASK_STATUS_FAILURE);//no chance problem was
solved
return;
                    }
                } else {//if part one was done
if (!b2) {//if part two was done
double d = calculateProblemSolutionProbabilityPhaseOne(f, role.unitForWork, this,
"phaseTwoMandatoryProblem", "phaseTwoBeneficialProblem");
if (d >0) {
double r = Random.getNext();
if (d > r) {
logger.info("second problem solved");
                                f.setVariable("partTwoProblemDone", true);
                                setStatus(TASK_STATUS_SUCCESS);//problem just got
solved
return;
                            } else {
logger.info("solving problem failed with probability:"+d);
                                setStatus(TASK_STATUS_SUCCESS);//problem was not
solved, but there is a chance that it would
return;
                            }
                        } else {
                            setStatus(TASK_STATUS_FAILURE);//no chance problem
was solved
return;
                        }
                    } else {
                        setStatus(TASK_STATUS_SUCCESS);//part one and two
analysis are done, we dont need to do anything
return;
                    }
                }

            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();
if
(taskList.wasCallSuccess()&&f.getBooleanVariable("partOneProblemDone")&&!f.getBoo
leanVariable("partOneCodeDone")) {//if we can continue to writing code - go there
return findTask(nextNodes, "WriteCode");
                }
if
(taskList.wasCallSuccess()&&f.getBooleanVariable("partTwoProblemDone")&&!f.getBoo
leanVariable("partTwoCodeDone")) {//if we can continue to writing code - go there
return findTask(nextNodes, "WriteCode");
                }

if
(f.getBooleanVariable("partOneProblemDone")&&f.getBooleanVariable("partOneCodeDon
e")&&f.getBooleanVariable("partTwoProblemDone")&&f.getBooleanVariable("partTwoCod
eDone")) {//if we have everything written - just go there
```

```
return findTask(nextNodes, "WriteCode");
                }

double d = Random.getNext();
if (d >0.4)
return findTask(nextNodes, "ReadDocumentation");
else
                    return findTask(nextNodes, "ReadCode");
            }
        };

        Task writeCode = new Task("WriteCode", 10 * 60 * 1000, this) {

public void taskActions(Focus f) {
//task consists of two parts,
                // first one needs headerA,methodA beneficial: docA
                // second one needs headerD, methodD beneficial: docD

DeveloperRole role = (DeveloperRole) getParent();
                Boolean b1 = (Boolean) f.getVariable("partOneCodeDone");
                Boolean b2 = (Boolean) f.getVariable("partTwoCodeDone");
if (!b1) {//if part one was not done
if (!f.getBooleanVariable("partOneProblemDone")){
                        setStatus(TASK_STATUS_FAILURE);//no chance this piece of
code could be written before the solution is created
return;
                    }
double d = calculateProblemSolutionProbabilityPhaseOne(f, role.unitForWork, this,
"phaseOneMandatoryWrite", "phaseOneBeneficialWrite");
if (d >0) {
double r = Random.getNext();
if (d > r) {
                            Main.methodANew = new StringRecord("CREATE TABLE
new_tbl SELECT * FROM orig_tbl where id > 100","methodANew");
                            f.setVariable("methodANew", Main.methodANew);
                            FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp1.tmp");
                            fwr.addRecord(Main.methodANew);
                            RiskCollection.addProductivityRisk(-3);
logger.info("first piece of code written");
                            f.setVariable("partOneCodeDone", true);
                            setStatus(TASK_STATUS_SUCCESS);//problem just got
solved
return;
                        } else {
logger.info("writing code failed with probability:"+d);
                            setStatus(TASK_STATUS_SUCCESS);//problem was not
solved, but there is a chance that it would
return;
                        }
                    } else {
                        setStatus(TASK_STATUS_FAILURE);//no chance problem was
solved
return;
                    }
                } else {//if part one was done
if (!b2) {//if part two was done
if (!f.getBooleanVariable("partTwoProblemDone")){
                            setStatus(TASK_STATUS_FAILURE);//no chance this piece
of code could be written before the solution is created
```

```
return;
                        }
double d = calculateProblemSolutionProbabilityPhaseOne(f, role.unitForWork, this,
"phaseTwoMandatoryWrite", "phaseTwoBeneficialWrite");
if (d >0) {
double r = Random.getNext();
if (d > r) {
                                Main.headerDNew = new StringRecord("public static
void main(String[] args, String name)","headerDNew");
                                f.setVariable("headerDNew", Main.headerDNew);
                                FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp2.tmp");
                                fwr.addRecord(Main.headerDNew);

                                Main.methodDNew = new
StringRecord("System.out.println(\"test\");System.exit(0)","methodDNew");
                                f.setVariable("methodDNew", Main.methodDNew);
                                fwr.addRecord(Main.methodDNew);
                                RiskCollection.addProductivityRisk(-5);
logger.info("second piece of code written");
                                f.setVariable("partTwoCodeDone", true);
                                setStatus(TASK_STATUS_SUCCESS);//problem just got
solved
return;
                            } else {
logger.info("writing code failed with probability:"+d);
                                setStatus(TASK_STATUS_SUCCESS);//problem was not
solved, but there is a chance that it would
return;
                            }
                        } else {
                            setStatus(TASK_STATUS_FAILURE);//no chance problem
was solved
return;
                        }
                    } else {
                        setStatus(TASK_STATUS_SUCCESS);//part one and two
analysis are done, we dont need to do anything
return;
                    }
                }
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();

if (taskList.wasCallSuccess() &&
f.getBooleanVariable("partOneCodeDone")&&!f.getBooleanVariable("partOneCodeSubmit
ted")) {//if ok, then continue
return findTask(nextNodes, "SubmitCode");
                }
if (taskList.wasCallSuccess() &&
f.getBooleanVariable("partTwoCodeDone")&&!f.getBooleanVariable("partTwoCodeSubmit
ted")) {//if ok, then continue
return findTask(nextNodes, "SubmitCode");
                }
if (f.getBooleanVariable("partTwoCodeDone")) {//should be covered by previous if
statement
return findTask(nextNodes, "SubmitCode");
```

```
                }

if (!f.getBooleanVariable("partTwoProblemDone")){
return findTask(nextNodes, "ProblemSpecification");
                }

if (!taskList.wasCallSuccess()){
return findTask(nextNodes, "ProblemSpecification");
                }

double d = Random.getNext();
if (d >0.7)
return findTask(nextNodes, "ReadDocumentation");
else
                    return findTask(nextNodes, "ReadCode");
//  return findTask(nextNodes, "ProblemSpecification");
}
        };


        Task submitCode = new Task("SubmitCode", 2 * 60 * 1000, this) {
public void taskActions(Focus f) {
boolean stageOneDone = (boolean) f.getVariable("partOneCodeDone");
boolean stageTwoDone = (boolean) f.getVariable("partTwoCodeDone");

                DeveloperRole role = (DeveloperRole) getParent();

//DataSource.getRecordWithErrorHandling(role, role.unitForWork,
Main.projectTask);
Record r = null;


if (stageOneDone && !stageTwoDone) {
                    LinkedList<Locator> loc = Locator.getRecordsLocations(role,
Main.methodA,Main.buildServer);
                    DataSource ds = loc.getFirst().getSource();
                    Pair<Record, Integer> p =
ds.getRecordP("c:\\create_database.sql", role.unitForWork, (Record)
f.getVariable("methodANew"), "MethodA1");
if (p.b == DataSource.SUCCESS) {
                        f.setVariable("partOneCodeSubmitted",true);
                        setStatus(TASK_STATUS_SUCCESS);
                        RiskCollection.addProductivityRisk(-1);
                    } else
setStatus(TASK_STATUS_FAILURE);
return;
                }

if (stageOneDone && stageTwoDone) {
                    LinkedList<Locator> loc = Locator.getRecordsLocations(role,
Main.methodD,Main.buildServer);
                    DataSource ds = loc.getFirst().getSource();
                    Pair<Record, Integer> p1 =
ds.getRecordP("c:\\mobile_application.java", role.unitForWork, (Record)
f.getVariable("headerDNew"), "HeaderD1");
                    Pair<Record, Integer> p2 =
ds.getRecordP("c:\\mobile_application.java", role.unitForWork, (Record)
f.getVariable("methodDNew"), "MethodD1");

if (p1.b == DataSource.SUCCESS && p2.b == DataSource.SUCCESS) {
```

```
                        f.setVariable("partTwoCodeSubmitted",true);
                        setStatus(TASK_STATUS_SUCCESS);
                        RiskCollection.addProductivityRisk(-2);
                    } else
setStatus(TASK_STATUS_FAILURE);
return;
                }
                setStatus(TASK_STATUS_FAILURE);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
boolean stageTwoDone = (boolean) f.getVariable("partTwoCodeDone");

if (taskList.wasCallSuccess()&&stageTwoDone) {//if ok, then continue
return findTask(nextNodes, "TestCodeAutomatically");}
if (taskList.wasCallSuccess()&&!stageTwoDone) {//if ok, then continue
return findTask(nextNodes, "WriteCode");
}

return findTask(nextNodes, "TestCodeLocally");
            }
        };


        Task testCodeAutomatically = new Task("TestCodeAutomatically", 60 * 1000,
this) {
public void taskActions(Focus f) {
if (Main.methodANew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.methodA, Main.buildServer)) {
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
if (Main.methodDNew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.methodDNew, Main.buildServer)) {
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
if (Main.headerDNew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.headerDNew, Main.buildServer)) {
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
                setStatus(TASK_STATUS_SUCCESS);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
if (taskList.wasCallSuccess()) {//if ok, then continue
return findTask(nextNodes, "TaskDone");
                }
return findTask(nextNodes, "ProblemSpecification");
}
        };

        Task testCodeLocally = new Task("TestCodeLocally", 60 * 1000, this) {
public void taskActions(Focus f) {
if (Main.methodANew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.methodA, ((DeveloperRole) this.getParent()).unitForWork))
{
```

```
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
if (Main.methodDNew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.methodDNew, ((DeveloperRole)
this.getParent()).unitForWork)) {
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
if (Main.headerDNew == null || !Locator.isRecordHere(((DeveloperRole)
this.getParent()), Main.headerDNew, ((DeveloperRole)
this.getParent()).unitForWork)) {
                    setStatus(TASK_STATUS_FAILURE);
return;
                }
                setStatus(TASK_STATUS_SUCCESS);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
if (taskList.wasCallSuccess()) {//if ok, then continue
return findTask(nextNodes, "TaskDone");
                }
return findTask(nextNodes, "ProblemSpecification");
}
        };
```

```
        Task getCode = new Task("GetCode", 2 * 60 * 1000, this) {
public void taskActions(Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();

boolean gotSomethingNew = false;

int recordsCount = 0;

                Record[] recordsToGet = {Main.headerA, Main.methodA,
Main.headerB, Main.methodB, Main.headerC, Main.methodC, Main.headerD,
Main.methodD};
for (Record r : recordsToGet)
if (!Locator.isRecordHere(role, r, role.unitForWork)) {
                        DataSourceGetResults results =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, r);
if (results.wasCallSuccess()) {
                            String name = r.toString();
                            FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\" + name);
                            fwr.addRecord(results.getLastRecord(), name);
                            gotSomethingNew = true;
                        }
                    } else recordsCount++;


if (gotSomethingNew) {
```

```
                    setStatus(TASK_STATUS_SUCCESS);
return;
                } else {
if (recordsCount == recordsToGet.length)
                        setStatus(TASK_STATUS_SUCCESS);
else
setStatus(TASK_STATUS_FAILURE);
                }
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
return findTask(nextNodes, "ReadCode");//default return point
}
        };

        Task readCode = new Task("ReadCode", 15 * 60 * 1000, this) {
public void taskActions(Focus f) {
                String[] args = {
"headerA", "methodA",
"headerB", "methodB",
"headerC", "methodC",
"headerD", "methodD"
};
                DeveloperRole role = (DeveloperRole) getParent();
for (String s : args)
                    updateReadRecord(f, role.unitForWork, this, s);
                setStatus(TASK_STATUS_SUCCESS);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {

if (taskList.wasLastTaskNamed("GetCode", 1))//if we actually got something new
and we were called from Read documentation we can return
return findTask(nextNodes, callStack.getLastTask(2).getTaskName());
else
                    return findTask(nextNodes, "GetCode");
            }
        };

        Task getDocumentation = new Task("GetDocumentation", 10, this) {
public void taskActions(Focus f) {
                DeveloperRole role = (DeveloperRole) getParent();

boolean gotSomethingNew = false;

                Record r;
int recordsCount = 0;

                r = Main.docA;
if (!Locator.isRecordHere(role, r, role.unitForWork)) {
                    DataSourceGetResults resultsB1 =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, r);
if (resultsB1.wasCallSuccess()) {
                        String name = "DocA1";
                        FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\" + name);
                        fwr.addRecord(resultsB1.getLastRecord(), name);
                        gotSomethingNew = true;
```

```
                    }
                } else recordsCount++;

                r = Main.docB;
if (!Locator.isRecordHere(role, r, role.unitForWork)) {
                    DataSourceGetResults resultsB1 =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, r);
if (resultsB1.wasCallSuccess()) {
                        String name = "DocB1";
                        FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\" + name);
                        fwr.addRecord(resultsB1.getLastRecord(), name);
                        gotSomethingNew = true;
                    }
                } else recordsCount++;

                r = Main.docC;
if (!Locator.isRecordHere(role, r, role.unitForWork)) {
                    DataSourceGetResults resultsB1 =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, r);
if (resultsB1.wasCallSuccess()) {
                        String name = "DocC1";
                        FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\" + name);
                        fwr.addRecord(resultsB1.getLastRecord(), name);
                        gotSomethingNew = true;
                    }
                } else recordsCount++;

                r = Main.docD;
if (!Locator.isRecordHere(role, r, role.unitForWork)) {
                    DataSourceGetResults resultsB1 =
DataSource.getRecordWithErrorHandling(role, role.unitForWork, r);
if (resultsB1.wasCallSuccess()) {
                        String name = "DocD1";
                        FileWithRecords fwr =
role.unitForWork.getOs().createNewFile("c:\\temp\\" + name);
                        fwr.addRecord(resultsB1.getLastRecord(), name);
                        gotSomethingNew = true;
                    }
                } else recordsCount++;

if (gotSomethingNew) {
                    setStatus(TASK_STATUS_SUCCESS);
return;
                } else {
if (recordsCount == 4)
                        setStatus(TASK_STATUS_SUCCESS);
else
setStatus(TASK_STATUS_FAILURE);
                }
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
return findTask(nextNodes, "ReadDocumentation");
            }
        };

        Task readDocumentation = new Task("ReadDocumentation", 15 * 60 * 1000,
```

```
this) {
public void taskActions(Focus f) {
                String[] args = {
"docA",
"docB",
"docC",
"docD"
};
                DeveloperRole role = (DeveloperRole) getParent();
for (String s : args)
                    updateReadRecord(f, role.unitForWork, this, s);
                setStatus(TASK_STATUS_SUCCESS);
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
if (taskList.wasLastTaskNamed("GetDocumentation", 1))//if we actually got
something new and we were called from Read documentation we can return
return findTask(nextNodes, callStack.getLastTask(2).getTaskName());
else
                    return findTask(nextNodes, "GetDocumentation");
            }
        };


Task taskDone = new Task("TaskDone") {
public void taskActions(Focus f) {
                RMS.modellingIsCompleted();
            }

public Task chooseCustomNext(TaskResults taskList, TaskResults callStack,
LinkedList<Task> nextNodes, Focus f) {
for (TaskResult trP : taskList.getList())
logger.info(trP);
                TaskResult tr = taskList.getLastTaskResult(1);
if (tr.getStatus() == TASK_STATUS_SUCCESS &&
tr.getTask().isTaskNamed("TestCodeLocally")) {
                    RiskCollection.addProductivityRisk(-2);
                }

if (tr.getStatus() == TASK_STATUS_SUCCESS &&
tr.getTask().isTaskNamed("TestCodeAutomatically")) {
                    RiskCollection.addProductivityRisk(-5);
                }
return null;
            }
        };


        Random.getNext();

processTree.addEntryPoint(setEnvironment);
processTree.addExitPoint(taskDone);

        setEnvironment.addNextNode(getTask);

        getTask.addNextNode(getTask);
        getTask.addNextNode(taskDone);
        getTask.addNextNode(problemSpecification);
```

```
        problemSpecification.addNextNode(problemSpecification);
        problemSpecification.addNextNode(readDocumentation);
        problemSpecification.addNextNode(readCode);
        problemSpecification.addNextNode(writeCode);

        writeCode.addNextNode(submitCode);
        writeCode.addNextNode(testCodeLocally);
        writeCode.addNextNode(problemSpecification);
        writeCode.addNextNode(writeCode);
        writeCode.addNextNode(readCode);
        writeCode.addNextNode(readDocumentation);

        submitCode.addNextNode(testCodeLocally);
        submitCode.addNextNode(testCodeAutomatically);
        submitCode.addNextNode(problemSpecification);
        submitCode.addNextNode(writeCode);

        testCodeLocally.addNextNode(problemSpecification);
        testCodeLocally.addNextNode(taskDone);//test was success, however code
was not submitted and properly tested, developer needs to submit code manually
and verify its consistency

testCodeAutomatically.addNextNode(problemSpecification);
        testCodeAutomatically.addNextNode(taskDone);//test was success

getDocumentation.addNextNode(readDocumentation);
        getCode.addNextNode(readCode);

        readDocumentation.addNextNode(getDocumentation);
        readDocumentation.addNextNode(problemSpecification);
        readDocumentation.addNextNode(writeCode);

        readCode.addNextNode(getCode);
        readCode.addNextNode(problemSpecification);
        readCode.addNextNode(writeCode);
    }


}
```

A.32 Role.java

```
package CA.Entities.Role;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Logical.DataSource;
import CA.Entities.Physical.Location;
import CA.Processes.ProcessTree;
import CA.Processes.Task;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.Set;

/**
 * Created by root on 09/04/2015.
 */
public class Role {
    Location location;
    String RoleName = this.getClass().getName();
```

```
    CompUnit unitForWork = null;
    HashMap<String, DataSource>dataSources = new HashMap<String, DataSource>();


public Role(Location location) {
this.location = location;
    }

public Role(Location location, String roleName) {
this(location);
RoleName = roleName;
    }

public Location getLocation() {
return location;
    }

public void setLocation(Location location) {
this.location = location;
    }

public String getRoleName() {
return RoleName;
    }

public void setRoleName(String roleName) {
RoleName = roleName;
    }

public DataSource getDataSource(String name){
return dataSources.get(name);
    }

public void addDataSource(DataSource ds, String name){
dataSources.put(name,ds);
    }

public LinkedList<DataSource> getDataSourceWithCompUnit(CompUnit cu){
        LinkedList<DataSource> result = new LinkedList<>();
for(String s:dataSources.keySet())
if (dataSources.get(s).getLocation()==cu) result.add(dataSources.get(s));
return result;
    }

    LinkedList<Task>tasks = new LinkedList<>();

public LinkedList<Task> getTasks() {
return tasks;
    }

public void addTask(Task task) {
this.tasks.add(task);
    }
}
```

A.33 Browser.java

```
packageCA.Entities.Software;

import CA.Entities.Hardware.CompUnit;
```

```
import CA.Entities.Logical.Message;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 08/04/2015.
 */
public class Browserextends Component {
private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Browser.class);

public Browser(OperatingSystem systemComponentIsInstalledOn) {
super(systemComponentIsInstalledOn);
parent.addChild(this,"Browser");
    }

@Override
public boolean messageArrived(Message m) {
logger.info("message has arrived to browser:"+m);
return true;
    }

public void sendMessage(Message m) {
parent.sendMessage(m,true);
    }

public void sendMessage(CompUnit target, int messageType, String command, String
targetName){
        Message m = new Message(target,messageType,command,targetName);
        sendMessage(m);
    }

}
```

A.34 Component.java

```
packageCA.Entities.Software;

import CA.Entities.Logical.Message;
import CA.lib.Entity;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 12/04/2015.
 */
public abstract class Component implements Entity {
private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Component.class);
    OperatingSystem parent;
public boolean messageArrived(Message m){
logger.info("message has arrived to component:"+m);
return false;
    }

public Component(OperatingSystem parent) {
this.parent = parent;
    }

public OperatingSystem getParent() {
return parent;
```

```
    }
}
```

A.35 DatabaseServer.java

```
packageCA.Entities.Software;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public class DatabaseServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(DatabaseServer.class);
static int t = 0;

    HashMap<String, HashMap<String, Record>>data = new HashMap<String,
HashMap<String, Record>>();
public static String name = "DatabaseServer";

public static String listCommand = "list";
public static String listParameter = "value";
public static String listResponse = "response";
public static String listFail = "not found";

public static String saveCommand = "save";
public static String saveParameter = "value";
public static String saveParameterRecord = "record";
public static String saveResponse = "saved";
public static String saveFail = "failed";

public DatabaseServer(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent, passwordFile);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                Record r = getRecord(s.getUsername(),
m.getParameterString(listParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
logger.warn("failed to locate requested record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
```

```
listResponse, m.getFromPort());
m.setParameter(listResponse, r);
logger.warn("located record for:"+m);
parent.sendMessage(response, true);
return true;
                }
            }
if (m.getCommand().equals(saveCommand)) {
                Session s = getSession(m.getSessionID());



addRecord(s.getUsername(),m.getParameterString(saveParameter),m.getParameterRecor
d(saveParameterRecord));

                Record r = getRecord(s.getUsername(),
m.getParameterString(saveParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
saveFail, m.getFromPort());
logger.warn("failed to  save requested record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
saveResponse, m.getFromPort());
m.setParameter(saveResponse, r);
logger.warn("saved record for:"+m);
parent.sendMessage(response, true);
return true;
                }
            }
        }

return false;
    }

    Record getRecord(String username, String recordName) {
        HashMap<String, Record> records = data.get(username);
if (records == null) return null;
return records.get(recordName);
    }

    HashMap<String, Record> getRecords(String userName){
return  data.get(userName);
    }

public void addRecord(String username, String recordName, Record r){
if (!data.containsKey(username)){
data.put(username, new HashMap<>());
        }
        HashMap<String, Record> records = data.get(username);
        records.put(recordName,r);
    }

public Record removeRecord(String username, String recordName){
if (!data.containsKey(username)) return null;
return data.get(username).remove(recordName);
```

```
    }
}
```

A.36 Firewall.java

```
packageCA.Entities.Software;

import CA.Entities.AccessControl.FirewallDecisionMaker;
import CA.Entities.AccessControl.FirewallPolicy;
import CA.Entities.Logical.Message;
import CA.engine.Main;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 12/04/2015.
 */
public class Firewall extends Component {
private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Firewall.class);
public static final String name = "Firewall";
private FirewallDecisionMaker dm = new FirewallDecisionMaker();

//    static int a=0;

public Firewall(OperatingSystem parent) {
super(parent);
        parent.addChild(this,name);
    }

public void addNewPolicy(FirewallPolicy fp){
dm.addPolicy(fp);
    }

public boolean isAllowedToPass(Message m){
logger.debug("Firewall making decision");
boolean result = dm.allowAccess(m);
if (result)
logger.debug("Firewall granted access");
else
logger.info("Firewall denied access");
return result;

}
}
```

A.37 FTPServer.java

```
packageCA.Entities.Software;

import CA.Entities.AccessControl.ServerLocalAccessControl;
import CA.Entities.Hardware.HardDrive;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 15/04/2015.
 */
public class FTPServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
```

```
LogManager.getLogger(FTPServer.class);
static int t = 0;

public static String name = "FTPServer";
public static String listCommand = "list";
public static String listParameter = "value";
public static String listParameterRecord = "recordToRead";
public static String listResponse = "response";
public static String listFail = "not found";

public static String saveCommand = "save";
public static String saveParameter = "value";
public static String saveParameterRecord = "recordToSave";
public static String saveFail = "not saved";

public static String saveResponse = "saved";



public FTPServer(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent, passwordFile);
dm = new ServerLocalAccessControl(true);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                FileWithRecords fwr = getFile(s.getUsername(),
m.getParameterString(listParameter));

if (fwr == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
logger.warn("failed to locate requested record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
listResponse, m.getFromPort());
if (m.getParameter(listParameterRecord)!=null) {

response.setParameter(listParameter,fwr.getRecord((String)
m.getParameter(listParameterRecord)));
                    } else
response.setParameter(listParameter, fwr);
logger.info("located record for:" + m);
parent.sendMessage(response, true);
return true;
                }
            }
if (m.getCommand().equals(saveCommand)) {
                Session s = getSession(m.getSessionID());

                FileWithRecords fwr = getFile(s.getUsername(),
m.getParameterString(saveParameter));
if (fwr==null){
```

```
                    fwr =
parent.createNewFile(m.getParameterString(saveParameter));
                }
                fwr.addRecord(m.getParameterRecord(saveParameterRecord));

if (fwr == null) {
                    Message response = new Message(m.getFrom(),  responseType,
saveFail, m.getFromPort());
logger.warn("failed to save record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
saveResponse, m.getFromPort());
response.setParameter(saveParameter, fwr);
logger.info("saved record for:"+m);
parent.sendMessage(response, true);
return true;
      }
            }
         }
return false;
    }

    FileWithRecords getFile(String username, String recordName) {
        HardDrive hd = getParent().getDrive(recordName);
return hd.getRecords(recordName);
    }

public void addFile(String username, String recordName, FileWithRecords fwr){
        HardDrive hd = getParent().getDrive(recordName);
        hd.addRecord(fwr,recordName);
    }

public FileWithRecords removeFile(String username, String recordName){
        HardDrive hd = getParent().getDrive(recordName);
return hd.removeFile(recordName);
    }
}
```

A.38 Listener.java

```
packageCA.Entities.Software;

import CA.Entities.Logical.Message;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 11/06/2015.
 */
public abstract class Listener extends Component{

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Listener.class);

public Listener(OperatingSystem parent) {
super(parent);
    }

public boolean messageArrived(Message m){
```

```
logger.debug("message has arrived to listener:"+m);
        processMessage(m);
return false;
    }

public void processMessage(Message m){

    };
}
```

A.39 MailServer.java

```
package CA.Entities.Software;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.EmailRecord;
import CA.Entities.Logical.Records.Record;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public class MailServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(MailServer.class);
static int t = 0;

    HashMap<String, HashMap<String, EmailRecord>>data = new HashMap<String,
HashMap<String, EmailRecord>>();


public static String name = "MailServer";
public static String listCommand = "list";
public static String listParameter = "value";
public static String listResponse = "response";
public static String listFail = "not found";

public MailServer(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent, passwordFile);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                Record r = getRecord(s.getUsername(),
m.getParameterString(listParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
logger.warn("failed to locate requested record for:"+m);
parent.sendMessage(response, false);
```

```
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
listResponse, m.getFromPort());
response.setParameter(listParameter, r);
logger.info("located record for:"+m+" param is:"+r+" response is:"+response);
parent.sendMessage(response, false);
return true;
                }
            }
        }

return false;
    }

    Record getRecord(String username, String recordName) {
        HashMap<String, EmailRecord> records = data.get(username);
if (records == null) return null;
return records.get(recordName);
    }

    HashMap<String, EmailRecord> getRecords(String userName){
return  data.get(userName);
    }

public void addRecord(String username, String recordName, EmailRecord r){
if (!data.containsKey(username)){
data.put(username, new HashMap<>());
        }
        HashMap<String, EmailRecord> records = data.get(username);
        records.put(recordName, r);
        r.setStored(this);
    }

public Record removeRecord(String username, String recordName){
if (!data.containsKey(username)) return null;
return data.get(username).remove(recordName);
    }

public String findNameByRecord(EmailRecord er){
for(String userNames:data.keySet()) {
            HashMap<String, EmailRecord> map = data.get(userNames);
for(String s:map.keySet())
if (map.get(s)==er) return s;
        }
return null;
    }
}
```

A.40 OperatingSystem.java

```
package CA.Entities.Software;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Hardware.HardDrive;
import CA.Entities.Hardware.NetworkingInterface;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Logical.FileWithRecords;
import CA.NotModel.RiskCollection;
```

```
import CA.engine.ActionsQueue;
import CA.engine.Main;
import CA.lib.Entity;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Future;

import static CA.lib.Lib.VERBOSE;

/**
 * Created by root on 08/04/2015.
 */
public class OperatingSystem implements Entity {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(OperatingSystem.class);
    CompUnit runningOn;
    HashMap<String,Component>children = new HashMap<String, Component>();
    HashMap<String, LinkedList<Object>>listeningThreads = new HashMap<String,
LinkedList<Object>>();
public static final int READ_FILE=0;
public static final int WRITE_FILE=1;
public static final int EXECUTE_FILE=2;

public Future<?> sendMessage(final Message m){
        m.setFrom(this.runningOn);
logger.log(VERBOSE, ("sending message from:"+m.getFrom())+" to:"+m.getTo());
        m.setFrom(runningOn);
        LinkedList<NetworkingInterface> nis = runningOn.getNetworkConnections();
for (final NetworkingInterface ni:nis) {
final LinkedList<NetworkingInterface> path = ni.buildPath(m.getFrom(),
m.getTo());
if (path!=null){
                m.setPath(path);
                Runnable r = new Runnable() {
@Override
public void run() {
do{
m.getCurrentLocation().getInstalledOn().getOs().passToListeners(m);

if (m.getCurrentLocation().getInstalledOn().getOs().canMessagePass(m)) {


ActionsQueue.currentQueue.addNewEvent(ActionsQueue.currentQueue.getCurrentTime()
+ 1);
logger.log(VERBOSE, "passing node:" + m.getCurrentLocation());
                            } else{
logger.warn("message was not allowed to pass:" + m);
break;
                            }
}while (m.traverse());
if (m.getCurrentLocation().getConnectedTo().getInstalledOn().equals(m.getTo())) {
logger.debug("message has arrived:" +
m.getCurrentLocation().getConnectedTo().getInstalledOn());
m.getCurrentLocation().getConnectedTo().getInstalledOn().getOs().messageArrived(m
);
```

```
                        }
                    }
                };
return ActionsQueue.currentQueue.start(r);
            } else
{
                m.failedToBeDelivered = true;
                m.noRouteForMessage=true;
            }

        }
return null;
    }

public void passToListeners(Message m){
for(String name:children.keySet()){
            Component c = children.get(name);
if (c instanceof Listener){
                ((Listener) c).messageArrived(m);
            }
        }
    }


public Future<?> sendMessage(final Message m, boolean wait){
        Future<?> f = sendMessage(m);
if (wait)
try {
                    f.get();
                } catch (InterruptedException | ExecutionException e) {
                    e.printStackTrace();
                }
return f;
}

public boolean canMessagePass(Message m){
if (children.get("Firewall")!=null){
            Firewall f = (Firewall)children.get("Firewall");
return f.isAllowedToPass(m);
        }
return true;
    }

public void messageArrived(Message m){
        passToListeners(m);
        String componentName = m.getTargetName();
if (componentName==null) {
logger.debug("component name is null, number of listening threads:" +
listeningThreads.size() + " and number of children:" + children.size());
if (listeningThreads.size() == 1 &&children.size() == 0) {
logger.debug("forwarding message to the only thread");
listeningThreads.entrySet().iterator().next().getValue().addLast(m);
synchronized (((ExecutorService)
listeningThreads.entrySet().iterator().next().getValue().get(0))) {
                    ((ExecutorService)
listeningThreads.entrySet().iterator().next().getValue().get(0)).notify();
                }
return;
            }
if (children.size() == 1 &&listeningThreads.size() == 0) {
```

```
if (children.size() == 1) {
logger.debug("forwarding message to the only component");
children.get(children.keySet().iterator().next()).messageArrived(m);
return;
                }
            }
        }
if (children.get(componentName)!=null){
//message is properly delivered
children.get(componentName).messageArrived(m);
return;
        }
if (listeningThreads.containsKey(componentName)){
listeningThreads.get(componentName).add(m);
            Object o = listeningThreads.get(componentName).get(0);
synchronized (o) {
                o.notify();
            }
return;
        }
if (children.size()==1&&listeningThreads.size() == 0){
logger.debug("forwarding message to the only component");
children.get(children.keySet().iterator().next()).messageArrived(m);
return;
        }
if (listeningThreads.size() == 1 &&children.size() == 0) {
logger.debug("forwarding message to the only thread");
listeningThreads.entrySet().iterator().next().getValue().addLast(m);
            ExecutorService es = ((ExecutorService)
listeningThreads.entrySet().iterator().next().getValue().get(0));
synchronized (es){
                es.notify();
                ((ExecutorService)
listeningThreads.entrySet().iterator().next().getValue().get(0)).notify();
            }
return;
        }
logger.warn("could not get destination, message is ignored");
    }

public OperatingSystem(CompUnit runningOn) {
this.runningOn = runningOn;
    }

public void addChild(Component component, String name){
children.put(name,component);
    }

public FileWithRecords getFile(String fileName){
String driveName = fileName.substring(0,2);
        LinkedList<HardDrive> drives = runningOn.getHardDrives();
for(HardDrive hd:drives){
if (hd.getDriveName().equals(driveName))
return hd.getRecords(fileName);
        }
return null;
    }

public Record getRecord(String fileName, String recordName){
return getFile(fileName).getRecord(recordName);
```

```
    }

public boolean getAccessControlDecision(int accessType, String fileName){
return true;
    }

public Message listenAtSpecificPort(String port){


LinkedList<Object> objects = new LinkedList<Object>();
        Object o = listeningThreads.get(port);
if (o!=null){
logger.warn("busy port at:"+port);
return null;
        }
logger.debug("listening at:" + port + " at " + runningOn);
listeningThreads.put(port,objects);

ActionsQueue.currentQueue.addNewEventWithInterruption(ActionsQueue.currentQueue.g
etCurrentTime() + 500 * 1000,objects);
if (objects.size()>1){
            Object o1 = objects.get(0);
synchronized (o1){
                o1.notify();
            }
logger.debug("got message at:"+port);
            doneListeningAtPort(port);
return (Message) objects.get(1);//message arrived
}
logger.info("timeout at:"+port+" at "+runningOn);
        doneListeningAtPort(port);
return null;
    }

public void doneListeningAtPort(String portName){
listeningThreads.remove(portName);
logger.debug("removed at:"+portName+" at "+runningOn);
    }

public void messageArrived(Message m, String threadName){
listeningThreads.get(threadName).add(m);
    }

public CompUnit getRunningOn() {
return runningOn;
    }

public HardDrive getDrive(String name){
for(HardDrive hd:runningOn.getHardDrives())
if (name.toLowerCase().startsWith(hd.getDriveName().toLowerCase())) return hd;
return null;
    }

/**
     *
     * @param newFile
* @return true if file was created, false if there was an error
     */
public boolean createNewFile(FileWithRecords newFile){
        HardDrive hd = runningOn.getHardDrives().get(0);
```

```
if (newFile.getFilename().contains(":")){
            hd = getDrive(new String(newFile.getFilename()).substring(0,2));

hd.addRecord(newFile,newFile.getFilename().substring(newFile.getFilename().indexO
f('\\')+1));
        }else{
            hd.addRecord(newFile,newFile.getFilename());
        }
return true;
    }

public LinkedList<Component> getChildren(){
        LinkedList<Component> result = new LinkedList<>();
for(String s:children.keySet())
            result.add(children.get(s));
return result;
    }

public Component getChild(String name){
return children.get(name);
    }

public FileWithRecords createNewFile(String fullName){
        FileWithRecords fwr = new FileWithRecords(fullName,runningOn);
if (!createNewFile(fwr)) fwr = null;
return fwr;
    }


}
```

A.41 RemoteControlServer.java

```
packageCA.Entities.Software;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public class RemoteControlServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(RemoteControlServer.class);
static int t = 0;

    HashMap<String, HashMap<String, Record>>data = new HashMap<String,
HashMap<String, Record>>();

public static String name = "RemoteControlServer";
public static String listCommand = "list";
public static String listParameter = "value";
public static String listResponse = "response";
public static String listFail = "not found";
```

```
public RemoteControlServer(OperatingSystem parent, FileWithRecords passwordFile)
{
super(parent, passwordFile);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                Record r = getRecord(s.getUsername(),
m.getParameterString(listParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
logger.warn("failed to locate requested record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
listResponse, m.getFromPort());
m.setParameter(listCommand, r);
logger.warn("located record for:"+m);
parent.sendMessage(response, true);
return true;
      }
            }
        }
return false;
    }

    Record getRecord(String username, String recordName) {
        HashMap<String, Record> records = data.get(username);
if (records == null) return null;
return records.get(recordName);
    }

    HashMap<String, Record> getRecords(String userName){
return  data.get(userName);
    }

public void addRecord(String username, String recordName, Record r){
if (!data.containsKey(username)){
data.put(username, new HashMap<>());
        }
        HashMap<String, Record> records = data.get(username);
        records.put(recordName,r);
    }

public Record removeRecord(String username, String recordName){
if (!data.containsKey(username)) return null;
return data.get(username).remove(recordName);
    }
}
```

A.42 Server.java

```
package CA.Entities.Software;

import CA.Entities.AccessControl.ServerLocalAccessControl;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.engine.ActionsQueue;
import CA.engine.Random;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public abstract class Server extends Component{

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Server.class);
public static final String authorize = "auth";
public static final String authorizeParamUsername = "username";
public static final String authorizeParamPassword = "password";
public static final String authorizeSuccess = "ack";
public static final String authorizeFail = "denied";

public static final String accessControlFail = "AC denied";

    HashMap<String,Session>sessions = new HashMap<String, Session>();
int responseType = Message.HTTP;
    ServerLocalAccessControl dm = null;

//String DataFileName
public Server(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent);
dm = new ServerLocalAccessControl(passwordFile,this);

    }

public Server(OperatingSystem parent, String passwordFileName) {
super(parent);
dm = new ServerLocalAccessControl(passwordFileName,this);
    }

/**
     * Genetic message processing for servers
     * @param m message
     * @return true if message was processed with generic approach
     */
public boolean messageArrived(Message m) {
logger.debug("message has arrived to " + this.getClass() + ":" + m);
if (m.getCommand()==null){
logger.debug("Command in Message is empty, ignoring");
return true;
        }
if (!dm.allowAccess(m)){
if (m.getCommand().equals(authorize)) {
                Message response = new Message(m.getFrom(), responseType,
authorizeFail, m.getFromPort());
logger.debug("Authorization request resolved with " + "authorizeFail");
```

```
parent.sendMessage(response);

} else {
                Message response = new Message(m.getFrom(), responseType,
accessControlFail, m.getFromPort());
logger.info("Authorization control request answered with " +
"accessControlFail");
parent.sendMessage(response);
return true;
            }
        }
if (m.getCommand().equals(authorize)){
logger.debug("Authorization request arrived to:" + this.getClass());
if (dm.isAllowNotAuthorizedAccess()){
                Message response = new
Message(m.getFrom(),responseType,authorizeSuccess,m.getFromPort());
logger.debug("Authorization request resolved with " + "authorizeSuccess");
                Session session = new Session(Random.getNewID(),
"No Authentication",
                        m.getParameterString(authorizeParamUsername),
                        ActionsQueue.currentQueue.getCurrentTime(),
                        ActionsQueue.currentQueue.getCurrentTime()+20000);
sessions.put(session.getSessionID(),session);
                response.setSessionID(session.getSessionID());
parent.sendMessage(response);
return true;
            }
if
(dm.passwordsMatch(m.getParameterString(authorizeParamUsername),m.getParameterStr
ing(authorizeParamPassword))){
                Message response = new
Message(m.getFrom(),responseType,authorizeSuccess,m.getFromPort());
logger.debug("Authorization request resolved with " + "authorizeSuccess");
                Session session = new Session(Random.getNewID(),
dm.getAccountRole(m.getParameterString(authorizeParamUsername)),
                        m.getParameterString(authorizeParamUsername),
                        ActionsQueue.currentQueue.getCurrentTime(),
                        ActionsQueue.currentQueue.getCurrentTime()+20000);
sessions.put(session.getSessionID(),session);
                response.setSessionID(session.getSessionID());
parent.sendMessage(response);
return true;
            } else{
                Message response = new
Message(m.getFrom(),responseType,authorizeFail,m.getFromPort());//password is
wrong
logger.info("Authorization request resolved with " + "authorizeFail");
parent.sendMessage(response);
return true;
            }
        }
return false;

    }

public Session getSession(String name){
return sessions.get(name);
    }
```

```
}
```

A.43 Session.java

```
packageCA.Entities.Software;

import CA.Entities.Role.Role;

/**
 * Created by root on 22/04/2015.
 */
public class Session {
    String sessionID;
    String userRole;
    String username;
long timeSessionStarted;
long timeSessionExpires;

public Session(String sessionID, String userRole, String username, long
timeSessionStarted, long timeSessionExpires) {
this.sessionID = sessionID;
this.userRole = userRole;
this.timeSessionStarted = timeSessionStarted;
this.timeSessionExpires = timeSessionExpires;
this.username = username;
    }

public String getUsername() {
return username;
    }

public String getSessionID() {
return sessionID;
    }

public void setSessionID(String sessionID) {
this.sessionID = sessionID;
    }

public String getUserRole() {
return userRole;
    }

public void setUserRole(String userRole) {
this.userRole = userRole;
    }

public long getTimeSessionStarted() {
return timeSessionStarted;
    }

public void setTimeSessionStarted(long timeSessionStarted) {
this.timeSessionStarted = timeSessionStarted;
    }

public long getTimeSessionExpires() {
return timeSessionExpires;
    }
```

```
public void setTimeSessionExpires(long timeSessionExpires) {
this.timeSessionExpires = timeSessionExpires;
    }
}
```

A.44 SourceCodeWebServer.java

```
package CA.Entities.Software;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Logical.RecordsCollection;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public class SourceCodeServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(SourceCodeServer.class);

    HashMap<String, FileWithRecords>data = new HashMap<String,
FileWithRecords>();

public static String name = "SourceCodeServer";

public static String listCommand = "list";
public static String listParameter = "value";
public static String listResponse = "response";
public static String listFail = "not found";

public static String saveCommand = "save";
public static String saveParameterFileName = "filename";
public static String saveParameterRecordName = "recordname";
public static String saveParameterRecord = "record";
public static String saveResponse = "saved";
public static String saveFail = "not found";

public SourceCodeServer(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent, passwordFile);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                FileWithRecords r =
getRecords(m.getParameterString(listParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
```

```
logger.warn("failed to locate requested record for:" + m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
listResponse, m.getFromPort());
m.setParameter(listCommand, r);
logger.warn("located record for:" + m);
parent.sendMessage(response, true);
return true;
                }
            }
if (m.getCommand().equals(saveCommand)) {
                Session s = getSession(m.getSessionID());
                String fileName = m.getParameterString(saveParameterFileName);
                String recordName =
m.getParameterString(saveParameterRecordName);

                FileWithRecords fwr = getRecords(fileName);
if (fwr==null){
                    fwr = new
FileWithRecords("c:\\CVS\\"+fileName,parent.getRunningOn());
data.put(fwr.getFilename(),fwr);
                }

fwr.addRecord(m.getParameterRecord(saveParameterRecord),recordName);

                Message response = new Message(m.getFrom(), responseType,
saveResponse, m.getFromPort());
m.setParameter(saveCommand, fwr);
logger.warn("saved record for:" + m);
parent.sendMessage(response, true);
return true;
            }
        }
return false;
    }

    FileWithRecords getRecords(String fileName) {
return data.get(fileName);
    }

public void addRecords(String fileName, FileWithRecords r) {
data.put(fileName,r);
    }

public void addRecords(FileWithRecords r) {
data.put(r.getFilename(),r);
    }


public FileWithRecords removeRecord(String fileName) {
return data.remove(fileName);
    }



}
```

A.45 WebServer.java

```
packageCA.Entities.Software;

import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import org.apache.logging.log4j.LogManager;

import java.util.HashMap;

/**
 * Created by root on 15/04/2015.
 */
public class WebServer extends Server {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(WebServer.class);

    HashMap<String, HashMap<String, Record>>data = new HashMap<String,
HashMap<String, Record>>();

public static String name = "WebServer";

public static String listCommand = "list";
public static String listParameter = "value";
public static String listResponse = "response";
public static String listFail = "not found";

public static String saveCommand = "save";
public static String saveParameter = "value";
public static String saveParameterRecord = "record";
public static String saveResponse = "response";
public static String saveFail = "failed";

public WebServer(OperatingSystem parent, FileWithRecords passwordFile) {
super(parent, passwordFile);
        parent.addChild(this, name);
    }

public boolean messageArrived(Message m) {
boolean bool = super.messageArrived(m);
if (bool) return bool;
if (sessions.containsKey(m.getSessionID())) {
if (m.getCommand().equals(listCommand)) {
                Session s = getSession(m.getSessionID());
                Record r = getRecord(s.getUsername(),
m.getParameterString(listParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
listFail, m.getFromPort());
logger.warn("failed to locate requested record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
listResponse, m.getFromPort());
m.setParameter(listCommand, r);
logger.warn("located record for:"+m);
```

```
parent.sendMessage(response, true);
return true;
                }
            }
if (m.getCommand().equals(saveCommand)) {
                Session s = getSession(m.getSessionID());
                addRecord(s.getUsername(), m.getParameterString(saveParameter),
m.getParameterRecord(saveParameterRecord));
                Record r = getRecord(s.getUsername(),
m.getParameterString(saveParameter));

if (r == null) {
                    Message response = new Message(m.getFrom(), responseType,
saveFail, m.getFromPort());
logger.warn("failed to save record for:"+m);
parent.sendMessage(response, true);
return true;
                } else {
                    Message response = new Message(m.getFrom(), responseType,
saveResponse, m.getFromPort());
m.setParameter(saveParameter, r);
logger.warn("saved record for:"+m);
parent.sendMessage(response, true);
return true;
                }
            }
        }

return false;
    }

    Record getRecord(String username, String recordName) {
        HashMap<String, Record> records = data.get(username);
if (records == null) return null;
return records.get(recordName);
    }

    HashMap<String, Record> getRecords(String userName){
return  data.get(userName);
    }

public void addRecord(String username, String recordName, Record r){
if (!data.containsKey(username)){
data.put(username, new HashMap<>());
        }
        HashMap<String, Record> records = data.get(username);
        records.put(recordName,r);
    }

public Record removeRecord(String username, String recordName){
if (!data.containsKey(username)) return null;
return data.get(username).remove(recordName);
    }


}
```

A.46 Entity.java

```
packageCA.lib;

/**
 * Created by root on 08/04/2015.
 */
public interface Entity {

}
```

A.47 FileNameCleaner.java

```
packageCA.lib;

import java.util.Arrays;

/**
 * Created by root on 28/06/2015.
 */
public class FileNameCleaner {
final static int[] illegalChars = {34, 60, 62, 124, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 58, 42, 63, 92, 47};
static {
        Arrays.sort(illegalChars);
    }
public static String cleanFileName(String badFileName) {
        StringBuilder cleanName = new StringBuilder();
for (int i = 0; i < badFileName.length(); i++) {
int c = (int)badFileName.charAt(i);
if (Arrays.binarySearch(illegalChars, c) <0) {
                cleanName.append((char)c);
            }
        }
return cleanName.toString();
    }
}
```

A.48 Lib.java

```
packageCA.lib;

import org.apache.logging.log4j.Level;

/**
 * Created by root on 27/04/2015.
 */
public class Lib {
public static final Level VERBOSE = Level.forName("VERBOSE", 650);
public static final Level TRACE = Level.TRACE;
public static final Level DEBUG = Level.DEBUG;
public static final Level INFO = Level.INFO;
public static final Level WARN = Level.WARN;
public static final Level ERROR = Level.ERROR;
public static final Level FATAL = Level.FATAL;


}
```

A.49 NamedEnvironment.java

```
packageCA.lib;

import CA.Entities.Logical.Records.Record;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Created by root on 16/04/2015.
 */
public class NamedEnvironment<T> {
    HashMap<String,T>records = new HashMap<String, T>();

public T getObject(String recordName){
return records.get(recordName);
    }

public void addObject(T r, String name){
records.put(name,r);
    }

public boolean removeObject(String name, T r) {
T o = records.remove(name);
if (o == null) {
for (Map.Entry<String, T> e : records.entrySet()) {
if (r == e.getValue()) {
records.remove(e.getKey());
return true;
                }
            }
return false;
        } else {
return true;
        }
    }

public boolean removeObject(T r) {
for (Map.Entry<String, T> e : records.entrySet()) {
if (r == e.getValue()) {
records.remove(e.getKey());
return true;
            }
        }
return false;
    }

public Set<String> getKeySet(){
return records.keySet();
    }

public boolean isObjectPresent(String name){
return records.containsKey(name);
    }
}
```

A.50 Pair.java

```
packageCA.lib;

/**
 * Created by root on 15/04/2015.
 */
public class Pair<V1,V2> {
public V1 a = null;
public V2 b = null;

public Pair(){

    }

public Pair(V1 a, V2 b) {
this.a = a;
this.b = b;
    }
}
```

A.51 Locator.java

```
package CA.NotModel;

import CA.Entities.Hardware.CompUnit;
import CA.Entities.Logical.DataSource;
import CA.Entities.Logical.FileWithRecords;
import CA.Entities.Logical.Records.EmailRecord;
import CA.Entities.Logical.Records.Record;
import CA.Entities.Role.Role;
import CA.Entities.Software.MailServer;
import CA.Entities.Software.Server;
import CA.Entities.Software.WebServer;
import CA.engine.Random;

import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Set;

/**
 * Created by root on 30/04/2015.
 */
public class Locator {
public DataSource source;
    String targetRecordName;
    String targetRecordCollectionName;


public static LinkedList<Locator> getShuffledRecordsLocations(Role role, Record
targetRecord){
        LinkedList<Locator> locators =
Locator.getRecordsLocations(role,targetRecord);
        LinkedList<Locator> shuffledLocators = new LinkedList<>();
for(int i=0;i<locators.size()-1;i++){

shuffledLocators.add(locators.remove(Random.getNext(locators.size())));
        }
        shuffledLocators.add(locators.remove());
return shuffledLocators;
```

```
    }

public static boolean isRecordHere(Role role, Record targetRecord, CompUnit cu){
        LinkedList<Locator> locators = getRecordsLocations(role,targetRecord);
for(Locator l:locators){
if (l.source.getLocation()==cu)
return true;
        }
return false;
    }

public static LinkedList<Locator> getRecordsLocations(Role role, Record
targetRecord, CompUnit dataSourceConnectToLocation){
        LinkedList<Locator> loc = getRecordsLocations(role,targetRecord);
        LinkedList<Locator> result = new LinkedList<>();
for(Locator l:loc){
if (l.source.getLocation()==dataSourceConnectToLocation)
                result.add(l);
        }
return result;
    }

public static LinkedList<Locator> getRecordsLocations(Role role, Record
targetRecord){
        LinkedList<Object> records = getRelatedRecords(targetRecord,10);
        LinkedList<Locator> result = new LinkedList<>();
for(Object r:records){

if (r instanceof EmailRecord){
                Locator l = new Locator();
                EmailRecord er = (EmailRecord) r;
                LinkedList<DataSource> dataSources =
role.getDataSourceWithCompUnit(er.getStored().getParent().getRunningOn());
                DataSource suitableDS = null;
for(DataSource ds:dataSources) {
if (ds.getType()==DataSource.TYPE_EMAIL_SERVER){
                        suitableDS = ds;
                    }
                }
if (suitableDS==null) continue;
                MailServer mailServer = (MailServer)
suitableDS.getLocation().getOs().getChild(MailServer.name);
if (mailServer==null) continue;//this is not ok, should not be
l.targetRecordName = mailServer.findNameByRecord(er);
                DataSource newDS = new DataSource(DataSource.TYPE_EMAIL_SERVER,
l.targetRecordName, er.getStored().getParent().getRunningOn(),
suitableDS.getUsername(),suitableDS.getPassword());
                l.source = newDS;
                result.add(l);
            }
if (r instanceof FileWithRecords){
                Locator l = new Locator();
                FileWithRecords fwr = (FileWithRecords) r;
                DataSource newDS = new
DataSource(DataSource.TYPE_FILE,fwr.getFilename(),fwr.getLocation(),null);
                l.source = newDS;
                l.targetRecordCollectionName = fwr.getFilename();
                l.targetRecordName = fwr.getNameOfRecord(targetRecord);
                result.add(l);
            }
```

```
        }
return result;
    }

static LinkedList<Object> getRelatedRecords(Record record, int depth){
if (depth==0) return new LinkedList<>();
        LinkedList<Object> result = new LinkedList<>();
        HashMap<String,Object> map = record.getReferences();
for(String s:map.keySet()){
            Object r = map.get(s);
if (!result.contains(r)) {
                result.add(r);
}
        }
return result;
    }

public DataSource getSource() {
return source;
    }

public String getTargetRecordName() {
return targetRecordName;
    }

public String getTargetRecordCollectionName() {
return targetRecordCollectionName;
    }
}
```

A.52 Risk.java

```
packageCA.NotModel;

import CA.Entities.Logical.Records.Record;
import CA.Entities.Software.Component;

/**
 * Created by root on 30/04/2015.
 */
public class Risk {
/**
     * Value of risk
     */
double value=0;

/**
     * Object this risk is associated with
     */
Record object;

    Component riskRelatedTo;

    String comment;

long time;

int type;

public Risk(double value, Record object, Component riskRelatedTo, String comment,
```

```
long time, int type) {
this.value = value;
this.object = object;
this.riskRelatedTo = riskRelatedTo;
this.comment = comment;
this.time = time;
this.type = type;
    }

public Risk(double value, long time, Record object, int type) {
this.value = value;
this.time = time;
this.object = object;
this.type = type;
    }

public Risk(double value, long time, int type) {
this.value = value;
this.time = time;
this.type = type;
    }

public String toString(){
        String res = "";
switch (type){
case RiskCollection.TYPE_PRODUCTIVITY:{
                res+="P Risk";
break;
            }
case RiskCollection.TYPE_SECURITY:{
                res+="S Risk";
break;
            }
default:{
                res+="Unknown Risk";
break;
            }
        }
        res+="; Value="+value;
if (object!=null){
            res+=";Related to "+object;
        }
return res;
    }
}
```

A.53 RiskCollection.java

```
packageCA.NotModel;

import CA.Entities.Logical.Records.Record;
import CA.engine.ActionsQueue;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.*;

/**
 * Created by root on 03/05/2015.
 */
```

```
public class RiskCollection {

private static final Logger logger = LogManager.getLogger(RiskCollection.class);

public static RiskCollectioncurrentRisks = new RiskCollection();
public static final int TYPE_SECURITY=1;
public static final int TYPE_PRODUCTIVITY=2;


    List<Risk>risks;

public RiskCollection(){
risks = Collections.synchronizedList(new LinkedList<Risk>());
    }

//    public static RiskCollection allRisks;

public static synchronized void addSecurityRisk(double value, long time, Record
object){
if (value>1)
logger.warn("Risk value is more than one,value:"+value+" time:"+ time+ "
Object:"+object);
        Risk r = new Risk(value,time,object,TYPE_SECURITY);
currentRisks.risks.add(r);
    }

public static synchronized  void addSecurityRisk(double value){
addSecurityRisk(value, ActionsQueue.currentQueue.getCurrentTime());
    }

public static synchronized  void addProductivityRisk(double value){
addProductivityRisk(value, ActionsQueue.currentQueue.getCurrentTime());
    }

public static synchronized  void addSecurityRisk(Double value, long time){
addSecurityRisk(value,time,null);
    }

public static synchronized  void addProductivityRisk(Double value, long time){
        Risk r = new Risk(value,time,TYPE_PRODUCTIVITY);
currentRisks.risks.add(r);
    }

public static synchronized  double getSummedRiskValue(int type){
return getSummedRiskValue(type,Long.MAX_VALUE);
    }

/**
     * Security risk description:
     * Each transition of record or local query for r/w/x has a
     */

    /**
     * Productivity risk description:
     * Task has expected duration-Tex
     * Time to actually complete task Tact
     * Productivity Risk = (Tact-Tex)/Tex, can be negative and more than 1, the
lower the better
     */
```

```
public static synchronized  double getSummedRiskValue(int type, long time){
double res = 0;
        HashMap<Record,LinkedList<Double>> addedValues = new HashMap<>();
for(Risk r:currentRisks.risks){
if (r.time>time) continue;
if (r.type==type) {
if (r.object ==null)
                    res+=r.value;
else{
if (addedValues.containsValue(r.value)){
                        addedValues.get(r.object).add(r.value);
                    }else{
                        LinkedList<Double> list = new LinkedList<>();
                        list.add(r.value);
                        addedValues.put(r.object,list);
                    }
                }
            }
        }
for(Record r:addedValues.keySet()){
double mul = 1;
if (r.getSecurityValue()!=null){
                mul = r.getSecurityValue();
            }
            res+=mul*computeLikelihood(addedValues.get(r));
        }
return res;
    }

public static synchronized  TreeSet<Long> getTimepoints(){
        TreeSet<Long> res = new TreeSet<>();
for(Risk r:currentRisks.risks)
            res.add(r.time);
return res;
    }

public static synchronized  double computeLikelihood(LinkedList<Double>
probabilitiesList){
double mul = 1;
for(Double d:probabilitiesList){
            mul*=(1-d);
        }
return 1-mul;
    }

public static synchronized  void purgeRisks(){
currentRisks.risks = Collections.synchronizedList(new LinkedList<Risk>());
    }
}
```

A.54 Focus.java

```
packageCA.Processes;

import CA.engine.ActionsQueue;
import CA.lib.NamedEnvironment;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.LinkedList;
```

```
/**
 * Created by root on 12/04/2015.
 */
public class Focus implements Runnable{

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Focus.class);
    NamedEnvironment<Object>variables = new NamedEnvironment<>();
    TaskResults tasksExecuted = new TaskResults();
    TaskResults callStack = new TaskResults();
    ProcessTree tree;

public Focus(ProcessTree tree) {
this.tree = tree;
    }

@Override
public void run() {
        Task current = tree.chooseEntryPoint().clone();
do {
logger.info("Running:
Time"+ActionsQueue.currentQueue.getCurrentTime()+";Task:"+current.taskName);

ActionsQueue.currentQueue.addNewEvent(ActionsQueue.currentQueue.getCurrentTime()
+ current.actualDuration);
int taskStatus = current.runTask(this);

tasksExecuted.add(new TaskResult(current,taskStatus));

            Task chosenOne = current.chooseNext(tasksExecuted,callStack,this);
callStack.addToStack(current,taskStatus);

if (chosenOne!=null)
                current = chosenOne.clone();
else
current = null;
        } while (current!=null);
    }

public TaskResults getTasksExecuted() {
return tasksExecuted;
    }

public Object getVariable(String name){
return variables.getObject(name);
    }

public void setVariable(String name, Object object){
variables.addObject(object,name);
    }

public boolean getBooleanVariable(String name){
        Boolean b = (Boolean) variables.getObject(name);
if (b==null||!b) return false;
return true;
    }
}
```

A.55 ProcessTree.java

```
packageCA.Processes;

import CA.engine.Random;

import java.util.LinkedList;

/**
 * Created by root on 10/04/2015.
 */
public class ProcessTree {
    String name;
    LinkedList<Task>entryPoints = new LinkedList<Task>();
    LinkedList<Task>exitPoints = new LinkedList<Task>();

public ProcessTree(String name) {
this.name = name;
    }

public void addEntryPoint(Task task){
entryPoints.add(task);
    }

public void addExitPoint(Task task){
exitPoints.add(task);
    }

public Task chooseEntryPoint(){
return entryPoints.get(Random.getNext(entryPoints.size()));
    }
}
```

A.56 Task.java

```
packageCA.Processes;

import CA.engine.Random;

import java.util.LinkedList;
import CA.Processes.TaskResult.*;
import org.apache.logging.log4j.LogManager;

/**
 * Created by root on 10/04/2015.
 */
public class Taskimplements Cloneable{//implements Runnable {

private static final org.apache.logging.log4j.Logger logger =
LogManager.getLogger(Task.class);

    String taskName;
int standardDuration=10;
int actualDuration=10;
    Object parent;
int status;

    LinkedList<Task>nextNodes = new LinkedList<Task>();
    LinkedList<Task>previousNodes = new LinkedList<Task>();

public Task(String taskName) {
```

```
this.taskName = taskName;
    }

public Task(String taskName, int standardDuration) {
this.taskName = taskName;
        setStandardDuration(standardDuration);
    }

public Task(String taskName, int standardDuration, Object parent) {
this(taskName,standardDuration);
        setParent(parent);
    }

public Task chooseNext(TaskResults executedSoFar,TaskResults callStack, Focus f){
Task task = chooseCustomNext(executedSoFar,callStack,nextNodes,f);
if (task!=null) return task;
if (nextNodes.size()==0) return null;
return nextNodes.get(Random.getNext(nextNodes.size()));
    }

public Task chooseCustomNext(TaskResults taskList, TaskResults
callStack,LinkedList<Task> nextNodes, Focus f){
return null;
    }


public final int runTask(Focus f) {
        taskActions(f);
return status;
    }

public void taskActions(Focus f){

    }

public Task clone(){
try {
return (Task) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
return null;
    }

public void addNextNode(Task task){
if (task==null) return;
nextNodes.add(task);
        task.previousNodes.add(this);
    }

public int getStandardDuration() {
return standardDuration;
    }

public void setStandardDuration(int standardDuration) {
this.standardDuration = standardDuration;
this.actualDuration = standardDuration;
    }

public int getActualDuration() {
```

```
return actualDuration;
    }

public void setActualDuration(int actualDuration) {
this.actualDuration = actualDuration;
    }

public Object getParent() {
return parent;
    }

public void setParent(Object parent) {
this.parent = parent;
    }

public int getStatus() {
return status;
    }

public void setStatus(int status) {
this.status = status;
    }

public Task findTask(LinkedList<Task> tasks, String name){
for(Task t:tasks){
if (t.taskName.toLowerCase().startsWith(name.toLowerCase())) return t;
        }
logger.error("not found task named:"+name);
return null;
    }

public String getTaskName() {
return taskName;
    }

public void setTaskName(String taskName) {
this.taskName = taskName;
    }

public boolean isTaskNamed(String taskName) {
return (this.taskName.toLowerCase().startsWith(taskName.toLowerCase()));
    }

public String toString(){
return taskName+":"+super.toString();
    }
}
```

A.57 TaskResult.java

```
packageCA.Processes;

/**
 * Created by root on 30/05/2015.
 */
public class TaskResult {

public static final int TASK_STATUS_SUCCESS = 1;

public static final int TASK_STATUS_FAILURE = 2;
```

```
public static final int TASK_STATUS_CONDITIONAL_SUCCESS = 3;

    Task task;
int status;

public TaskResult(Task task){
this.task = task;
status = TASK_STATUS_SUCCESS;
    }

public TaskResult(Task task, int taskStatus){
this.task = task;
status = taskStatus;
    }

public Task getTask() {
return task;
    }

public void setTask(Task task) {
this.task = task;
    }

public int getStatus() {
return status;
    }

public void setStatus(int status) {
this.status = status;
    }

public String toString(){
        String s = super.toString();
        String res = "";
if (task!=null) res+=task.getTaskName();
if (status==TASK_STATUS_SUCCESS) res+=":Success";
if (status==TASK_STATUS_FAILURE) res+=":Failure";
return res;
    }
}
```

A.58 TaskResults.java

```
packageCA.Processes;

import java.util.LinkedList;

/**
 * Created by root on 14/06/2015.
 */
public class TaskResults {
    LinkedList<TaskResult>results = new LinkedList<>();

public TaskResults(){
results = new LinkedList<>();
    };

public TaskResults(LinkedList<TaskResult> copy){
results = copy;
```

```
    }

public boolean wasCallSuccess(){
if (results==null) return false;
if (results.size()==0) {
throw new NullPointerException();//not supposed to happen
//            return false;
}
        TaskResult taskResult = results.getLast();
if (taskResult==null) {
throw new NullPointerException();//not supposed to happen
//            return false;
}
return taskResult.getStatus()==TaskResult.TASK_STATUS_SUCCESS;
    }

public boolean wasCallSuccess(int offset){
return getLastStatus(offset)==TaskResult.TASK_STATUS_SUCCESS;
    }

public void add(TaskResult result){
results.add(result);
    }

public void add(Task t, int status){
results.add(new TaskResult(t,status));
    }

public void addToStack(Task t, int status){
for (int i=0;i<results.size();i++){
if (results.get(i).getTask().isTaskNamed(t.getTaskName())){
int del = results.size()-i;
for(int j=0;j<del;j++)
results.removeLast();;
            }
        }
        add(t,status);
    }

public Task getLastTask(){
return getLastTaskResult(0).getTask();
    }

public Integer getLastStatus(){
return getLastTaskResult(0).getStatus();
    }

public Task getLastTask(int offset){
return getLastTaskResult(offset).getTask();
    }

public Integer getLastStatus(int offset){
return getLastTaskResult(offset).getStatus();
    }

public int count(){
return results.size();
    }

public TaskResult getLastTaskResult(int offset){
```

```
if (results!=null&&results.size()>=(offset)) return results.get(results.size() -
offset-1);
return null;
    }

public boolean wasLastTaskNamed(String name, int offset){
return getLastTask(offset).isTaskNamed(name);
    }

public boolean wasLastTaskNamed(String name){
return getLastTask().isTaskNamed(name);
    }


public LinkedList<TaskResult> getList(){
return results;
    }
}
```

A.59 RMS

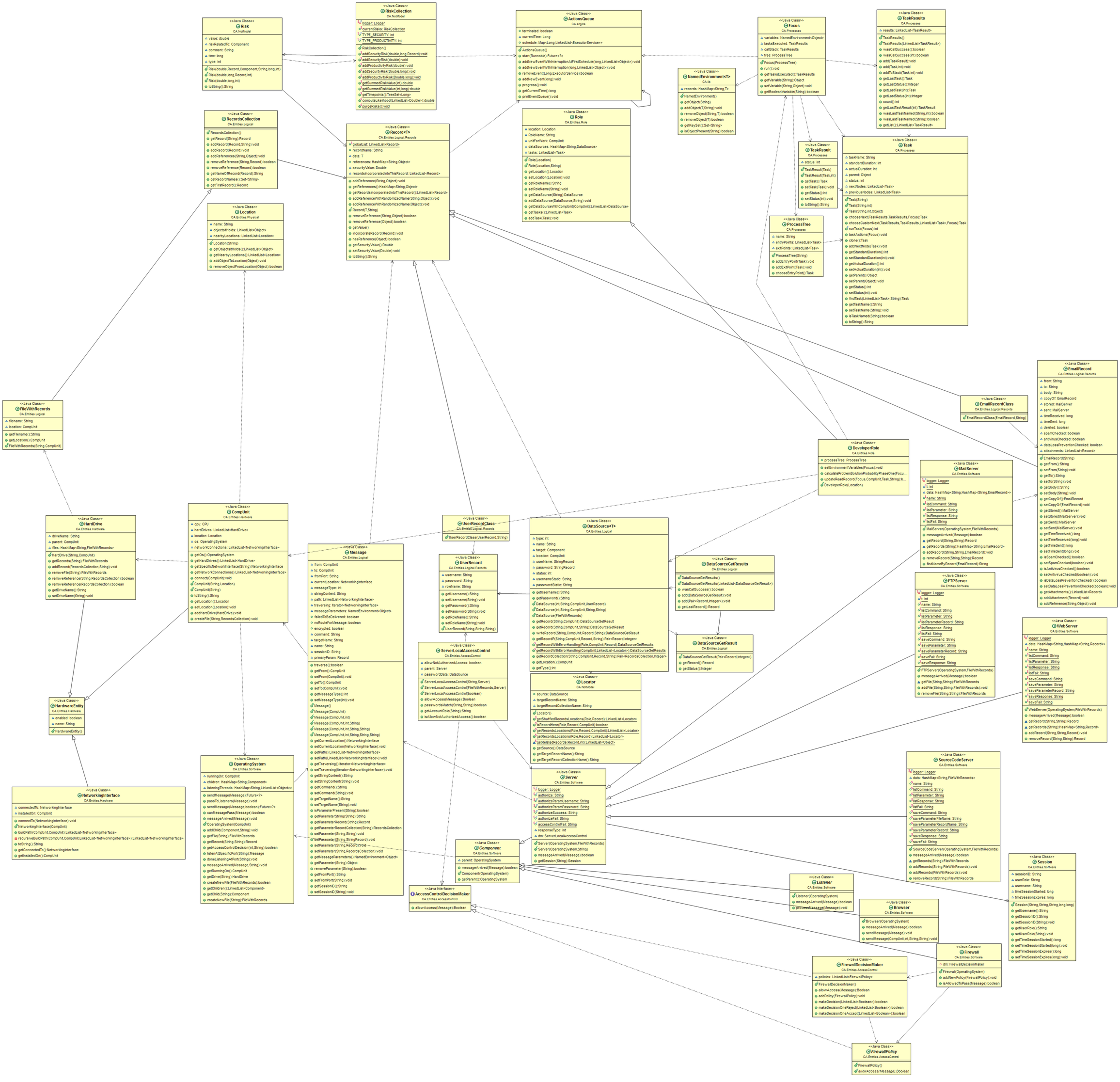
```
package RMS;

import CA.Entities.AccessControl.FirewallPolicy;
import CA.Entities.Logical.Message;
import CA.Entities.Logical.Records.Record;
import CA.NotModel.RiskCollection;
import CA.engine.ActionsQueue;
import CA.engine.Main;
import CA.engine.Random;
import CA.lib.FileNameCleaner;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.LinkedList;
import java.util.TreeSet;

/**
 *
 */
public class RMS {

private static final Logger logger = LogManager.getLogger(RMS.class);
static Thread watcherThread;

static String outputName = "test";
static final LinkedList<Record>policiesNotAllowedToPass = new LinkedList<>();
static final LinkedList<Record>mandatoryRecord = new LinkedList<>();

public static void main(String [] args){

        Main main = new Main();
        main.prepareEnvironment();
for(int i=0;i<Record.globalList.size();i++) {
for(int j=i+1;j<Record.globalList.size();j++) {
for(int k=j+1;k<Record.globalList.size();k++) {
```

```
for (int m = k + 1; m < Record.globalList.size(); m++) {

mandatoryRecord.clear();
mandatoryRecord.add(Main.docA);
mandatoryRecord.add(Main.headerA);
mandatoryRecord.add(Main.methodA);
mandatoryRecord.add(Main.docD);
mandatoryRecord.add(Main.headerD);
mandatoryRecord.add(Main.methodD);
                        Record r = Record.globalList.get(i);
                        Record r2 = Record.globalList.get(j);
                        Record r3 = Record.globalList.get(k);
                        Record r4 = Record.globalList.get(m);
if (mandatoryRecord.contains(r)) continue;
if (mandatoryRecord.contains(r2)) continue;
policiesNotAllowedToPass.clear();

                        Random.reInit();
outputName = "test" + FileNameCleaner.cleanFileName(r.toString() +
r2.toString()+r3.toString()+r4.toString());
logger.info("starting scenario for:" + outputName);
policiesNotAllowedToPass.add(r);
policiesNotAllowedToPass.add(r2);
policiesNotAllowedToPass.add(r3);
policiesNotAllowedToPass.add(r4);

watcherThread = createNewWatcherThread(1000 * 60 * 21, "For not allowing access
for Record:");

                        Main.logicalFirewall.addNewPolicy(new FirewallPolicy() {
@Override
public Boolean allowAccess(Message m) {
for (Record r : policiesNotAllowedToPass)
if (r.hasReference(m)) {
logger.warn("denying message passing because Record:" + r.toString() + " is
referenced in this message");
return false;
                                        }
return true;
                            }
                        });
                        main.startScenario();

try {
watcherThread.join();
                            Thread.sleep(2500);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }

                        ActionsQueue.currentQueue = new ActionsQueue();
                        main.prepareEnvironment();
                        RiskCollection.purgeRisks();
                    }
                }
            }
        }
        System.exit(0);
    }
```

```
public static void modellingIsCompleted(){
watcherThread.interrupt();
    }

public static Thread createNewWatcherThread(long time, String message){
        Thread t = new Thread(new Runnable() {
@Override
public void run() {
try {
                    Thread.sleep(time);
                } catch (InterruptedException e) {
//                      e.printStackTrace();
}


                ActionsQueue.currentQueue.terminated=true;

try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

logger.warn("----------------------------------------------------------------------
-------------------");
logger.warn(message+"Current security
risk:"+RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECURITY));
logger.warn(message+"Current productivity
risk:"+RiskCollection.getSummedRiskValue(RiskCollection.TYPE_PRODUCTIVITY));
logger.warn(message+"Time to complete task
sequence:"+ActionsQueue.currentQueue.getCurrentTime());
logger.warn(message+"Total Utility of the solution:"+(-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECURITY)*1000-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_PRODUCTIVITY)*400-
((ActionsQueue.currentQueue.getCurrentTime()/3600000d)*25)));

logger.warn("----------------------------------------------------------------------
-------------------");

try {
saveResults();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        t.start();
return t;
    }

public static void saveResults() throws IOException{
int utility = (int) (-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECURITY)*1000-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_PRODUCTIVITY)*400-
((ActionsQueue.currentQueue.getCurrentTime()/3600000d)*25));
        File f = new File(".\\output\\"+utility+outputName+".csv");
if (!f.exists())
            f.createNewFile();
        FileOutputStream fos = null;
try {
```

```
            fos = new FileOutputStream(f);

            TreeSet<Long> timeTree = RiskCollection.getTimepoints();
            LinkedList<Double> securityRiskValues = new LinkedList<>();
            LinkedList<Double> productivityRiskValues = new LinkedList<>();
            LinkedList<Double> utilityValues = new LinkedList<>();
long currentTime = ActionsQueue.currentQueue.getCurrentTime();
for(Long l:timeTree){

securityRiskValues.add(RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECU
RITY,l));

productivityRiskValues.add(RiskCollection.getSummedRiskValue(RiskCollection.TYPE_
PRODUCTIVITY,l));
                utilityValues.add((-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_SECURITY,l)*1000-
RiskCollection.getSummedRiskValue(RiskCollection.TYPE_PRODUCTIVITY,l)*400-
((l/3600000d)*25)));
                System.out.println(l + "is done of:"+currentTime);
            }
for(Long l:timeTree){
                fos.write((l.toString()+",").getBytes());

            }
            fos.write("\n".getBytes());
for(Double l:securityRiskValues){
                fos.write((l.toString()+",").getBytes());

            }
            fos.write("\n".getBytes());
for(Double l:productivityRiskValues){
                fos.write((l.toString()+",").getBytes());

            }
            fos.write("\n".getBytes());
for(Double l:utilityValues){
                fos.write((l.toString()+",").getBytes());

            }
            fos.write("\n".getBytes());

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
finally{
if (fos!=null)
                fos.close();
        }}}
```

Appendix B - UML Diagram of the Implementation Classes



Appendix C - Documentation Describing Class Attributes and Methods

1. AccessControlDecisionMaker – makes simple access control decisions, RBAC based.
    1. Attributes
        i. None
    2. Methods
        i. allowAccess – returns true if a Message passing or processing is allowed, false if Message should not be allowed traversal or processing, depending on the Component that is querying AccessControlDecisionMaker
2. ActionsQueue – stores all scheduled events, starts all events scheduled at the closest scheduled time. After starting the events, the ActionsQueue waits until all these events are either finished or paused.
    1. Attributes
        i. currentTime - integer, measures current time of simulation since scenario start, in milliseconds
        ii. schedule – queue of all scheduled events
        iii. Terminated - boolean, if set to true no more new events would be scheduled, purpose is to stop terminate simulation to get the risk data
    2. Methods
        i. ActionsQueue – constructor for this class
        ii. addNewEvent – adds a new event and returns control to invoking thread without delay
        iii. addNewEventWithInterruption – pauses execution of the current thread, schedules a new event to be executed and another event to resume the thread if event to be executed does not happen for some reasons (e.g., timeout)

iv. addNewEventWithInterruptionAtFirstSchedule – similar to addNewEventWithInterruption, this method is present for debugging purposes
v. getCurrentTime – gets current time, multi-thread safe
vi. printEventQueue – prints all scheduled events with their scheduled time
vii. progress – progresses simulation for one tick, specifically gets all events scheduled to be executed at the time closest to the current simulation time and starts the event processing
viii. removeEvent – removes scheduled, but not yet executed event
ix. Start – starts simulation, assumes that events queue contains at least one scheduled event

3. Browser – component used for accessing data in the Servers via HTTP/HTTPS protocols.
    1. Attributes
        i. None
    2. Methods
        i. Browser – constructor for this class
        ii. messageArrived – this routine is invoked when a Message has arrived and OperatingSystem forwards it to this Component
        iii. sendMessage – sends Message to intended recipient
4. Component – represents software executing in CompUnit managed by OperatingSystem.
    1. Attributes
        i. parent – specifies an OperatingSystem in which the Component is running
    2. Methods
        i. Component – constructor for this class
        ii. getParent – returns the parent OperatingSystem
        iii. messageArrived – via this routine the OperatingSystem sends Messages that have arrived for this Component
5. CompUnit – computational unit representing a piece of hardware that has hardware components connected to it and is capable of executing an OperatingSystem.
    1. Attributes
        i. cpu – specifies CPU's, intended for resource management

ii. hardDrives – specifies HardDrives connected to this CompUnit
iii. location – specifies a Location in physical world
iv. networkConnections – set of NetworkConnections that connect this CompUnit to other CompUnits via networking
v. os – represents an OperatingSystem that manages this CompUnit

2. Methods
    i. addHardDrive – adds a new HardDrive to this CompUnit
    ii. CompUnit – constructor for this class
    iii. connect – connects two NetworkConnections, one from this CompUnit and another from the specified CompUnit
    iv. createFile – creates a new FileWithRecords at the specified HardDrive
    v. getHardDrives – returns a list of HardDrives that are connected to this CompUnit
    vi. getLocation – returns a physical location of this CompUnit
    vii. getNetworkConnections – returns a list of all NetworkConnections associated with this CompUnit
    viii. getOS – returns the OperatingSystem that manages this CompUnit
    ix. getSpecificNetworkInterface – returns specific NetworkConnection by its name
    x. setLocation – changes a physical Location of this CompUnit
    xi. toString– returns the object representation as a string, for debugging and printing purposes

6. DataSource – specifies the protocol of retrieving a data (i.e., a Record) from a data source and lists all necessary information required to access it, such as its current network location, username and password.
    1. Attributes
        i. location – CompUnit, machine where this DataSource points to
        ii. name – name of this data source
        iii. password – password of the account required to access the Component which this DataSource refers to

iv. passwordStatic – password of the account required to access to the Component which DataSource refers to, this password would not be updated even if actual password will change
v. record – an instance of the UserRecord class which can be used instead of username and password if this attribute is set
vi. status – contains a status of the last access attempt, can be success, access denied, timeout, general failure or not yet started
vii. target – a Component that is set as the current location of the data
viii. type – type of the DataSource, can be a file, a database, an email server, a web server, or a version control server, this value defines which protocol would be used to access the data through this DataSource
ix. username – username of the account required access to the Component which this DataSource refers to
x. usernameStatic – username of the account required to access to the Component which this DataSource refers to, the username would not be updated even if an actual username will change

2. Methods
   i. DataSource – constructor for this class
   ii. getLocation – returns a CompUnit that will be queried for the data
   iii. getPassword - returns a usable password (if set), needed because different protocols use different username and password definition mechanisms
   iv. getRecord – starts routine to get the data (the Record), usually results in exchange of Messages between the originating Component and the Components referred in this DataSource. The only exception is if type is file and this file is located on the same machine as the Component that originated this call
   v. getRecordCollection – returns a RecordCollection by the name (i.e., entire file containing a set of Records)
   vi. getRecordP – internal routine implementing calls to get the data
   vii. getRecordWithErrorHandling – actual routine implementing a logic for different protocols (that is, the client side protocol is executed in the

routine, and the server side protocol logic is executed in the classes that override a Server class)

viii. getType – returns a type of the DataSource

ix. getUsername – returns a usable username (if set), needed because different protocols use different username and password definition mechanisms

x. writeRecord – starts routine to replace a current Record with another Record

7. DataSourceGetResult – results of a DataSource retrieval operation, contains an operation status and a list (possible empty if operation failed) of data entries associated with the retrieval.
    1. Attributes
        i. None
    2. Methods
        i. DataSourceGetResult – constructor for this class
        ii. getRecord – returns a Record by its name
        iii. getStatus – returns a status of the retrieval operation, can be: success, access denied, timeout or a general failure
8. DataSourceGetResults – set of a DataSourceGetResult for a set of calls retrieving data
    1. Attributes
        i. results – a set of DataSourceGetResult instances
    2. Methods
        i. add – adds DataSourceGetResult to the results set
        ii. DataSourceGetResults – constructor for this class
        iii. getLastRecord – returns a last record of the set of calls
        iv. wasCallSuccess – returns true if a call originated sequence of Record retrieval operations resulted in success and all records were retrieved; if at least one record was not retrieved this method returns failure
9. DeveloperRole – describes a role of a software developer that is associated with a Process.

1. Attributes
    i. processTree – ProcessTree of this role
2. Methods
    i. calculateProblemSolutionProbabilityPhaseOne – calculates whether a developer actually solved the problem of the first part of a task
    ii. DeveloperRole – constructor for this class
    iii. setEnvironmentVariables – sets variables of the process representing the ProcessTree execution for this role
    iv. updateReadRecord – updates attributes associated with a Record developer may need to read to simulate that the Record was read

10. EmailRecord – a Record containing information about an email stored in a Server or passed from one server to another server

    1. Attributes
        i. antivirusChecked – indicates whether this Email was checked by an antivirus
        ii. attachments – a list of the Files attached to this EmailRecord
        iii. body – body of the Email
        iv. copyOf – indicates whether this is a copy of the Email or an original Email
        v. dataLossPreventionChecked – indicates whether this Email was checked by a data loss prevention software
        vi. deleted – indicates whether this email was deleted
        vii. from – specifies an address from which this Email came from
        viii. sent – EmailServer that was used to send this Email
        ix. spamChecked – indicates whether this Email was checked by a spam control
        x. stored – a location where this Email is stored
        xi. timeReceived – specifies the time when this Email was received
        xii. timeSent – specifies the time when this Email was sent
        xiii. to – specifies the address of this Email destination
    2. Methods

i. addAttachment – adds an Attachment to the Email
ii. addReference – adds a new Reference to this Email, and changes a DataLossPreventionChecked value to false since it modifies the content of this Email
iii. EmailRecord – constructor for this class
iv. getAttachments – returns a set of Attachments (if any)
v. getBody – returns the Email body value
vi. getCopyOf – returns the Email copyOf value
vii. getFrom – returns the Email from value
viii. getSent – returns a sent value
ix. getStored – returns a stored value
x. getTimeReceived – returns a TimeReceived value
xi. getTimeSent – returns a TimeSent value
xii. getTo – returns the value of an attribute named to
xiii. isAntivirusChecked – returns an AntivirusChecked value
xiv. isDataLossPreventionChecked – returns true if this email was scanned by a leak detection system
xv. isSpamChecked – returns a SpamChecked value
xvi. setAntivirusChecked – updates an AntivirusChecked value
xvii. setBody – updates the value of an attribute body
xviii. setDataLossPreventionChecked – updates the related attribute
xix. setFrom – updates the value of the from atribute
xx. setSent – updates the value of the sent atribute
xxi. setStored – updates the value of the stored atribute
xxii. setTimeReceived – updates a TimeReceived value
xxiii. setTimeSent – updates a TimeSent value
xxiv. setTo – updates the value of the to atribute

11. FileWithRecords – represents a File located in a HardDrive that contains a list (possible empty) of Records.
    1. Attributes
        i. Filename– a file path relative to a drive name

ii. location - a CompUnit specifying the file physical location

2. Methods
    i. FileWithRecords – constructor for this class
    ii. getFileName – returns the file name
    iii. getLocation – returns the CompUnit where the File is located

12. Firewall – represents a hardware Firewall, using a FirewallDecisionMaker that makes access control decisions about the Messages passing through this Firewall.
    1. Attributes
        i. dm – a decision maker, applying a set of Firewall policies to the Messages passing through this Firewall
    2. Methods
        i. addNewPolicy – adds a new policy to a set of Firewall policies
        ii. Firewall – constructor for this class
        iii. isAllowedToPass – returns true if the Firewall lets a Message pass
13. FirewallDecisionMaker - makes access control decision whether to allow a Message to pass or not based on a set of FirewallPolicies.
    1. Attributes
        i. policies – a set of policies defining which Messages are allowed to pass and which are prohibited to do so
    2. Methods
        i. addPolicy – adds a policy to a Firewall policy set
        ii. allowAccess – returns true if a Message is allowed to pass
        iii. FirewallDecisionMaker – constructor for this class
        iv. makeDecision – makes a decision about whether a Message is allowed or prohibited to proceed
        v. makeDecisionOneAccept – returns that a Message is allowed to proceed if one Policy is allowing
        vi. makeDecisionOneReject – returns a Message is allowed to proceed if all Policies allow it
14. FirewallPolicy – contains an ABAC policy specifying all types of Messages that fall under certain category and what a Firewall should do with them

1. Attributes
    i. None
2. Methods
    i. allowAccess – returns true if a Message is allowed to proceed according to this policy
    ii. FirewallPolicy – constructor for this class

15. FTPServer – represents a Server that is used to access local files remotely.

1. Attributes
    i. listCommand – a value identifying a command to list files that contain a specified Record
    ii. listFail – a value specifying that a list command has failed
    iii. listParameter – a value identifying a name of a specific directory to list
    iv. listParameterRecord – a value identifying a specific Record which, if specified, is used instead of the listParameter attribute
    v. listResponse – a value specifying a Message type as a response to a list request
    vi. logger – contains logs of all relevant activities
    vii. name – a name of the FTPServer
    viii. saveCommand – a value identifying a command to create a new File or append a Record to an existing File
    ix. saveFail – a value specifying that this Message is a reply to a failed save request
    x. saveParameter – a value specifying a File name
    xi. saveParameterRecord – a value specifying a Record to append to a specified File
    xii. saveResponse – a value specifying that this Message is a reply to a successful save request
    xiii. t – a time difference stored for debugging purposes
2. Methods
    i. addFile – adds a new File to a HardDrive

ii. FTPServer – constructor for this class
iii. getFile – returns the content of a File
iv. messageArrived – this method is invoked when a Message has arrived and it needs to be processed, overridden method of the Server class
v. removeFile – deletes a File from a HardDrive

16. HardDrive – represents a storage device in a CompUnit which stores FileWithRecords; this class is an instantiation of the HardwareEntity class
    1. Attributes
        i. driveName – represents a HardDrive name, a prefix for all Files located on this Hard Drive
        ii. files – a list of FileWithRecords and their names
        iii. parent – a CompUnit housing this HardDrive
    2. Methods
        i. addRecord – adds a new Record to a specific FileWithRecord
        ii. getDriveName – returns the name attribute of this HardDrive
        iii. getRecords – returns all Records by their names
        iv. HardDrive – constructor for this class
        v. removeFile – removes a FileWithRecord from HardDrive
        vi. removeReference – removes a reference of the Record from a FileWithRecord
        vii. setDriveName – changes the name attribute of this HardDrive
17. HardwareEntity – represents a piece of hardware that can perform certain function.
    1. Attributes
        i. enabled – if this value is false, then a functionality of the HardwareEntity is disabled
        ii. name – a name of the HardwareEntity, used for identification purposes
    2. Methods
        i. HardwareEntity – constructor for this class
18. Listener – an abstract Component that has a Message listening capability which is not able to consume or change a Message.

1. Attributes
    i. None
2. Methods
    i. Listener – constructor for this class
    ii. messageArrived – processes a Message that was forwarded to this Component by its parent OperatingSystem
    iii. processMessage – consumes an arrived Message

19. Location – a physical world location, used in risk assessment.
    1. Attributes
        i. Name – a Location name, serves for identification purposes
        ii. nearbyLocations – a list of Locations that are somehow physically connected to this Location, for example neighbouring rooms
        iii. objectsItHolds – CompUnits that are located in this Location and consequently can be used by a User searching for a computer in which to work
    2. Methods
        i. addObjectToLocation – adds a new CompUnit to this Location
        ii. getNearbyLocations – returns other Locations a User can move to
        iii. getObjectsHolds – returns the list of CompUnits that are located in this Location
        iv. Location – constructor for this class
        v. removeObjectFromLocation – removes a CompUnit from this Location
20. Locator – metaclass, that have access to all Records Locations, represents a User's memory about Records Locations (where did a User store the data) and DataSources (how to access it, if a User has access to that Location).
    1. Attributes
        i. source – a DataSource created once a Record is located
        ii. targetRecordCollectionName – name of the RecordsCollection that this Locator is searching for
        iii. targetRecordName – a Record name that this Locator is searching for
    2. Methods

i. getTargetRecordCollectionName – returns a name of the RecordCollection the Locator is searching for
ii. getRecordsLocations – returns a set of Locators, this set contains all possible ways of getting the specified Record both with the DataSources and anonymous
iii. getRelatedRecords – returns a set of references for the Record Location
iv. getShuffledRecordsLocations – reshuffles randomly the output of the getRecordsLocations
v. getSource – returns a DataSource created for a data access
vi. getTargetRecordName – returns a name of a Record the Locator is searching for
vii. isRecordHere – returns true if target Record is located at the specified CompUnit
viii. Locator – constructor for this class

21. MailServer – represents a Server that stores and exchanges Emails.
    1. Attributes
        i. data – a set of EmailRecords for each user account on this MailServer
        ii. listCommand – a value identifying a command to list emails associated with a specific user account
        iii. listFail – a value identifying that a list command has failed
        iv. listParameter – a value identifying a parameter of a specific Email to retrieve
        v. listResponse – a value identifying a Message as a reply to a list command
        vi. logger – contains logs of all relevant activity
        vii. name – name of the server
        viii. t – time difference for debugging purposes
    2. Methods
        i. addRecord – adds a new Record
        ii. findNameByRecord – returns a Record by part of its name
        iii. getRecord – returns a Record by its name
        iv. getRecords – returns a set of Records by part of their names

v. MailServer – constructor for this class
vi. messageArrived – this method is invoked when a Message has arrived and needs to be processed; this is an overridden method of the Server class
vii. removeRecord – removes a Record from a set of Records at this Server

22. Message – relates to an InfrastructureEvent and represents a Message of any protocol Components exchange.
    1. Attributes
        i. command – a command to a Server, must match one of Server values otherwise a Message would be ignored (can be empty for Server responses)
        ii. currentLocation – current NetworkingInterface where this Message is located
        iii. encrypted – boolean flag set to true if Message parameters and contents are encrypted and can be read only by target Component
        iv. failedToBeDelivered – boolean flag set to true if a Message could not be delivered
        v. from – a CompUnit that has originated this Message
        vi. fromPort – a port from which this Message was sent, a response to this Message should be sent to the same port
        vii. messageParameters – a set of parameters that was defined in its originating CompUnit
        viii. messageType – a value defining the type of the protocol used to send this Message
        ix. name – a name of the Message used for identification purposes
        x. noRouteForMessage – boolean flag set to true if the originating CompUnit could not construct a route to the intended destination of this Message
        xi. path – path of the Message, constructed at the originating CompUnit
        xii. primaryParam – one of the Message Parameters
        xiii. sessionID – id of a Session set by a Server if an authentication was successful and the Session was created by the Server. This value is passed

from a Server to communication originating a Component back and forth to identify a state of the communication Session and its context

xiv. stringContent – contents of the Message in text, if any

xv. targetName – name of the Component this Message is intended to be delivered to

xvi. to – a CompUnit that this Message was sent to

xvii. traversing – an iterator over the path of the Message

2. Methods

i. getCommand – returns a command for a Server in the Message

ii. getCurrentLocation – returns a NetworkInterface where the Message is currently located

iii. getFrom – returns a CompUnit this Message has originated from

iv. getFromPort – returns a port from which the Message was sent

v. getMessageParameters – returns a map of Parameter names and their values

vi. getMessageType – returns a value defining a communication protocol used to send this Message

vii. getParameter – returns (if present) a Parameter by its name

viii. getParameterRecord – returns (if present) a Parameter as a Record

ix. getParameterString – returns (if present) a Parameter in textual representation

x. getPath – returns a path of the Message

xi. getSessionID – returns a Session id (if set) of the Message

xii. getStringContent – returns a text passed along with the Message

xiii. getTargetName – returns a name of a Component this Message is intended for

xiv. getTo – returns a CompUnit this Message is sent to

xv. getTraversing – returns an iterator traversing over the path of the Message

xvi. isParameterPresent – returns true if a Parameter with a specified name is present

xvii. Message – constructor for this class

xviii. removeParameter – removes a Parameter by its name
xix. setCommand – changes a command for a Server in the Message
xx. setFrom – changes a CompUnit this Message has originated from
xxi. setFromPort – changes a port from which Message was sent
xxii. setMessageType – changes a value defining a communication protocol using which this Message was sent
xxiii. setParameter – sets the Parameter of the Message to a specific value
xxiv. setPath – changes a path that was constructed for the Message to achieve its target
xxv. setSessionId – changes an id of the Session
xxvi. setStringContent – changes the text passed along with the Message
xxvii. setTo – changes a CompUnit this Message is sent to
xxviii. setTraversing – changes an iterator traversing over the Message path
xxix. traverse – attempts to send a Message to a next Location; it is invoked when an infrastructure event is executed

23. NamedEnvironment – this class that contains a name of Parameters and their values, used in a Task for storing environment variables and in a Message for storing the Message Parameters
    1. Attributes
        i. records – a set of objects associated with this NamedEnvironment; each object has a name
    2. Methods
        i. addObject – adds a new object to this NamedEnvironment
        ii. getKeySet – returns names of all objects that this NamedEnvironment contains
        iii. getObject – finds an object by its name in this NamedEnvironment
        iv. isObjectPresent – checks if any object from specified list of objects is present in this NamedEnvironment, returns true if present
        v. NamedEnvironment – constructor for this class
        vi. removeObject – removes an object by its name from this NamedEnvironment

24. NetworkingInterface – represents a network link between two CompUnits, and is an instantiation of HardwareEntity class.
    1. Attributes
        i. connectedTo – another NetworkingInterface this NetworkingInterface is connected to
        ii. installedOn – a CompUnit where this NetworkingInterface is installed
    2. Methods
        i. buildPath – creates a path for a Message, calls recursiveBuildPath
        ii. connectTo – connects this NetworkingInterface to another NetworkingInterface
        iii. getConnectedTo – returns a Networking Interface that this NetworkingInterface is connected to
        iv. getInstalledOn – returns a CompUnit where this NetworkingInterface is installed
        v. NetworkingInterface – constructor for this class
        vi. recursiveBuildPath – recursively builds a path for a Message
        vii. toString– casts this object to a string for debugging and printing purposes
25. OperatingSystem – simulates the functionality of an operating system, including Message creation, routing, filtering, consumption, works with Files and includes a basic access control
    1. Attributes
        i. children – returns all Components that are running with a specified Component
        ii. listeningThreads – all threads (executing Task Events) that have subscribed to listening Messages coming to this OperatingSystem
        iii. runningOn – returns a CompUnit that is managed by the OperatingSystem
    2. Methods
        i. addChild – adds a new child Component
        ii. canMessagePass – checks whether a Message can continue its traversal; it is used to execute Firewall function
        iii. createNewFile – creates a new FileWithRecords at a specified HardDrive

iv. doneListeningAtPort – marks a Component as no longer interested in listening to a specific port activity
v. getAccessControlDecision – returns an AccessControlDecision, can be overridden to implement a Firewall not at an application level as a Component, but at a transport level
vi. getChild – returns a specific Component by its name
vii. getChildren – returns all Components managed by this Operating System
viii. getDrive – returns a HardDrive by its name
ix. getFile – returns a FileWithRecords by its name
x. getRecord – returns a Record from a FileWithRecords by Record's name and name of a File
xi. getRunningOn – returns a CompUnit managed by this Operating System; this is used for identification and for hardware search
xii. listenAtSpecificPort – listens at a specific port
xiii. messageArrived – routine to process a Message once it has arrived at an intended CompUnit
xiv. OperatingSystem – constructor for this class
xv. passToListeners – sends a received Message to all listeners that have subscribed to a specific port where the Message has arrived
xvi. sendMessage – sends a Message to a specified target

26. ProcessTree – a graph of Tasks, specifying its entry points, exit points, and possible Task connections
    1. Attributes
        i. entryPoints – lists possible entry points for a Process to start
        ii. exitPoints – lists the exit points. Once a Focus traversing over this ProcessTree reaches this node, the Process is considered to be completed and no future Tasks are scheduled
        iii. name – name of the ProcessTree used for identification purposes
    2. Methods
        i. addEntryPoint – adds a new entry point to this ProcessTree, one of them would be chosen when a Focus starts traversing over this ProcessTree

ii. addExitPoint – adds a new exit point to this ProcessTree, once a Focus traversing over this ProcessTree is at this node the Process is considered to be completed and no future Tasks would be scheduled
iii. chooseEntryPoint – chooses an entry point for the Process to start, and is called by a Focus when it is activated
iv. ProcessTree – constructor for this class

27. Record – Risk-related piece of data stored and processed in the simulated system
    1. Attributes
        i. data – an abstract piece of data associated with this Record, typically is overridden in child classes to a specific class
        ii. globalList – a list of all currently existing Records in the simulation, used for lookups
        iii. recordName – a Record's name, serves for records identification
        iv. recordsIncorporatedIntoThisRecord – a list of Records that are contained within this Record
        v. references – a list of all objects that reference this Record. These objects can either reference other Records or Messages. Each reference contains a link to a referenced Record and a name by which this reference can be found. For example, an attachment in an instance of the EmailRecord class is added with a list of names of attached Records. An EmailRecord can have multiple attachments and a Component searching for a specific Record attached to an EmailRecord can locate it by its intended name
        vi. securityValue – a value of how sensitive this Record is, based on a security risks, which represents the maximum amount of damage the software development company will incur if this Record is intercepted by a malicious entity
    2. Methods
        i. addReference – adds a new reference to this object
        ii. addReferenceWithRandomizedName – adds a new reference to this object; the name of the reference is randomized so that they will not be found if searched using a specific name

iii. getRecordIncorporatedIntoThisRecord – returns a list of other Records incorporated into this Record
iv. getReferences – returns a list of references to this Record
v. getSecurityValue – returns a security value associated with this Record
vi. getValue – returns an object associated with this Record
vii. hasReference – checks whether an object was referenced by this Record, and if it was, the method returns true, and false otherwise
viii. incorporateRecord – incorporates another Record in this Record. This includes setting a reference and iteration over the incorporated Records. These incorporated records can be, for example, Records that are incorporated in a Record that is intercepted by a malicious entity
ix. Record – constructor for this class
x. removeReference – removes a reference from this Record
xi. setSecurityValue – changes a security value associated with this Record
xii. toString – casts this object to a string, for debugging and printing purposes

28. RecordsCollection – A collection of Records
    1. Attributes
        i. Records – a list of all Records contained in this RecordsCollection
    2. Methods
        i. addRecord – adds a new record to the RecordsCollection either by specifying a Record name or by using a randomized name if none is specified
        ii. addReference – adds an object as a reference to all Records of this RecordsCollection; for example, when a Message is created it references all Records it contains
        iii. getFirstRecord – returns a first Record of this RecordsCollection
        iv. getNameOfRecord – returns a name of specific Record within this Collection
        v. getRecord – returns a specific Record by its name
        vi. getRecordNames – returns a set of all Records names contained in this RecordsCollection

vii. RecordsCollection – class constructor
viii. removeReference – removes an object from a list of references of this RecordsCollection; for example, when a Message is consumed, all references to it are removed

29. Risk – accumulated risk level of a specific risk dimension for each Record
    1. Attributes
        i. comment – comments on Risk used for identification purposes
        ii. Object – a Record associated with the Risk
        iii. riskRelatedTo – a Component related to this Risk, typically a Component that originated an Event that resulted in this Risk
        iv. time – time of the Risk occurrence
        v. type – type of the Risk; the software development scenario includes two types, security and productivity
        vi. value – a probability of realizing that Risk; it is used as part of probabilistic model to assess utility of this Risk
    2. Methods
        i. Risk – constructor for this class
        ii. toString – casts this object to a string, for debugging and printing purposes
30. RiskCollection – collection of Risks for all dimensions and all Records.
    1. Attributes
        i. currentRisks – a list of all Risks that might have been realized at this point of time during a simulation. The fact that a Risk is present in this list does not mean that the Event associated with the Risk has occurred, it means that there is greater than 0 probability that an Event might have had occurred. This class is used for utility calculation
        ii. Logger – contains a log of all events associated with added Risks
        iii. Risks – a list of Risks in this RiskCollection
        iv. TYPE_PRODUCTIVITY – a number identifying a productivity Risk
        v. TYPE_SECURITY – a number identifying a security Risk
    2. Methods
        i. addProductivityRisk – adds a new productivity Risk

ii. addSecurityRisk – adds a new security Risk
iii. computeLikelihood – computes a likelihood of a set of Risks associated with one Record. For example, if there were two cases when a Record was intercepted by a malicious listener, each with a probability of 0.5, a resulting likelihood of this would be not 0.5+0.5=1, but 0.5+0.5*(1-0.5)=0.75
iv. getSummedRiskValue – returns a total utility of a specific risk type
v. getTimepoints – returns all points on timeline when Risks were incurred
vi. purgeRisks – removes all risks; this method is used to restart a forecasting model simulation.
vii. RiskCollection – constructor for this class

31. Role – specifies a ProcessTree that is associated with a Component that fullfills that role
    1. Attributes
        i. dataSources – a list of Locations, usernames and passwords of Components a User can access; it is used to represent memories of the User
        ii. location – a current Location of a User with this Role
        iii. RoleName – a name of the Role, serves for identification purposes
        iv. tasks – a list of Tasks a Component (User) with this Role can execute
        v. unitForWork – a CompUnit selected for work of a User with this Role
    2. Methods
        i. addDataSource – adds a new DataSource
        ii. addTask – adds a new task that can be executed by a User
        iii. getDataSource – returns a DataSource by its name
        iv. getDataSourceWithCompUnit – returns a list of all DataSources associated with a specific CompUnit, useful when a User tries to get access to a certain resource and there are multiple Components that can be used to access the resource, e.g., FTP server and SourceCodeServer
        v. getLocation – returns a current Location of a User
        vi. getRoleName – returns a name attribute of the Role
        vii. getTasks – returns a list of tasks currently executed by a User

viii. Role – constructor for this class
ix. setLocation – changes a Location of a User, e.g., moving from one room to another
x. setRoleName – changes the name attribute of the Role

32. Server – an abstract class of Components that has a Message consumption capabilities with a number of default routines, such as a basic access control and is capable of creating Sessions .
    1. Attributes
        i. accessControlFail – a value identifying a Message as being a reply to a request that violates a current ServerLocalAccessControl scheme
        ii. authorize – a value identifying a Message as being an authorization request
        iii. authorizeFail – a value identifying a Message as being a reply to a failed authentication attempt
        iv. authorizeParamPassword – a value identifying a Message Parameter as containing a password for an authorization
        v. authorizeParamUsername – a value identifying a Message Parameter as containing a username for an authorization
        vi. authorizeSuccess – a value identifying that a request for authorization was successful
        vii. dm – (decision maker) a ServerLocalAccessControl that specifies which Messages can be processed at this Server
        viii. logger – a logger, used to record this class activity
        ix. responseType – a type of a protocol that this Server uses for communication
        x. sessions – a list of all current Sessions opened currently at this Server
    2. Methods
        i. getSession – intended to be overridden in subclasses; it returns the Session by its name
        ii. messageArrived –processes arriving Messages
        iii. Server – constructor for this class

33. ServerLocalAccessControl – an instantiation of the AccessControlDecisionMaker class that stores a set of UserRecords in a DataSource, and can be, for example a FileWithRecords.
    1. Attributes
        i. allowNotAuthorizedAccess – allows access without authorization, if this attribute is true then any authorization request will be granted and server will not require authentication for the actions
        ii. parent – a Server that this ServerLocalAccessControl is providing access control for
        iii. passwordData – a DataSource that contains a list of registered usernames, passwords and roles
    2. Methods
        i. allowAccess – the Server will invoke this method when a new Message arrives. This method returns an access control decision regarding the Message based on the type of the Server, the attributes of the Message and the set of access control policies
        ii. getAccountRole – returns a role of an account associated with a Server by its name
        iii. isAllowNotAuthorizedAccess – returns the allowNotAuthorizedAccess attribute
        iv. passwordsMatch – returns true if a specified username and password matches an existing Record from passwordData, and false otherwise
        v. ServerLocalAccessControl – constructor for this class

34. Session – stores all information necessary for a Server to process requests within a communication session between a specific Component and the Server.
    1. Attributes
        i. sessionID – an id of the Session created by the Server when the Server acknowledges an authentication request from a Component and either there was no previous communication between this Component and the Server or the previous communication Sessions are expired

ii. timeSessionExpires – the time when the Session will expire
iii. timeSessionStarted – the time when the Session was created
iv. username – username of an account for which the Session was created
v. userRole – role of an account for which the Session was created

2. Methods
   i. getSessionID – returns id of the session
   ii. getTimeSessionExpires – returns time when a Session will expire
   iii. getTimeSessionStarted – returns time when a Session was created
   iv. getUsername – returns a username of an account for which the Session was created
   v. getUserRole – returns a role of an account for which a Session was created
   vi. Session – constructor for this class
   vii. setSessioID – changes id of the Session
   viii. setTimeSessionExpires – changes time when the Session will expire
   ix. setTimeSessionStarted – changes time when the Session was created
   x. setUserRole – changes a role of an account for which the Session was created

35. SourceCodeServer – represents a Server that stores source code used in the software development case study.
    1. Attributes
       i. data – a set of Records associated with each User accounts
       ii. listCommand – a value identifying a command to list source code files associated with a specific User account
       iii. listFail – a value identifying that a list command has failed
       iv. listParameter – a value identifying a Parameter of specific source code files to retrieve
       v. listResponse – a value identifying that a Message is a response to a list command
       vi. logger – contains the logs of all relevant activities
       vii. name – name of the Server

viii. saveCommand – a value identifying a command to create a new Record associated with a specific User account
ix. saveFail – a value identifying that a save command has failed
x. saveParameterFileName – a value identifying a name attribute of a File to be created on the Server
xi. saveParameterRecord – a value identifying a value of a specific Record to be created on the Server
xii. saveParameterRecordName – a value identifying a name attribute of a specific Record to be created on the Server
xiii. saveResponse – a value identifying that a Message is a response to a save command

2. Methods
   i. addRecords – adds a Record to a source code File
   ii. getRecords – retrieves a Record from specified source code File
   iii. messageArrived – method invoked when a Message has arrived and needs to be processed; this method overrides the same method in the Server class
   iv. removeRecord – removes a Record from a source code File
   v. SourceCodeServer – constructor for this class

36. Task – contains the logic routines responsible for the execution of the actions, the conditions this task execution waits for, and the switching routines responsible for selecting a next Task in a ProcessTree.
    1. Attributes
       i. actualDuration – a modified duration of a standardDuration attribute
       ii. nextNodes – a set of all possible next Tasks to be executed after this Task is completed
       iii. parent – an object responsible for the Task execution, which can be either a Role or a Component
       iv. previousNodes – a set of Tasks that could have lead to the execution of the Task
       v. standardDuration – an expected duration of the Task. The duration can change, depending on the execution results of previous Tasks and Process

attributes. It is used to set the time when the Event associated with the Task completion would be processed

vi. status – a status of the Task execution, can be a success, a failure or not defined

vii. taskName – a name of the Task used for identification purposes

2. Methods

i. addNextNode – adds the Task in the list of Tasks that can be executed after this Task is completed, choosing next Task is done in a method called chooseNext, which in turns depends on a method called chooseCustomNext

ii. chooseCustomNext – a method responsible for choosing a Task that will be executed next. This method is invoked after the Task has been completed and has access to a list of Process attributes and history of previous Tasks executions and queue of executions that lead to the Task

iii. chooseNext – default implementation of a routine that chooses a next Task. If chooseCustomNext is not implemented, this routine will be invoked, in an implementation exists only for Tasks that has one next Task

iv. clone – copying constructor. All Tasks defined in scenario are used as templates, instead of executing them, they are being copied first and then executed

v. findTask – finds a Task by its name searching through NextNodes Tasks. It returns a list of Tasks that can be executed in the ProcessTree after this Task is completed

vi. getParent – return a parent object of the Task, a Role (that represents a User) or a Component

vii. getStandardDuration – returns an expected duration of the Task

viii. getStatus – returns an execution status of the Task

ix. getTaskName – returns a name of the Task, used for identification

x. isTaskNamed – returns true if the Task was named

xi. runTask – method wrapper for a method called taskActions

xii. setParent – sets a parent object for the Task

xiii. setTaskName – changes a name attribute of the Task
xiv. Task – constructor for this class
xv. taskActions – a method intended to be overridden by classes defining actual Tasks, contains actions that are executed during Task execution. This method starts when the ActionsQueue starts the Task-related Event. This method should finish before the ActionsQueue changes simulation time and starts a new set of Events. However if the Task execution was paused, for example because of a call of the method addNewEventWithInterruption, then this Task has to be activated by a timeout or by a scheduled event
xvi. toString– casts the object to a string, for debugging and printing purposes

37. TaskResult – results of a Task execution, used in a Task switching routine
    1. Attributes
        i. status – a status of the Task execution and it can either be a success or a failure
        ii. task – the Task for which TaskResult is constructed
    2. Methods
        i. getStatus – returns an execution status of the Task and can either be a failure, a success, or undefined (if status was not set yet, but it is not used)
        ii. getTask – returns the Task associated with the TaskResult
        iii. setStatus – changes a status of the Task execution. This happens after the Task execution has finished and can either be a success or a failure. Both statuses are set within the TaskActions method
        iv. setTask – changes the Task associated with the TaskResult
        v. TaskResult – constructor for this class
        vi. toString– casts the object to a string, for debugging and printing purposes
38. TaskResults – a sequence of TaskResults
    1. Attributes
        i. results – a set of TaskResults. The order of the elements in this set is the same in which they were added
    2. Methods

i. add – adds a new TaskResult to a sequence
ii. addToStack – used in adding a Task to an execution stack. It does not support recursive Task executions
iii. count – returns a count of the executed Tasks
iv. getLastStatus – returns a status of a last Task that completed its execution (if any)
v. getLastTask – returns a last Task that completed its execution
vi. getLastTaskResult – returns a TaskResult of the last Task execution
vii. getList – returns a copy of an internal list of TaskResults
viii. TaskResults – constructor for this class
ix. wasCallSuccess – returns true if the last executed Task was a success, false otherwise
x. wasLastTaskNamed – returns true if the last executed Task was named

39. Focus – a pointer to a ProcessTree of a User (or an active Component)
    1. Attributes
        i. callStack – a stack of executed Tasks, only contains non-repeating Tasks. If while adding a new Task a repetition occurs then all Tasks executed between these two calls are removed from this stack. This attribute is used to identify which Task has originated the current Task to be executed.
        ii. tasksExecuted – a list of all executed Tasks for the Focus
        iii. tree – ProcessTree of the Process the User (or a Component) is following
        iv. Variables – contains a list of attributes set by the User related to this Focus
    2. Methods
        i. Focus – constructor for this class
        ii. getBooleanValue – returns a Boolean attribute of the User
        iii. getTasksExecuted – returns a list of Tasks that were executed by this Focus previously
        iv. getVariable – returns an attribute associated with the User executing the Tasks of the ProcessTree
        v. Run – the Focus class executes the logic of Tasks via the Thread class. This method is necessary for the Thread class execution. The Task event

call executes this method and internally it executes a runTask and a chooseNext methods sequentially

vi. setVariable – changes an attribute associated with the User

40. UserRecord – a Record containing information about a User, including a role, username and password, used to provide access to an instance of a Server class.
    1. Attributes
        i. password – a password of the UserRecord
        ii. roleName – a role name of the UserRecord
        iii. username – a username of the UserRecord
    2. Methods
        i. getPassword – returns the password of the UserRecord
        ii. getRoleName – returns the role name of the UserRecord
        iii. getUserName – returns the username of the UserRecord
        iv. setPassword – changes the password of the UserRecord
        v. setRoleName – changes the role of the UserRecord
        vi. setUserName – changes the username of the UserRecord
        vii. UserRecord – constructor for this class
41. WebServer – represents a Server that allows a User to access its services via HTTP or HTTPS communication protocols, for example through a browser
    1. Attributes
        i. data – a set of Records associated with each User account
        ii. listCommand – a value identifying a command to list EmailRecords associated with a User account
        iii. listFail – a value identifying that a list command has failed
        iv. listParameter – a value identifying a Parameter of a specific EmailRecord to retrieve
        v. logger – contains logs of all relevant activities
        vi. name – name of the WebServer
        vii. saveCommand – a value identifying a command to create a new Record associated with the User account
        viii. saveFail – a value identifying that a save command has failed

ix. saveParameter – a value identifying a name of a specific Record to create on the WebServer
x. saveParameterRecord – a value identifying a specific Record to create on the WebServer
xi. saveResponse – a value identifying that a save command was successful

2. Methods
   i. addRecord – adds a new Record
   ii. getRecord – returns a Record by its name
   iii. getRecords – returns a set of Records by their names
   iv. messageArrived – method invoked when a Message has arrived and needs to be processed, overridden method of the Server class
   v. removeRecord – removes a Record from a set of Records in this WebServer
   vi. WebServer – constructor for this class

Appendix D - Basic Design Model and Risk Assessment

There is a lack of RAdAC implementations as mentioned in Chapter 2 as well as a significant number of different types of risk analysis. Each risk analysis type requires a specific set of data as an input. This appendix provides a sample of risk analysis done according to RAdAC and explains details of the risk analysis calculation. Moreover, a data model provided in this appendix can be used for any type of risk analysis as well as for a combination of several risk analysis types. This data model was used as a basic risk analysis model in the proposed approach. This appendix also includes an illustrative example of a risk analysis performed for a software development scenario. The results of the risk analysis illustrate the outcome of using the proposed approach for a software development project. Figure 36 represents a plan of the software development project for this example. The plan contains four tasks (A, B, C, D) that are required to complete the project, one optional task (E), and a maintenance task (F). Tasks B and D are part of the critical path for the project completion.

Software Development Scenario Initial Data

Figure 36. Project development diagram

An example is based on the initial data provided in Table 8 and Table 9. This data comprises information about resources, the critical project path, the project budget, conditions and consequences, which can be rewards or punishments, failure conditions, an impact of a company's reputation on the probability to obtain future contracts, and software maintenance statistics.

Table 8. Initial scenario data

Resources
Allocated time for project: 5 days, 40 hours; Allocated personnel: two developers; Company current reserves: 45K (CAD); Company public reputation: 5 (7 similar completed projects, 2 of them were delayed)

Critical Project Path
Task 1: Add shopping cart component (20 hours); Task 2: Checkout component (15 hours); Minimum estimated time to complete project: 35 hours; Time reserved for unexpected tasks: 5 hours

Project Budget
Developer salary: 1.5K*2 (CAD); Project manager salary: 2K (CAD); Office space rental: 1K (CAD); Total: 6K (CAD)

Rewards and Punishments
Reward for project completion: 15K (CAD), company public reputation: 1.5
Project delay, 1 day: Fee 1K, company public reputation: -0.5
Project delay, 2 days: Fee 1.5K, company public reputation: -1
Project delay, more than 2 days: fee: 8K, company public reputation: -5
If the source code was stolen (only relevant if "Create user accounts database" has started), company public reputation: -8
If there are too many bugs when software is delivered to client,
2 bugs: company public reputation -0.5; 3 bugs: company public reputation -1
More than 3 bugs: company public reputation -4

Failure conditions
If company financial reserves are less than zero or company public reputation is less than zero, Company will cease to exist.

Impact of reputation
There is possible large contract (potential revenue: 100K (CAD), 20 in company public reputation)

Software Maintenance Statistics
For each hour of software development, there is a 3% probability that a bug will be introduced. Average time to fix a bug is 2 hours
Projected number of bugs: 1.59 +/- 1.25
Each hour a developer tests software, there is 30% probability that a bug will be found if it exists.

Table 9. Connection between reputation and funds

Reputation of company after the project ends, measured in units of company public reputation	<3 Units	3-5 Units	>5 Units
Probability of obtaining the contract	20%	40%	70%

Risk analysis involves gathering data related to a set of possible risks and benefits associated with a situation that the AC has encountered. For example, assume that an AC decision needs to be made when a project manager has decided to add an external contractor to complete the software development project faster. Traditionally, a system administrator executes a process, which consists of two tasks: allowing access of a new user (external contractor) to Concurrent Versions System (CVS) and copying the existing source code files to a new computer where the new user will be working.

Risk Analysis

Data related to a general description of possible risks and benefits associated with adding a new user is presented in Table 10. Data related to a risk of having the code stolen is provided in Table 11.

Table 10. General risk analysis data

Risk analysis data that is related to the situation include
a. ***Task***: new user (contractor) needs permission to get access to project files
b. ***Location***: user accounts database
c. ***Parent Task***: Add checkout component (D)
1. Parent task cannot start without completing it?: yes
2. Affected attributes: Maximum delay, %
d. ***Estimated Duration***: 1 hour
e. ***Responsible actor***: System administrator
f. ***Required resources***:
    a. System administrator
    b. Target system
    c. Required attribute:
        1. Name: operational
        2. Value: True
        3. Attribute parent: System

Table 11. Risk of having the code stolen data

i. ***Risk*** of having the code stolen
1. ***Child risks***: risk of having security compromised
2. ***Parent risk***: risk of receiving reputation penalty
3. Adds to parent risk: no
4. ***Computed likelihood***: 68.6% +- 3%
    a. 10% set value
    b. *1.1 responsible actor is knowledgeable and has sufficient privileges to copy source code
    c. *1.7 $T_{critical}$ for this risk priority target
    d. *0.4 motivation of responsible actor
    e. *1.5 data transfer process is present
    f. *2.6 data transfer target is external
    g. *1.2 inherited from child risk
    h. *1.4 lack of experience with new user
    i. *1.4 Accumulated child risk
    j. ***Computed deviation***:
        i. 20% set value
        ii. *0.10 experience with selected actor
        iii. *1.5 T_{high} priority target
5. Possible to be combined with other risks: yes
6. Risks possible to be combined with:
    a. Risk Type:
        i. Attribute Value: security
7. ***Computed likely effect***: company public reputation -8
8. ***Total impact***: company public reputation -5.49
9. ***Recalculate on*** (or):
    a. Process has completed
    b. Child task has completed
    c. Time passed (1 hour step)
    d. Recalculating security risk for related processes or resources
10. Possible activation of undesirable templates:
    a. Asset being stolen
    b. Malformed access control
    c. Suspicious network (or removable devices) traffic
        i. Origin: affected component
    d. Undesirable functionality
    e. Corrupted system function

Data related to the risk of having the security compromised is presented in Table 12.

Table 12. Risk of having security compromised data

ii. ***Risk*** of having security compromised
1. ***Child risks***: none
2. ***Parent risk***: risk of having the code stolen, risk of having additional work to do (restoring corrupted components)
3. Adds to parent risk: yes
4. ***Computed likelihood***: 43.68% +- 2%
   a. 40% set value
   b. *1.2 responsible actor is knowledgeable and has sufficient privileges to install software that will compromise security
   c. *1.3 Lack of experience with new user
   d. *1.4 $T_{critical}$ for this risk priority target
   e. ***Computed deviation***:
      i. 20% set value
      ii. *0.10 experience with responsible actor
5. Possible to be combined with other risks: yes
6. Risks possible to be combined with:
   a. Risk Type:
      i. Attribute Value: security
   b. Process Attribute: Data Transfer
      i. Attribute Value: active
   c. Process Attribute: name
      i. Attribute Value: create user accounts database(C)
7. ***Computed likely effect***:

Company public reputation: -8

8. ***Total impact***: 0
   a. Risk only increases other risks: true
9. ***Recalculate on*** (or):
   a. Activating processes associated with affected component
      i. Attribute: data transfer active
   b. Recalculating security risk for related processes
10. Possible activation of undesirable templates:
   a. Malformed access control request
   b. Suspicious network (or removable devices) traffic
      i. Origin: affected component
   c. Undesirable functionality introduced
   d. Corrupted system function

Data related to a benefit of completing project without delay is provided in Table 13. Assume that some of this data is based on historical data. An example in this case is the history of the system administrator actions resulting under the AC assumption that the AC that he or she can be trusted from a security perspective. The remainder of the data comprises information about the context of adding a new user, and its associated entities. For example, context information can include the availability of the required resources to add a new user (e.g., system administrator and working computer).

Table 13. Benefit of completing project without delay data

iii. ***Benefit*** of completing project without delays
1. ***Parent benefit***: Benefit of not receiving penalties for delay
2. Adds to parent benefit: no
3. ***Computed likelihood***: 48%+-10%
    a. Mean Computed: 60% set value (adding new user)*0.8 lack of experience with new user
    b. ***Computed deviation***: 10% set value
4. Possible to be combined with other benefits related to delay: yes
5. Benefits possible to be combined with: D (+children), A (+children)
6. Benefits possible to be combined with:
    a. Benefit Type:
7. Attribute Value: productivity
8. ***Computed likely effect***: fee +1K and company public reputation: +0.5
9. ***Total impact***: Reserve +0.48K and company reputation +0.24
10. ***Recalculate on*** (or):
    a. Time passed (1 hour step)
    b. Changing attributes of required resources

The risk analysis data related to this situation, which is informally provided in what follows in natural language, is an instance of the class diagram illustrated in Figure 37, which presents the UML diagram of the basic design model. The informal data description is provided for readability. The data provided by a risk analyst specifies values that are estimates of relevant risk-related attributes.

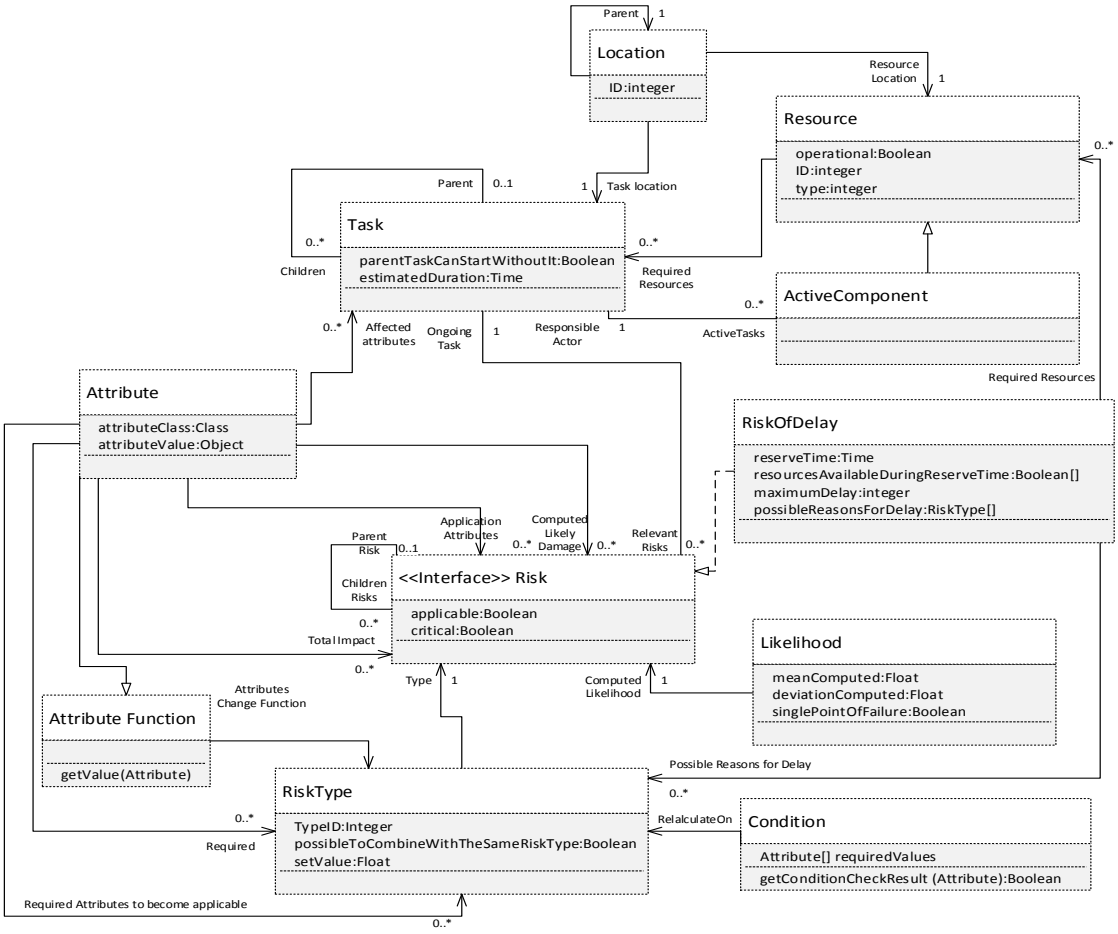


Figure 37. UML class diagram of risk assessment data including risk of delay extension

The importance of provided risk analysis is not the ability of the risk analyst to estimate risk-related attributes, however it is the possibility to relate heterogeneous risks through their consequence and likelihood. The provided UML diagram can be used to assess various risks and combine them in a single frame of reference required to compare them and consequently make decisions based on risk levels. Figure 37 is a UML class diagram for risk analysis extended with the variable *risk of delay*. This diagram contains classes that represent objects describing the risks.

The risk analysis in the basic design model is based on an hierarchy of risks. Some risks are always present. Other risks which are called children risks can only be present if the specific predefined risks called parent risks are present. The scenario description, presented in Table 10, includes contextual information about the process of adding a new user (a-f) and two relevant risks and one benefit: (i) the risk of having the code stolen; (ii) the risk of having the security compromised; and (iii) the benefit of completing the project without delays. Each of the relevant risks and benefits contains information about the parent and child risks and benefits.

A child risk or benefit is a risk or benefit that affects the parent risk. A risk or benefit becomes relevant (computed and involved in the risk analysis) only if there is no parent risk or if the parent risk is applicable to the situation. If the parent risk is irrelevant (e.g., with zero probability and zero distribution), according to the basic design model, the child risks are considered irrelevant and not evaluated.

The variable *adds to parent risk or benefit* assumes a boolean value. If this value is true, then the effects of child risks or benefits are added to the parent risk or benefit list of effects and the parent risk or benefit probability is modified by the function of the child risk, for example, by multiplying the probability of the parent risk to the value of the risk impact. If this value is false, then the child risks or benefits are considered relevant if the parent risk is relevant. The effects and probabilities of the parent risk or benefit are computed and applied independently of the child risks and benefits. The variable *computed likelihood* is a likelihood with which the most likely risk effects will occur. For each process, there is a predefined set value and a set of modifiers that change the computed likelihood. The variable *computed deviation* is a measure of uncertainty, which can be increased or reduced by either

corroborating or lowering the belief of AC in the user activities or the process. The variable *total impac*t is a utility related to the risks or benefits. For non-critical risks, i.e., a risk that has likely effects greater than the current reserves of a system for which AC is provided, this variable is taken into account during making an access control decision. If a risk is critical, then the access control decision has to either accept this risk or label the process for which this critical risk is too high as "not permitted". The variable *recalculate on* is a set of conditions that trigger the re-evaluation of the risk or benefit. Whether risks are additive is defined by AC designers. If two risks are additive, their utility is calculated by summing multiplication of a risk impact and risk probability, the deviation is calculated as a sum of multiplication of deviation and risk impact and risk probability of corresponding risks.

All attributes presented after the *recalculate on* variable in the provided tables are risk or benefit-specific attributes. For example, the variable *risk of having the code stolen* has the attribute *possible activation of undesirable templates*. This attribute indicates that the AC assumes that one or more processes corresponding to such undesirable templates are indeed active and need to be stopped. The AC does this operation by increasing the computed likelihood and decreasing the computed deviation.

The risk analysis indicates that two risks and one benefit are relevant to the analysis of the situation: the risk of having the code stolen, the risk of having security compromised, and the benefit of completing project without delays. These risks and one benefit are associated with the following QRM: financial, measured in K (CAD) and reputation measured in units. While risks are associated with the loss of the reputation, the benefit is associated with the gain in reserves related to reputation and finances. Thus, it would be helpful to establish a relationship between them.

As described in the scenario's initial data, the company has reserves in the financial dimension (45K CAD) and reserves in the public relations dimension (5 units of reputation), which are presented in

Table 8. Assuming the system for which AC is provided does not fail, neither reserves in the financial dimension, nor in the public relations dimension can be less than zero. QRMs forecasted at the end of the project are 53K (CAD) and 5.5 units of reputation. The forecast is based on the reserves, the project completion reward, and the penalty for delaying a software development project for one day and having two bugs in the code. The probability of receiving a large contract is projected to be at 70% according to the projected level of reputation, 40% if half a unit of reputation is lost, and 20% if 2.5 units of reputation are lost at the end of the project. The potential revenue for this contract is 100K and 20 units of reputation, which moves the company significantly further away from its failure conditions and, therefore, this revenue is desired.

Building the correlation between reserves of financial and public relations domains is calculated based on the following facts: the loss of five units of reputation is equivalent to a loss of 45K because if either financial or reputation reserves decrease to zero, the company ceases to exist. The projected change of reputation at the end of the project (0.5 units) is an equivalent of gaining 70K and 14 units of reputation, which is the mean of the QRMs the company is expected to gain and is calculated in this case by multiplying the probability and the QRMs of the effect. In this case, the company at the end of the project gains 0.5 units of reputation. In addition, a zero change in reputation results in gaining 40K CAD and 8 units of reputation and losing two or more units of reputation by the end of the project, which results in obtaining 20K CAD and 4 units of reputation. If one plots a graph of the correlation between the reputation and financial reserves, these facts represents points, and a linear approximation of this graph is used as one of the sources to calculate the current QRM exchange ratio multiplied by the number of reputation units that is considered the equivalent of 1K of financial reserves. Consequently, a linear approximation of these points plotted as a graph results in a ratio of one unit of reputation, which is equivalent to 19.2K CAD in reserves. The linear approximation of the constructed correlation graph between delay (hours) and financial QRM, measured in K (CAD) according to the known and projected facts indicates that the estimated correlation between a financial QRM and delay is 17.6K CAD per hour.

Appendix E - Process of the Patient-Privacy Case Study

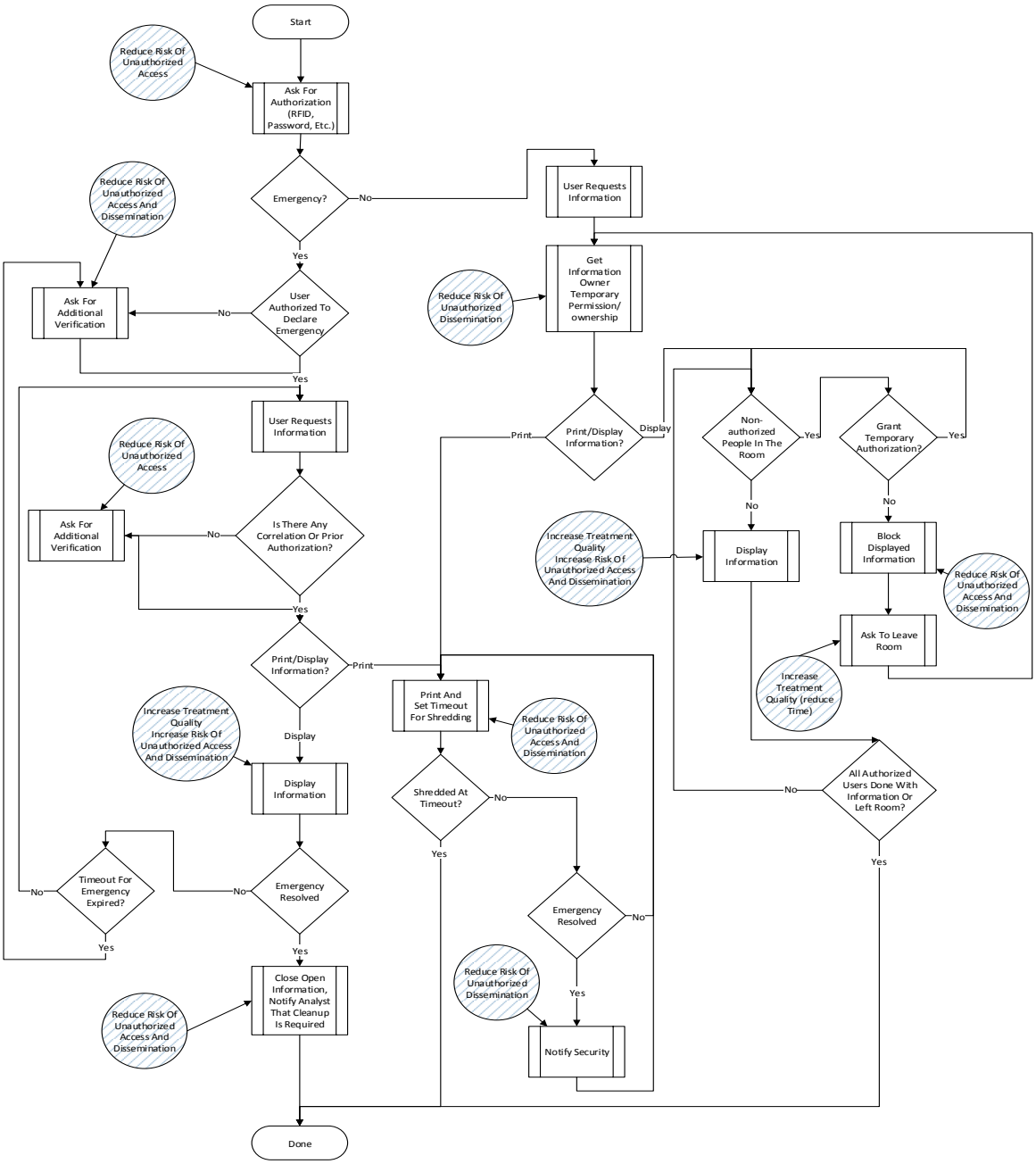


Figure 38. Process of the Patient-Privacy Case Study