electronics

Article

Design and Implementation of a UMLRPAsec-Extension for Robotic Process Automation

Anastasiya Kurylets and Nikolaj Goranin *

Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius, Lithuania; anastasiya.kurylets@vilniustech.lt
* Correspondence: nikolaj.goranin@vilniustech.lt

Abstract: Ensuring RPA (robotic process automation) security is a critical aspect when developing and operating automated software robots. One of the key steps for developing secure software robots is the design stage: the identification and specification of the requirements for the security of the system, the description of system precedents, the interaction between the classes involved in the robot being created, etc. Designs using security-oriented formal modeling languages, such as the UMLsec extension of UML, involve not only a visual representation of diagrams but also the possibility to focus the attention on security issues. However, currently, in the scientific community, there is no possibility of using stereotypes specialized for robots—a mechanism for expanding the unified modeling language that would explicitly reflect a specific problem in the subject area. In this article, we propose that the UMLRPAsec-extension for RPA can be used to model security in the RPA context.

Keywords: RPA; UMLRPAsec-extensions; class diagram for RPA; RPA-stereotypes

check for updates

Citation: Kurylets, A.; Goranin, N. Design and Implementation of a UMLRPAsec-Extension for Robotic Process Automation. *Electronics* **2024**, *13*, 769. https://doi.org/10.3390/electronics13040769

Academic Editors: George Hatzivasilis, Sotiris Ioannidis, Vasileios Mavroeidis and Vasilis Katos

Received: 12 January 2024
Revised: 8 February 2024
Accepted: 13 February 2024
Published: 15 February 2024

1. Introduction

Robotic process automation (RPA) is a family of business process automation technologies based on the use of software robots and artificial intelligence. The software robot reproduces human actions by interacting with the interfaces of information systems. The script of its behavior is programmed by the developer based on observations of a real user performing a task using computer technology [1]. The robotic process automation (RPA) does not represent neither a physical nor mechanical robot, even if it brings to mind a vision of some electromechanical machine [2].

The usage of robotic process automation (RPA) in organizations has rapidly increased in recent years and is projected to grow in the foreseeable future by 20–30% per year or USD 3.97 billion in 2025. RPA growth has also been predicted to occur at a rate of 32.8% from 2021 and 2028. Organizations, such as Diogo da Silva Costa, Henrique Sao Mamede, and Miguel Mira da Silva, are adopting RPA with the motivation to reduce costs and improve efficiency, productivity, and service quality [3].

Fajet Kositere's thesis "Robotic Process Automation (RPA) and Security" [4] states that "...there are many potential security weaknesses in the implementation of this software, and the implementation of RPA must also be accompanied by a proper and thorough risk analysis that establishes policies and procedures that are consistent with RPA security best practices..." which supports the need for more research into improving RPA security.

One of the critical steps for system security is the design and specification stages, which are typically documented using unified modeling language (UML). Unfortunately, this formal language lacks integrated mechanisms for security requirement specification. This limitation has been overcome by the introduction of UMLsec extension [5,6], introducing new stereotypes, tags, and conditions. Stereotypes are used to define connections as encrypted ("encrypted"), link data with security requirements on the logical level (e.g., "secrecy" or "integrity"), and define information security guidelines for a system (e.g., "secure links"). A complete description of the main UMLsec stereotypes and tags can be found in [6].

The main problem solved by the development of UMLsec extensions for RPA is the insufficient level of the consideration of security aspects in the design of robotic processes and systems. The developed UMLsec extensions allow developers to consider security features and risks when designing and implementing robotic processes and to model and analyze threats, vulnerabilities, and security measures. This offering improves the overall security of RPA and reduces the likelihood of data security incidents. The innovation of this article lies in proposing unique, previously unavailable UMLsec extensions in the RPA. The proposed extensions can be used both for analyzing general security requirements in the RPA domain and adapting them to a specific industry in which robotics technology is used. The proposed work solves the complex problem of RPA systems design, as RPA systems interact with various information systems, databases, web services, and other components of the information infrastructure, which creates complex threat scenarios and attack opportunities that also require a systematic approach to security.

However, like any other technology, RPA possesses some risks, especially in the area of security. To ensure the security of automation processes, UML security extensions can be used while developing RPA robots.

UML security extension (UMLsec) is a combination of design methodology and a set of notation and tools for analyzing and designing secure systems.

The use of UMLsec when developing an RPA robots can provide the following benefits:

1. Security Modeling: UMLsec allows you to explicitly model security aspects as a part of the RPA robot development process. This helps improve understanding and provide transparency regarding security in an RPA system.
2. Threat Analysis: UMLsec provides specific diagrams and concepts for modeling security threats, which allows you to identify and describe the main threats associated with the RPA robot. Threat analysis in the early stages of development allows you to take security measures in advance.
3. Defining the Security Policy: By using UMLsec, you can explicitly define the security policy for the RPA robot. This includes defining access rights, encryption and authentication policies, data integrity controls, and other security aspects.
4. Interoperability with other models: UMLsec can be used in combination with other models and methodologies to develop an RPA robot. For example, it can be integrated with UML activity diagrams or UML sequence diagrams to specify the secure interaction of the RPA robot with the surrounding system or other entities.

However, in order to apply UMLsec in a specific subject area, it is typically necessary to create additional extensions. Extension mechanisms allow you to define new elements based on existing ones in a unified way. Unfortunately, currently, there are no RPA-specific UMSsec extensions.

There are three common extensibility mechanisms defined by the UML: stereotypes, tagged values, and constraints.

In this article, we propose the UMLRPAsec extension for RPA with a set of unique stereotypes. The use of the proposed UMLRPAsec extension for RPA allows us to improve the RPA system security, identify risks, and take appropriate measures to reduce them.

This article is organized as follows: Section 1 gives a general overview of security importance while designing RPA systems and motivates the need for the RPA-specific UMLsec extension; Section 2 provides analysis of prior work in this area; in Section 3, the proposed UMLRPAsec extension is presented, while in Section 4, the verification of the extension provides several sample cases of code generation from the domain-specific diagrams; finally, this article is finalized with our conclusions.

2. Prior and Related Work

The idea of using a classical life cycle model to represent the entire implementation of an RPA development process is widely accepted in scientific literature. This life cycle contains six to seven phases, depending on the literature, and is very well suited to combining different concepts and implementation.

2.1. General RPA Development Process

According to [7], the following phases of the RPA life cycle can be distinguished:

1. Analysis Phase. This phase consists of analyzing and determining the viability of carrying out the automation of a certain process by means of a detailed analysis of the effort involved in the self-motivation of such process considering the execution characteristics of the process itself.
2. Design Phase. The process design phase begins for those processes that have passed the previous feasibility analysis. The purpose of this phase is to detail the set of actions, data flow, activities, etc., that must be implemented in the RPA process.
3. Construction Phase. This phase consists of implementing each of the automatable parts of each process identified in the design phase.
4. Deployment Phase. The robots obtained as a result of the construction phase need an environment to be executed, just as a human operator needs an environment to perform their work. This environment, in the context of RPA, usually corresponds to a computer that has an installation of one or more information systems. Each robot must be executed in its own execution environment since the replacement between the human operator and software is direct.
5. Control and Monitoring Phase. Once the robots are deployed in their respective execution environments, this phase oversees the controlling and monitoring performances of each robot. In this phase, the execution of robots is launched, it stops in case of serious errors, the execution status is monitored, etc., until they have finished their tasks.
6. Evaluation and Performance Phase. The last phase of the process consists of evaluating robot performance.

2.2. Role and Perspectives of UML and UMLsec in RPA Development

The design stage is one of the most important stages in the life cycle of any product as well the most important phase for defining security aspects [8]. UML (unified modeling language) is a modeling language that is widely used for designing software systems, including robotic process automation (RPA) solutions. Analyzing the experience of the practical application of UML in various fields, which is demonstrated in the scientific work of Ozkaya M [9], the following reasons for application in the field of RPA were identified:

1. Visualization and documentation: UML (unified modeling language) provides powerful tools for visualizing and documenting system architecture. Using UML makes it easy for developers and stakeholders to understand the structure and functionality of the RPA robot. This helps in creating clear and understandable documentation that can be used to facilitate communication, train new developers, and provide further support for the system.
2. Requirement-Based Design: The RPA robot development process should begin with defining the requirements. Using UML allows you to formalize and structure requirements to create models that reflect the relationships and interactions between system components. This helps to significantly reduce the risk of misunderstanding the requirements and provides a clearer understanding of the robot's functionality.
3. Identify potential problems: UML provides the ability to design a system at a higher level of abstraction, allowing developers to specify and test the logic and potential problems associated with the RPA robot. Modeling at the UML level can reveal problems such as code duplication, irrational data structure, or poor coupling between

components. This allows you to make adjustments to the robot's architecture at the early stages of development and avoid problems in the future.

4. Increased scalability and flexibility: UML allows you to develop models that support system extensibility and flexibility. This is especially useful in the case of RPA robots, which often work with different systems and processes. Using UML allows you to create modular and reusable components, which makes it easier to make changes and scale the system if necessary.
5. Improve quality and reduce risks: Designing an RPA robot based on UML allows developers to analyze and evaluate the system in the early stages of development. This helps identify potential problems and improve quality and reliability. Preliminary analysis and compliance testing help reduce risks and potential errors in robot performance and security.

The strongest argument, from a security perspective, in favor of using UML design is not only the ability to identify potential problems but also the possibility for the "visual display" of options for correcting them.

Several basic types of UML diagrams [10] can be identified, which can be used when designing an RPA robot.

Use Case diagrams: These are diagrams that help to identify the basic functionality of the robot and its interaction with the environment. A Use Case diagram allows the identification of actors (users or systems interacting with the robot) and their interactions with the robot.

Activity diagrams: Activity diagrams are used to visualize the sequence of actions that a robot performs. They can represent robot steps, decision conditions, and control structures such as loops and branches.

Sequence diagrams: Sequence diagrams allow one to model the interaction between various system components, including the interaction with external systems or the user. Using a Sequence diagram, you can show the sequence of messages and method calls between objects.

State Machine diagrams: State Machine diagrams help in modelling the various states and transitions of a robot. They allow you to determine how the system reacts to external events and how it changes its state.

Class diagrams: Class diagrams are used to model the structure of classes in a system. This allows you to determine the main components of the robot, their properties and methods, as well as their relationships.

Component diagrams: Component diagrams are used to model the structure of components in a system. In the context of an RPA robot, this can be useful for identifying individual components of the robot, such as automation modules, libraries, or external systems with which the robot interacts.

Deployment diagrams: Deployment diagrams are used to model the physical placement of system components and their relationships. In the case of an RPA robot, this may include hosting the robot on a server or in the cloud, as well as communicating with other systems or devices.

Package diagrams: Package diagrams are used to organize classes and other model elements into logically related groups. This can be useful for separating the robot's functionality into separate modules or packages, making the system easier to understand and maintain.

Communication diagrams: Communication diagrams allow for modeling the interactions between objects in a system. In the context of an RPA robot, this can be used to show the interactions between various components of the robot, as well as between the robot and other systems or users.

Timing diagrams: Timing diagrams allow you to model the timing behavior of a system, including delays, synchronization, and contention between different processes. In the case of an RPA robot, timing diagrams can be used to model timing constraints and synchronize the robot's actions.

However, in order to model security aspects, it is necessary to use the UML modeling language extension UMLsec. Based on [5], the following advantages of applying UMLsec can be identified:

Integrating security into system design: UMLsec allows security requirements to be integrated into system modeling early in the project. This facilitates the early identification of potential vulnerabilities and security issues, allowing remediation measures to be taken early in development.

Automation of security analysis: UMLsec provides automated methods and tools to assess whether the model is secure. This allows you to detect potential vulnerabilities and security threats, such as attacks on models, information leaks, integrity violations, and others.

Extensibility: UMLsec has a flexible architecture that allows you to expand and complement its functionality according to the needs of a specific project. Models and security specifications can be tailored to the specific requirements of a system.

UML compatibility: UMLsec is fully compatible with the UML modeling language, allowing it to be used with existing development tools and techniques. This makes the security integration process smoother and more convenient.

Support of security standards: UMLsec complies with security standards such as Common Criteria and ISO/IEC 15408. This ensures high reliability and quality of system security analysis.

Standardized approach to security modeling: UMLsec defines standard specifications and modeling rules to address security aspects.

Risk Management: UMLsec helps to identify and analyze potential vulnerabilities and security threats, thus allowing you to identify and assess risks, resulting in making informed decisions about what security measures to take in order to reduce risks.

Documenting Security Requirements: UMLsec allows security requirements to be explicitly specified in the form of models and documentation. This facilitates understanding and communication between development team members and stakeholders regarding the required level of system security.

Attack analysis support: UMLsec allows you to model and analyze various types of attack on a system.

Possibility of integration with other security methods: UMLsec can be integrated with other security analysis tools. This allows you to take an integrated approach and gain a more complete picture of system security.

In general, using UMLsec helps to create more secure systems. It allows security requirements to be considered at every stage of development, from modeling and requirements analysis to system development and testing. This helps to reduce risks and protect against security threats.

2.3. Main Aspects of RPA Security

In the context of RPA, UMLsec can be used to analyze and assess the security of automation processes. However, it is important to note that UMLsec was developed for the security analysis of classical information systems and is not adapted for RPA. RPA differs from traditional systems because it involves the use of robots and scripts to automate business processes. It is appropriate to use the UMLsec methodology to analyze these aspects of RPA security, but the methodology may need to be adapted to consider the specifics of RPA systems. The main aspects of RPA security were identified in our previous work in the form of ontology [1]. In Table 1, the main security aspects and the corresponding requirements are provided.

Table 1. RPA security aspects.

Security Aspect	Requirements
Credential protection	The way credentials are transmitted and stored in the RPA system must be reviewed to ensure their security.
Access control	It is necessary to evaluate what resources and functionality are available to each robot and user of the RPA system. Use of access control concepts and mechanisms, such as permissions or role models, to ensure adequate levels of access; integration with IAM (identity and access management).
Malware protection	It is necessary to consider the use of malware detection mechanisms and antivirus software to keep RPA robots secure. It is also necessary to restrict the access rights of robots and users to prevent the introduction of malicious software.
Ensuring data integrity	It should be ensured that data integrity verification mechanisms are enabled to prevent unauthorized or incorrect changes to data. Also, creating backup copies of the robot script data and periodically restoring them to detect and correct data corruption.
Monitoring and logging	Implementation of a monitoring and logging system that will record in detail the actions of robots, as well as any anomalies and suspicious activities. This will enable detection and warning of possible security breaches.
Updates and patches	Timely updates and installation of patches for the RPA platform, tools, and components corrects security vulnerabilities and reduces the risk of exploitation.
User training and awareness	Providing training and guidance on user security related to the use of RPA.
Checking for confidential data leaks	Regularly checks for confidential data leaks. Using data leak detection and Internet monitoring tools to find information.
Error and exception management	Development of an error and exception management strategy to ensure security and reliability in the event of failures or unexpected behavior of robots. An additional way to identify and correct the problem is through logging and reporting.
Security testing	Conduct regular security tests of the RPA system, including penetration testing to identify vulnerabilities and weaknesses.
Data backup and recovery	Regular backup of data used by robots and development of a disaster recovery plan for quick recovery and minimal downtime for business processes continuity.

2.4. Process of UMLsec Extention for RPA Development

The development of a UML security extension for RPA will require a specific approach and set of tools as follows:

1. Identify Security Requirements: Determine the specific security requirements associated with RPA [1].
2. Development of a UML RPA security metamodel.
3. Develop an extension to an existing UML tool.
4. Application of security extension in modeling. Using the new UML Security Extension in RPA process modeling to incorporate security aspects and architectural decisions early on an early stage.
5. Assessment of models.

During the design of an RPA robot using UMLsec, the following extension mechanisms can be used:

- Annotations: The annotation engine allows you to add additional semantic annotations to UMLsec class diagram elements to indicate the security features of each element. For example, you can annotate classes or attributes that indicate required access levels or security requirements.
- Specific stereotypes: UML stereotypes allow the creation of specific metamodels that can be applied to UMLsec class diagram elements. It is possible to define your own

stereotypes that represent security concepts such as Threat, SecurityPolicy, SecurityControl, and others and to apply these stereotypes to the corresponding elements.

- Additional relations: Additional relations can be used to represent the corresponding relationships between security elements. For example, it is possible to use the "realize" relationship to link between a class representing Threat and a class representing SecurityControl to indicate which security controls are adopted to prevent a given threat.
- Additional diagrams: In addition to the class diagram, other types of UML-sec diagrams can be used to model the security of the RPA robot.

The mentioned extension mechanisms make it possible to supplement the standard UMLsec semantics and adapt it to the specific security requirements of RPA robots. They allow for specifying additional security details and connections between elements so that relevant security aspects can be considered when designing and developing an RPA robot.

2.5. Review of Existing UMLsec Extension in Different Domain Areas

The search performed on Web of Science and other resources did not show any RPA-related UMLsec extensions or similar solutions. Queries "RPA"AND"UMLsec"AND"UML", "RPA"AND"UMLsec", "RPA"AND"UML", "Robotic Process Automation uml", "Robotic Process Automation uml-sec", and "Robotic Security Process Automation" in Web of Science were performed and did not yield any results. Still, there are a number of recent research papers presenting UMLsec extensions in other areas. Thus, in the paper "Modeling data protection in fog computing systems using UMLsec and SysML-Sec", the author Jan Laufer suggests the application of the UMLsec extension for areas such as the Internet of Things, cloud computing, and edge computing [8]. Given the contiguity of RPA technology with the above technologies, one can speculate about the possibility of applying UMLsec and RPA among others. There exist a number of UMLsec extensions in other research areas that demonstrate the importance of the research direction. In the article "IoTsec: UML extension for Internet of things system ssecurity modelling" by David Alejandro Robles-Ramirez, UML extensions for modeling IoT (Internet of Things, IoT) applications are proposed [11]. The article also introduces a UML extension which involves the security issues encapsulated within a nomenclature, UML stereotypes to model common actors and UML notation extensions. In "UML2Merge: a UML extension for model merging" by Farias, K., de Oliveira Caval-cante, T., José Gonçales, L., and Bischoff, V., the authors propose the use of UML2Merge, which is a UML extension for expressing merging relationships [12]. The results are encouraging and show the potential for using UML2Merge to express the evolution of UML models through merge relationships.

It is necessary to mention that a lot of UML extensions for different modern technologies are presented not in scientific but technical literature. The following extension can be found:

UML for Web Applications: UML extension designed for modeling web applications adds elements and diagrams related to web development such as use case diagram, class diagram and component diagram [13,14].

UML for Cloud Computing: UML extension specialized for cloud computing modeling adds elements and diagrams related to describing the architecture of cloud applications, such as component diagrams, deployment diagrams, and environment diagrams [15,16].

UML for Big Data Analytics: UML extension designed for modeling big data analytical systems adds elements and diagrams that allow you to describe data, analytical algorithms, and data processing flows, such as sequence diagrams, component diagrams, and activity diagrams [17].

Each UML extension for a specific IT technology has its own characteristics and specific elements and diagrams. These extensions facilitate the modeling process and are tailored to relevant development and architecture areas. The lack of a UML extension for RPA technology was the basis for starting research in this area.

3. Proposed UMLRPAsec Extension

In this research, the choice of diagrams for designing RPA systems with security requirements was based on two main aspects: describing system components and defining possible states when threats are detected. The class diagram and the activity diagram were found to be the most appropriate. The class diagram is useful for defining system components and their relationships, including security components like authentication and authorization, while the activity diagram helps to represent the different states the system can be in, including security-related states such as authorized and unauthorized access. Other diagrams could be important in the design of RPA as well and cover other aspects that are less related to information security and can be one of the stages of a future research.

Further research could focus on developing additional RPA-specific diagrams, which could be a valuable step for future exploration in this field. The proposed UMLRPAsec extension includes RPA and RPA security-related activity and class diagrams, specific stereotypes, and a UML metamodel that are described below.

3.1. UMLRPAsec Class Diagram

A class diagram allows for organizing the logic and structure of an RPA robot. Classes represent various components and modules of a robot and define their properties and methods. This helps to differentiate the functionality and control of different parts of the robot, making the system easier to understand and maintain. A class diagram provides a visual representation of the architecture and components of an RPA robot. This makes it easier to understand the system and make the necessary changes.

From a security point of view, constructing a UMLsec class diagram when creating an RPA robot allows you to describe the structure of the RPA robot and define the classes, their properties and methods, as well as the relationships between them. This gives a clear picture of how the different components of the robot interact with each other. Also, the UMLsec class diagram allows you to take into account security aspects when designing an RPA robot. It can help identify classes that require special security measures and describe the appropriate security measures. This is important because RPA robots can access sensitive data and perform mission-critical operations. The proposed class diagram for RPA can be seen in Figure 1.

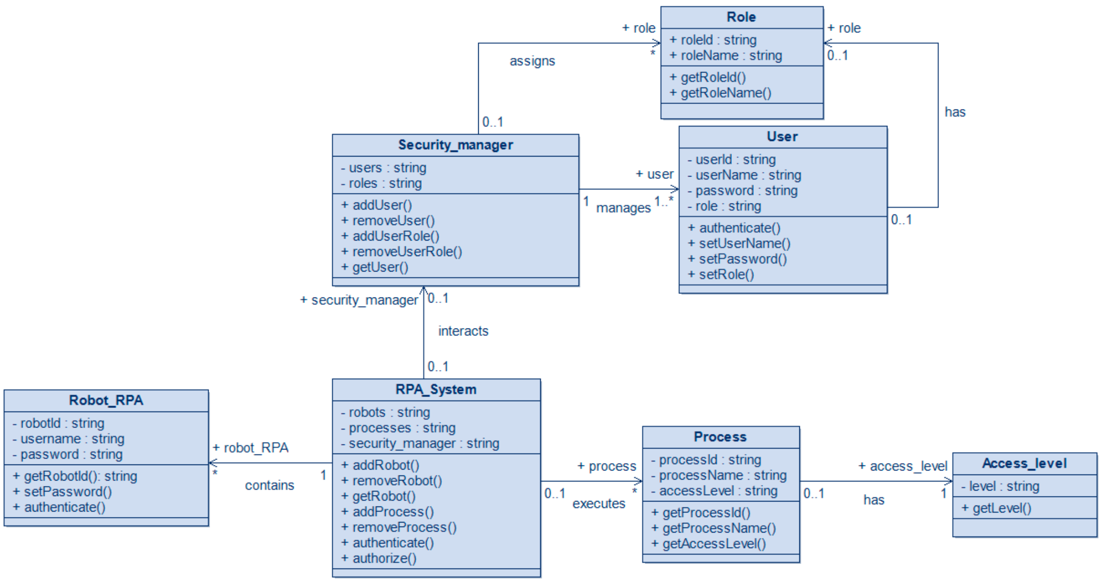


Figure 1. UMLsec class diagram for RPA.

On this diagram, different classes and relationships between them related to RPA robot security are presented.

Rpa_System: This class represents the RPA system as a whole. It contains a list of robots, a list of processes, and a *security_manager* object, which is responsible for authenticating and authorizing users in the system. The class offers methods for adding and removing robots, adding and removing processes, authenticating users, and authorizing their access to processes.

Robot_rpa: This class represents a robot in an RPA system. It has *robot_ID*, *username*, and *password* attributes. The class provides methods for getting the robot's ID, authenticating the robot, and setting its password.

Process: This class represents a process in an RPA system. It has *process_Id*, *process_name*, and *access_level* attributes. The class provides methods to obtain the process ID, name, and access level.

Security_manager: This class is responsible for managing users and roles in the RPA system. It has *users* and *roles* attributes. The class offers methods for adding and removing users and for adding and removing user roles.

User: This class represents a user in the RPA system. It has *userID*, *username*, and *password attributes*. The class provides methods for obtaining the user ID and name and setting the user's password.

Role: This class represents a role in the RPA system. It has *role_Id* and *role_name* attributes. The class provides methods to obtain the role ID and name.

Access_level: This class represents the access level of a process in the RPA system. It contains the *level* attribute—the access level. The class provides a method to obtain the access level.

The diagram uses the following element visibility parameters:

private (private, available only inside the class) - set by the "minus" symbol (-);

public (public, available to all) - specified by the "plus" (+) symbol.

The following relationships were defined:

The relationship between *Security_manager* and *Role*: "*assigns*".

Relationship between *Security_manager* and *User*: "*manages*".

Relationship between *RPA_System* and *Security_manager*: "*interacts*".

Relationship between *RPA_System* and *Robot_RPA*: "*contains*".

Relationship between *RPA_ System* and *Process*: "*executes*".

Relationship between *Process* and *Access_level*: "*has*".

Relationship between *User* and *Role*: "*has*".

In addition, the association has multiplicity:

"1" to "*"- 1 to many.

"0..1" to "*"- 0 or 1 to many.

"0..1" to "1"- 0 or 1 to 1.

"0..1" to "*"- 0 or 1 to many.

"1" to "1..*"- 1 to 1 or many.

"0..1" to "0..1"- 0 or 1 to 0 or1.

The provided list of classes and relations is not finite and can be extended depending on the specific security and project requirements and project features. The diagram can be extended and adapted to specific needs.

3.2. UMLRPAsec Activity Diagram

Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. For a clear example of state transition, a fragment, presenting the beginning of work with the robot, was chosen (Figure 2).

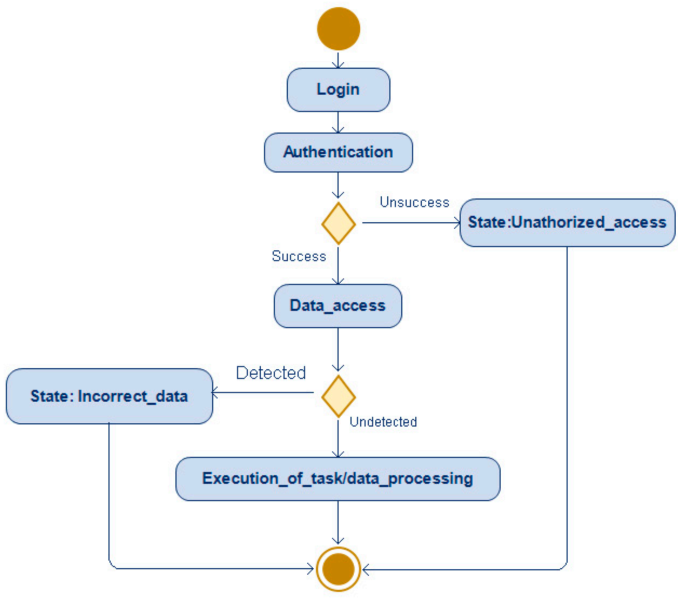


Figure 2. UMLsec activity diagram for RPA.

START: This is the initial state where the automated process begins.

Login: The robot performs an authentication process to gain access to the system or application where the data used by the process are stored.

Authentication: In this state, the robot performs identification and authentication for further authorization. Only properly authenticated robots can access data.

Data_access: After successful authentication, the robot gains access to the data necessary to complete the task. It is important to ensure access control so that only authorized robots have access to the desired data.

Incorrect_data: If errors or inconsistencies are detected in the data, the robot enters this state to process and correct the problems.

State: Unauthorized_access: The robot enters this state if the robot has gained unauthorized access or a security violation is detected. This situation could be caused by, for example, the unauthorized use of credentials or mismatched access rights.

Data_processing/Execution_of_task: The robot processes the received data, performs tasks, and interacts with the system or application. The robot performs planned actions and tasks using the processed data.

END: The robot enters this state once the task is completed. Actions to clear data, save results, or other final operations can be performed here.

This RPA security activity diagram presents a general concept that can be tailored to the specific needs and requirements of an RPA project. The context and internal security policies of a specific organization must be taken into account when designing and implementing secure automated RPA processes.

3.3. *Stereotypes for RPA*

The UMLsec was initially proposed in the article "UMLsec: A UML Profile for Secure Systems Development" by Jan Jurjens [6]. It introduced the list of security stereotypes that can be considered as fundamental ones for UMLsec, followed by a description of them: "Internet, encrypted, LAN, wire, smart card, POS device, issuer node, secure links, secrecy, integrity, high, secure, dependency, critical, no down-flow, no up-flow, data, security fair exchange, provable, guarded, access guarded". After reviewing this list, it was decided to create our own stereotypes specific to the RPA domain. There was no overlapping between original UMLsec stereotypes and the newly proposed.

To model the security of RPA robots using UMLsec, we must define specific stereotypes that represent security concepts specific to that domain.

1. <<*robot*>>: This stereotype can be applied to a class representing an RPA robot. It may have security-related properties such as access level, authentication rules, or valid operations that the robot can handle.

2. <<*securitycontrol*>>: This stereotype can be applied to a class or element representing security controls in an RPA robot. It can have properties that define types of controls, such as encryption, authentication, or security auditing. This stereotype can also be used to highlight relevant elements in class diagrams and sequence diagrams.

3. <<*threat*>>: This stereotype can be applied to a class representing potential security threats that an RPA robot might encounter. It can be used to identify and model different types of threat such as malware, infrastructure attacks, or privacy threats.

4. <<*securitypolicy*>>: This stereotype can be applied to a class that represents a security policy that defines the rules and restrictions that apply to RPA robots. It may have properties related to security settings, such as access settings, security templates, or authentication requirements.

5. <<*securityrequirement*>>: This stereotype can be applied to a class representing a security requirement for an RPA robot. It may include properties such as a requirement description, severity classification, or due date. This stereotype helps to establish clear security requirements for RPA robot development.

6. <<*securityaudit*>>: This stereotype can be applied to a class representing a security audit for an RPA robot. It may contain properties that define audit frequency, security check methods, and reporting. This stereotype allows us to model the security audit process for an RPA robot.

7. <<*sensitivedata*>>: This stereotype can be applied to a class or attribute representing confidential or sensitive information that is processed by an RPA robot. It may include properties that define the level of confidentiality, encryption mechanisms, or access requirements for these data.

8. <<*accesscontrol*>>: This stereotype can be applied to a class or element that represents the access control mechanisms used in an RPA robot. It may include properties that define access level, user roles and rights, or the grouping of access to certain robot functionality.

9. <<*incident*>>: This stereotype can be applied to a class representing a security incident that has occurred with an RPA robot. It may contain properties such as incident type, description, date and time, and actions taken to respond to the incident. This stereotype helps us to track and manage security incidents related to the RPA robot.

10. <<*securityprotocol*>>: This stereotype can be applied to a class representing the security protocol used by an RPA robot. It may include properties that define the encryption algorithms used, authentication mechanisms, and data integrity methods. This stereotype allows for modeling and managing secure communication protocols for the RPA robot.

11. <<*vulnerability*>>: This stereotype can be applied to a class representing a security vulnerability associated with an RPA robot. It may contain properties such as vulnerability type, description, and recommended mitigation measures. This stereotype helps us to identify and manage security vulnerabilities in RPA robots.

The use of these stereotypes in RPA robot security modeling is intended to create a clear and structured security model that reflects the specific requirements and security mechanisms of this specific domain.

3.4. RPA Metamodel

Another tool for defining the structure and connections between the various components of an RPA robot can be a metamodel for RPA. It allows the expression of basic concepts, abstractions, and classes that make up a robot, as well as their relationships and interactions. This definition of structure and connections allows a better understanding of the system and its components.

The UML security profile for RPA is an extension of the standard UML modeling language designed to model security aspects in the context of developing and operating RPA robots. This profile adds new elements, relationships, and stereotypes specific to RPA security to provide a more complete view of security requirements, policies, vulnerabilities, and procedures in the context of a robotic process.

The presented metamodel (Figure 3) is constructed using some of the previously proposed stereotypes.

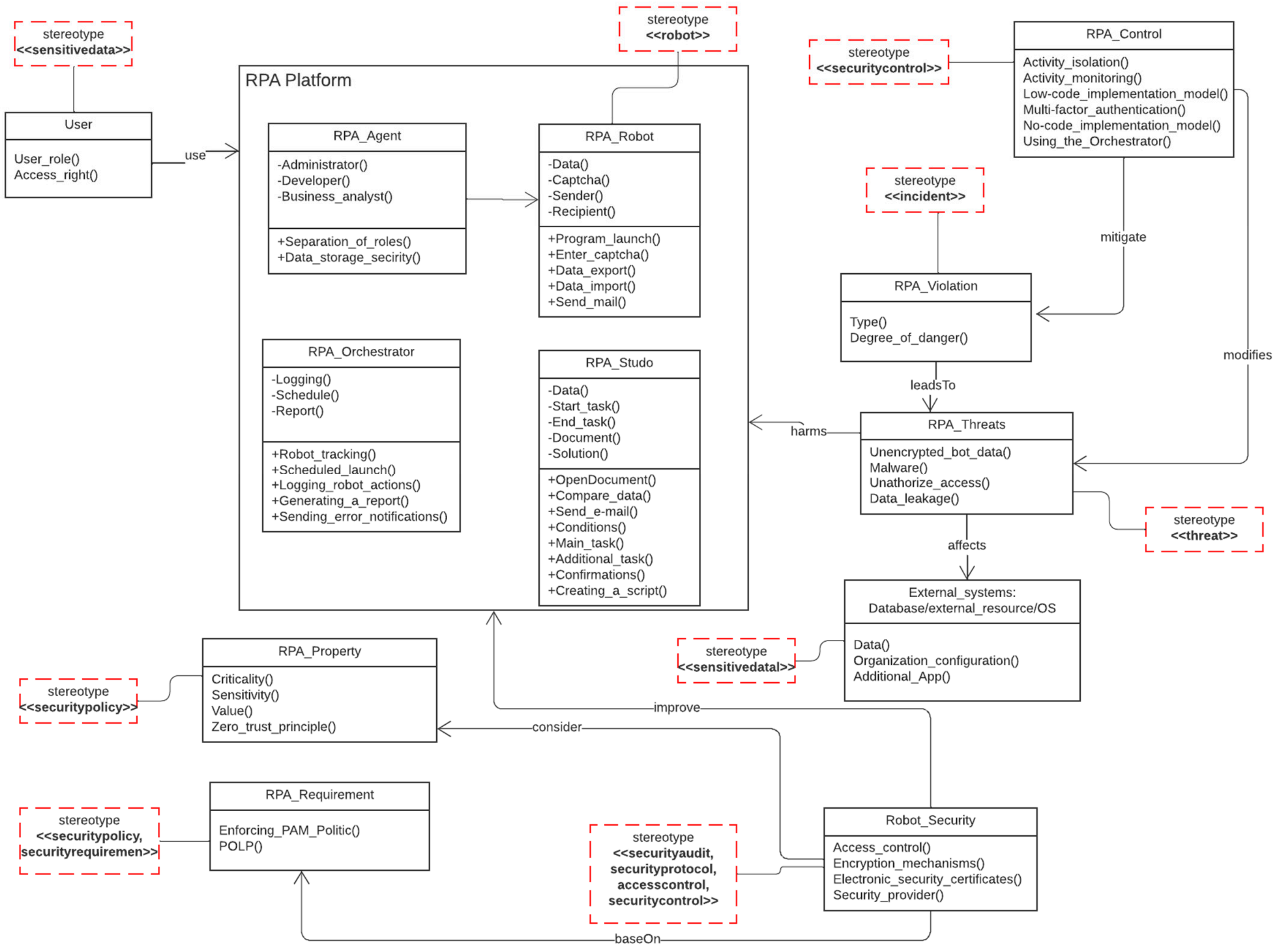


Figure 3. UML metamodel for RPA.

User: This class represents the user of the RPA system and contains information about the user's role, access rights, and other attributes.

Robot_Security: This class represents a security service provider that provides authentication, authorization, and encryption functions and can be integrated with the RPA robot to ensure secure operation.

RPA_Control: This class represents the security controls that are applied to protect the RPA robot. This may be an authentication mechanism, data encryption, integrity control, and other security techniques.

RPA_Violation: This class represents security breaches that can occur in an RPA system and contains information about the type of violation and danger level.

RPA_Requirement: This class represents the security requirements that must be considered when designing and developing RPA robot. This includes requirements for authentication, authorization, encryption, and other security aspects.

RPA_threats: This class represents possible security threats that an RPA robot might encounter.

RPA_Property: This class represents the basic properties for building a secure robot.

External sources: This *OS* class represents possible third-party resources that can interact with robot components.

The following four classes are included in *box RPA-Platform*:

RPA_Studio—the space dedicated to process design. It describes the actions that must be performed to achieve the result, e.g., open a document, compare data for the past day, and send a report to a specified email. The list of processes includes conditions, main and auxiliary tasks, confirmations, and other elements.

RPA_Robot—a performer acting in accordance with the given algorithm. It launches the program, enters captcha, exports data, etc.

RPA_Orchestrator—one of the most important components. It is an application for administering virtual employees, which monitors the work of all robots, since RPA technology is designed in such a way that one robot can perform several processes, but large-scale automation in large enterprises requires hundreds and thousands of robots. Orchestrator creates an effective simultaneous operation of all robots, clearly managing and monitoring the work of each. Orchestrator is also used to distribute tasks among robots, track their completion, and flexibly build business processes. Orchestrator is suitable for all employees, as it involves no-code development, where processes are organized in the form of blocks. It is also able to review robot action logs, launch robots according to a schedule, in turn, or on a trigger, generate reports, and send notifications about failures and other emergency situations.

RPA_Agent is a link between the control application (*Orchestrator* or similar) and the robot.

On the RPA platform, to differentiate rights to perform various actions, users are divided into groups. There are three default roles: administrator, business analyst, and developer.

The agent contains data storage and provides access to it in a certain sequence. It also contains schedule templates for setting up software bot launches.

The list of processes contains conditions, auxiliary and main tasks, confirmations, and other elements. A special block ensures sending http requests to obtain data from the information base (archive). Monitoring occurs by performing an audit, which allows recording all log actions in the Orchestrator.

The rationale behind the metamodel is presented in Table 2:

Table 2. Metamodel's properties.

Property	Class	Dependent Class
Use	User	RPA Platform
Improve	Robot_Security	RPA Platform
Harms	RPA_Threats	RPA Platform
Mitigate	RPA_Control	RPA_Violation
Modifes	RPA_Control	RPA_Threats
Affects	RPA_Threats	External_systems: Database/external_resources/OS
LeadsTo	RPA_Violation	RPA_Threats
BaseOn	Robot_Security	RPA_Requirement
Consider	Robot_Security	Robot_Property

The metamodel uses the following element visibility parameters:

private (private, available only inside the class) - set by the "minus" symbol (-);

public (public, available to all) - specified by the "plus" (+) symbol.

4. Results and Discussion

The experimental evaluation of the proposed extension contained three main steps: model verification using an automatic tool for identifying inconsistencies between model elements, violations of the general rules of modeling UML, inconsistencies between types of attributes and operations, inadmissible dependencies between model elements, incorrect use of stereotypes and attributes of UML element; automatic code generation; and expert evaluation in order to ensure practical applicability.

4.1. UMLsec Model Verification

UMLsec model verification is the process of verifying models created using UMLsec, a formal metamodel for modeling system security. Its main purpose is to confirm the correctness and consistency of the model. It helps to ensure the quality and correctness of system modeling and prevents problems and errors in the further process of system development and operation.

Verification helps to identify errors, inconsistencies, and contradictions in the model, such as insufficient or redundant connections between model elements, incorrect arrangement of element. The process may include removing redundant elements, merging, or redistributing elements to improve the efficiency and simplicity of the model.

Several tools for automatic verification have been considered, the main purpose of which is to find only the main specific diagramming errors, without guaranteeing the absence of others. There are several widely used tools for the verification or validation of UML models:

1. UMLet is a free tool for creating and verifying UML diagrams. It provides the ability to check the syntax and semantics of diagrams, as well as automatic error detection [18].
2. Sparx Systems Enterprise Architect is a commercial modeling and development tool that supports the verification and model checking of UML models. It provides the ability to check the syntax, semantics, and integrity of models, as well as the possibility to automatically generate documentation and code from models [19].
3. Modelio is an integrated development and modeling environment that provides verification and model checking capabilities for UML models. It supports various verification techniques, such as static code analysis and model analysis [20].

Taking into account the scope functionality, the Modelio 5.3 tool was chosen as the verification tool. The automatic verification of UML models in Modelio 5.3 is performed for a number of purposes, including the following:

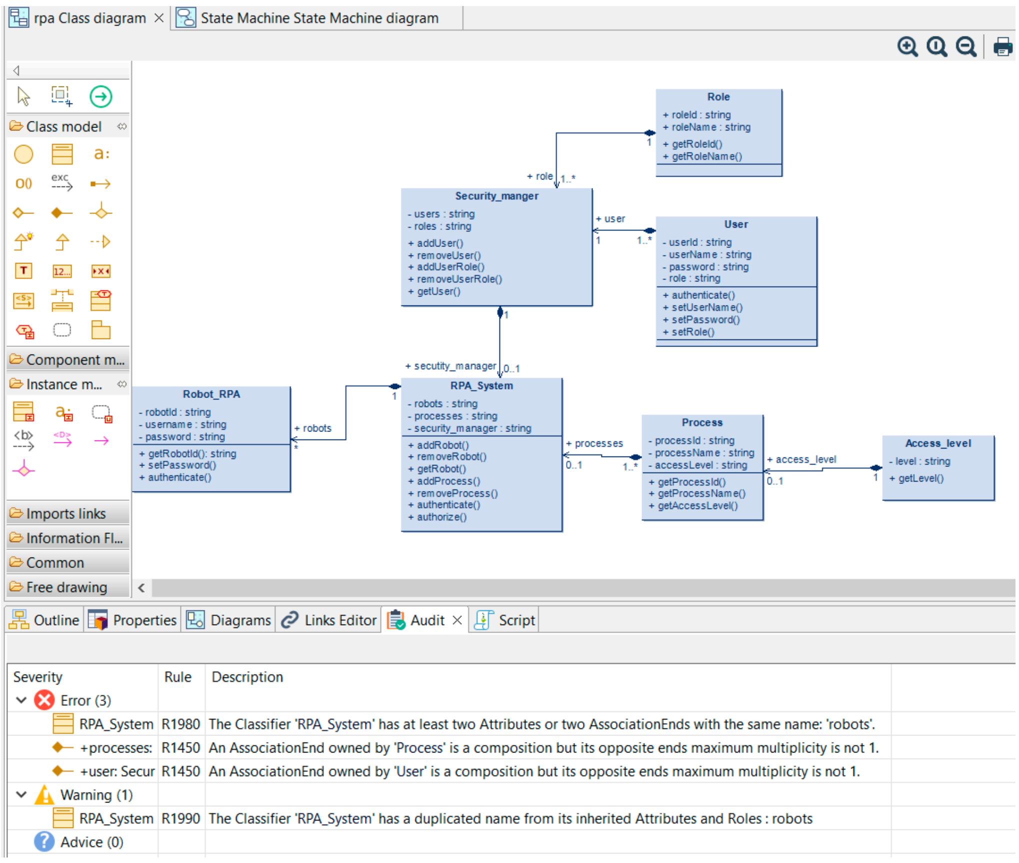
1. Checking the syntactic correctness of the model: Automatic auditing helps to detect and correct errors in the modeling, such as the incorrect use of UML elements, incorrect communication between elements, and incorrect inheritance.
2. Detection of model deficiencies: Automated auditing can help in finding flaws in the modeling, such as missing or incorrect extraction of entities, incompleteness, or inconsistency in the description of interactions between elements.
3. Support modeling standards compliance: Automated auditing can verify that models comply with a specific modeling standard or set of rules, such as a UML standard or organizational standards.
4. Increased modeling quality: Automatic auditing helps to detect and eliminate flaws and errors in modeling, which leads to improved model quality.

The inspection using the tool has shown three errors and one warning related to the *RPA_System* class and connections between classes. The types of errors are presented in Table 3.

Table 3. Identified error types at verification stage.

Error	Description
R1980	The *Attributes* and *AssociationEnds* of a *Classifier* represent variables that can be identified by their names, so they must have a unique name in that classifier.
R1450	A composition is a strong association that implies that a model element (the composite) is responsible for the life cycle of another model element (the part). Therefore, an instance of the part element cannot be linked to more than one instance of the composite element.
R1990	The *Attributes* and *AssociationEnds* of a *Classifier* must have a unique name in that *Classifier*. Furthermore, since when inheriting from a *Classifier*, some of its properties are inherited as well, the name must be unique in the classifier but also in all its parents.

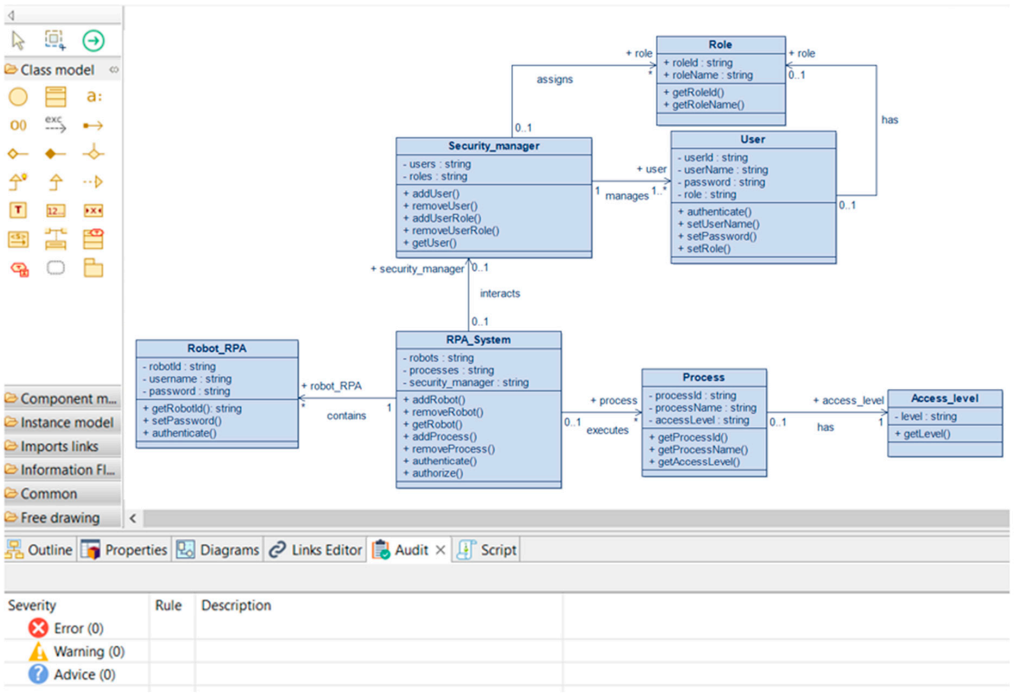
The diagram before its verification is shown in Figure 4.



Severity	Rule	Description
Error (3)		
RPA_System	R1980	The Classifier 'RPA_System' has at least two Attributes or two AssociationEnds with the same name: 'robots'.
+processes:	R1450	An AssociationEnd owned by 'Process' is a composition but its opposite ends maximum multiplicity is not 1.
+user: Secur	R1450	An AssociationEnd owned by 'User' is a composition but its opposite ends maximum multiplicity is not 1.
Warning (1)		
RPA_System	R1990	The Classifier 'RPA_System' has a duplicated name from its inherited Attributes and Roles : robots
Advice (0)		

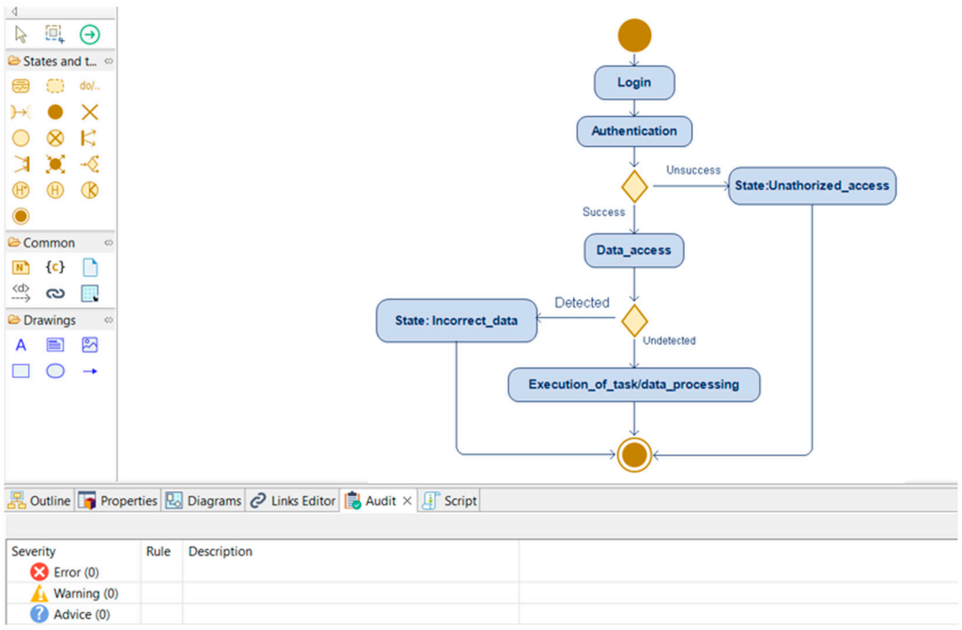
Figure 4. RPA class diagram before verification.

Further, the identified violations were eliminated. The results of the verification of the *Class* diagram and *Activity* diagram are shown in Figures 5 and 6, respectively.



Severity	Rule	Description
Error (0)		
Warning (0)		
Advice (0)		

Figure 5. RPA class diagram after verification.



Severity	Rule	Description
Error (0)		
Warning (0)		
Advice (0)		

Figure 6. RPA activity diagram after verification.

The automatic verification of UML models increased the efficiency and quality of the modeling process but does not guarantee the consistency and accuracy of the models. Therefore, additional verification is needed in order to ensure that model does not contain logical mistakes.

Additional expert evaluation of the model was performed. The questionnaire was given to seven experts that have practical experience in the field of information security (security architects (2), developers (2), and academic staff (3)). All interviewed experts have higher technical education with a specialization related to information technology. Four out of seven interviewed experts have certificates from information security centers. The average experience of the experts is 5–6 years. Three out of seven experts have practical or academic experience with RPA systems. As with all expert evaluation, it can be considered subjective. Still, answers have shown that five out of seven interviewees considered the UMLsec extension of RPA as appropriate. The evaluation methodology was the following: At first, the familiarization of experts with UMLsec extension of RPA with the provided extensions was performed. After that, experts had to provide evaluation according to the following criteria:

1. Functionality: The evaluation of how well the extensions meet the RPA needs and what additional functionality they provide.
2. Security: An assessment of the level of security provided by the extensions and their ability to protect RPA data and processes.
3. Integration: An assessment of how well the extensions integrate with existing RPA systems and other tools.
4. Extensibility: An assessment of whether the proposed extensions can be modified based on the need of a particular project.
5. Performance: The evaluation of the impact of extensions on RPA process performance and efficiency.
6. Support and documentation: The evaluation of the quality of support and documentation provided by extension developers.

A summary table with the results of the peer review is presented in Table 4. The number of points varied from 1 to 5, where 1 is the minimal, and 5 is the maximal value.

The results in Table 4 clearly demonstrate the acceptance of the extension by the majority of experts that took part in evaluation.

Table 4. Results of expert evaluation.

Evaluation Criteria	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Expert 6	Expert 7	Average Value
Functionality	4	5	5	4	5	3	3	**4.1**
Security	4	4	5	5	5	3	4	**4.2**
Integration	4	5	5	5	5	4	4	**4.5**
Expandability	5	5	5	5	5	5	4	**4.8**
Performance	5	4	4	4	4	3	3	**3.8**
Support and documentation	4	4	3	4	4	3	3	**3.5**

4.2. Code Generation

Verification has proved the correctness of the model proposed. But in order to ensure not just formal but also practical verification, it was decided to perform automatic code generation to demonstrate that the model can be used to generate valid code.

Generating code for UML models is necessary for automatic program source code generation on the basis of a graphical representation of the model [21]. This helps to speed up the development process and reduce the likelihood of errors since the code generated based on clear and unambiguous specifications defined in the UML model.

Umbrello UML Modeller [22] was chosen as a tool for automatic class code generation. However, existing tools require the creation of a “frame” or the main structure of the project, including classes, interfaces, connections between them, and methods. This allows for developing a project with an already defined architecture and basic components.

According to the class diagram (Figure 1), six classes were generated: *RPASystem*, *SecirutyManager*, *Role*, *User*, *Proccess*, and *Access_level*. Also, in order to demonstrate the relationships between them, two more classes were written: *Security_check* and *main,cs*. A sample of code fragment of the RPASystem class, where a list of robots and a list of processes is created, is presented on Figure 7. It demonstrates the method for adding a robot.

The *SecurityManager* class code fragment demonstrates creating a store for users and their passwords (Figure 8).

Figure 9 demonstrates a manually created *SecurityCheck* class, which has the ability to add personal RPA code, according to a given project. It is possible to implement additional logic.

Figure 10 shows the *mail* class with the ability to add logic for obtaining the process name, its ID, and the process access level.

The code generation experiment has demonstrated that a specialized tool can generate the valid code from the diagrams created with the help of the extension proposed. No generation issues were detected. Of course, the generated code may require further tuning and manual logic addition, but the total development process is speed, and quality is increased. A full description of the classes, as well as the code generated, is provided on Github [23].

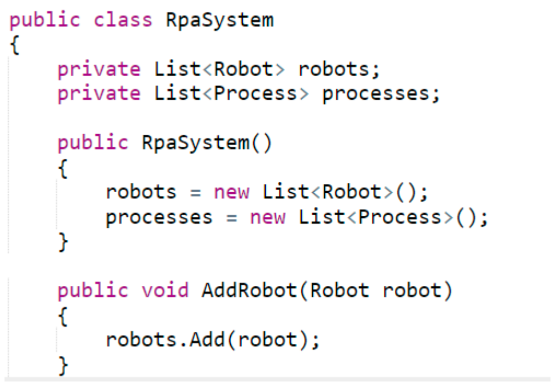
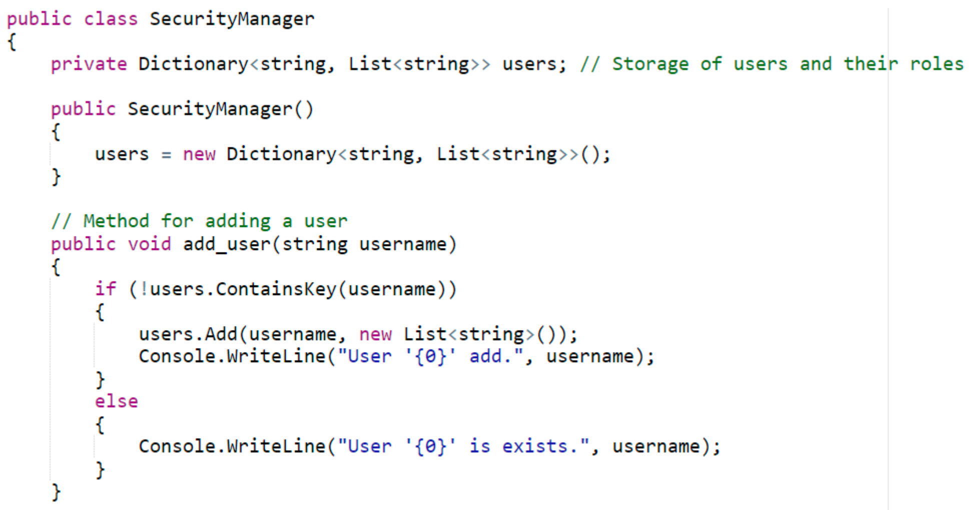
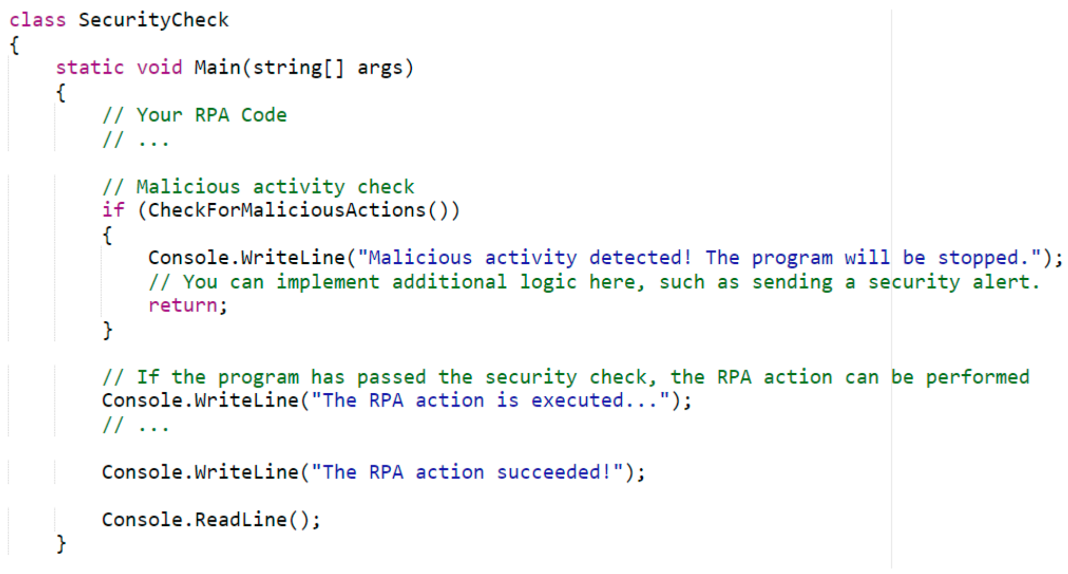

```
public class RpaSystem
{
    private List<Robot> robots;
    private List<Process> processes;

    public RpaSystem()
    {
        robots = new List<Robot>();
        processes = new List<Process>();
    }

    public void AddRobot(Robot robot)
    {
        robots.Add(robot);
    }
```

Figure 7. Class RPASystem.



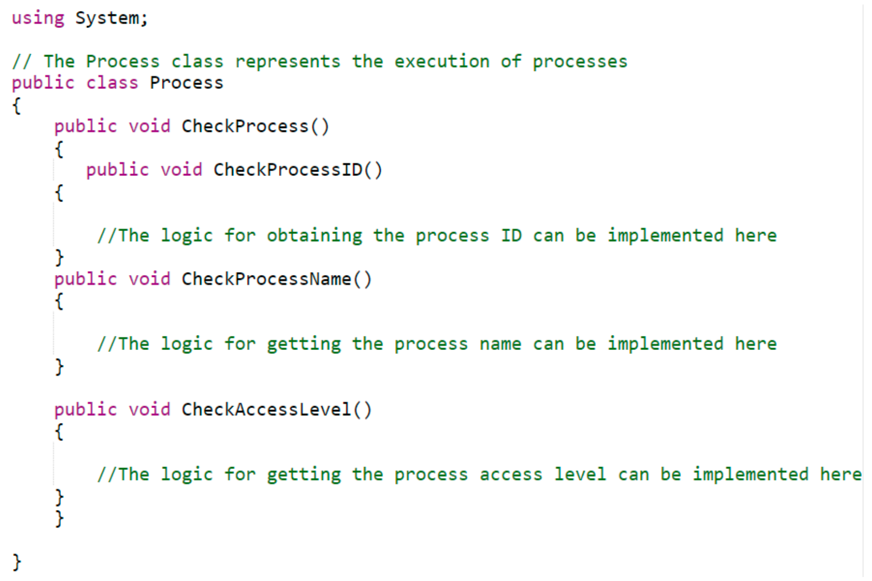
```
public class SecurityManager
{
    private Dictionary<string, List<string>> users; // Storage of users and their roles

    public SecurityManager()
    {
        users = new Dictionary<string, List<string>>();
    }

    // Method for adding a user
    public void add_user(string username)
    {
        if (!users.ContainsKey(username))
        {
            users.Add(username, new List<string>());
            Console.WriteLine("User '{0}' add.", username);
        }
        else
        {
            Console.WriteLine("User '{0}' is exists.", username);
        }
    }
```

Figure 8. Class SecurityManager.



```
class SecurityCheck
{
    static void Main(string[] args)
    {
        // Your RPA Code
        // ...

        // Malicious activity check
        if (CheckForMaliciousActions())
        {
            Console.WriteLine("Malicious activity detected! The program will be stopped.");
            // You can implement additional logic here, such as sending a security alert.
            return;
        }

        // If the program has passed the security check, the RPA action can be performed
        Console.WriteLine("The RPA action is executed...");
        // ...

        Console.WriteLine("The RPA action succeeded!");

        Console.ReadLine();
    }
```

Figure 9. Class SecurityCheck.



```
using System;

// The Process class represents the execution of processes
public class Process
{
    public void CheckProcess()
    {
       public void CheckProcessID()
    {

        //The logic for obtaining the process ID can be implemented here
    }
    public void CheckProcessName()
    {

        //The logic for getting the process name can be implemented here
    }

    public void CheckAccessLevel()
    {

        //The logic for getting the process access level can be implemented here
    }
    }

}
```

Figure 10. Class Main.cs.

5. Conclusions

The scientific and technical literature analysis has shown that the increasing popularity of RPA technology creates the demand for formalized RPA system modeling methods, which would include domain-specific security requirements on all life cycle stages. The most suitable modeling tool would be the UMLsec extension of the UML language, which allows for the creation of domain-specific extensions. Unfortunately, at the moment, there are no RPA-specific UMLsec extensions.

The UMLRPAsec extension, which includes RPA and RPA security related activity and class diagrams, specific stereotypes, and UML metamodel, was proposed, which is based on information from OntoSecRPA ontology.

The creation of a UML metamodel for RPA made it possible to standardize and improve the processes of the development and analysis of robots from a security point of view, which can contribute to a more efficient and high-quality use of RPA. Stereotypes allowed us to describe domain-specific security requirements, both on technical and organizational levels. The RPA class UMLsec diagram made it possible to visualize the main components and the relationships between them, while activity diagram can be useful to analyze how the system reacts to external events and how it changes its state.

The proposed extension UMLRPAsec verification included automatic formal verification with the help of the Modelio tool and automatic code generation from the sample extension class diagrams with Umbrello UML Modeller. Formal verification helped in identifying and correcting several non-critical mistakes, while code generation allowed the creation of class framework without any noticeable issues. Verification has demonstrated the proposed model correctness both on formal and practical levels. UMLRPAsec can be seen as an effective tool for automating and securing RPA solutions on several life cycle stages.

Author Contributions: Conceptualization, N.G.; methodology, N.G.; validation, A.K.; formal analysis, N.G.; investigation, A.K.; writing—original draft preparation, A.K.; writing—review and editing, N.G.; visualization, A.K.; supervision, N.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available and can be found here: Available online: https://github.com/oleferovich/UML-sec-class-diagram (accessed on 20 November 2023).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kurylets, A.; Goranin, N. Security Ontology OntoSecRPA for Robotic Process Automation Domain. *Appl. Sci.* **2023**, *13*, 5568. [CrossRef]
2. Jovanović, S.Z.; Ðurić, J.S.; Šibalija, T.V. Robotic process automation: Overview and opportunities. *Int. J. Adv. Qual.* **2018**, *46*.
3. Costa, S.A.S.; Mamede, H.S.; Silva, M.M. Robotic Process Automation (RPA) adoption: A systematic literature review. *Eng. Manag. Prod. Serv.* **2022**, *14*, 1–12. [CrossRef]
4. Kosi, F. Robotic Process Automation (RPA) and Security. Master's Thesis, Mercy College, New York, NY, USA, 2019.
5. Jürjens, J. UMLsec: Extending UML for secure systems development. In Proceedings of the 5th International Conference on the Unified Modeling Language, Dresden, Germany, 30 September–4 October 2002; Volume 2460, pp. 412–425.
6. Jürjens, J. *Secure Systems Development with UML*; Springer: Berlin/Heidelberg, Germany, 2005.
7. Enríquez, J.G.; Jiménez-Ramírez, A.; Domínguez-Mayo, F.J.; García-García, J.A. Robotic Process Automation: A Scientific and Industrial Systematic Mapping Study. *IEEE Access* **2020**, *8*, 39113–39129. [CrossRef]
8. Laufer, J.; Mann, Z.Á.; Metzger, A. Modelling Data Protection in Fog Computing Systems using UMLsec and SysML-Sec. In Proceedings of the 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Fukuoka, Japan, 10–15 October 2021.
9. Ozkaya, M.; Erata, F. A survey on the practical use of UML for different software architecture viewpoints. *Inf. Softw. Technol.* **2020**, *121*, 106275. [CrossRef]

10. Intuitive Visual Modeling for All UML Diagrams. Available online: https://www.altova.com/umodel#rev_engineer/ (accessed on 20 September 2023).
11. Robles-Ramirez, D.A.; Escamilla-Ambrosio, P.J.; Tryfonas, T. IoTsc: UML Extension for Internet of Things Systems Security Modelling. In Proceedings of the 2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), Cuernavaca, Mexico, 21–24 November 2017; pp. 151–156.
12. Farias, K.; de Oliveira Cavalcante, T.; Gonçales, L.J.; Bischoff, V. UML2Merge: A UML extension for model merging. *IET Softw.* **2019**, *13*, 575–586. [CrossRef]
13. Kothamasu, D.; Jiang, Z. *Web Application Development Using UML*; West Chester University: West Chester, PA, USA, 2017.
14. Hennicker, R.; Koch, N. Systematic Design of Web Applications with UML. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*; IGI Global: Hershey, PA, USA, 2001.
15. Bergmayr, A.; Troya Castilla, J.; Neubauer, P.; Wimmer, M.; Kappel, G. UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In Proceedings of the CloudMDE 2014: 2nd International Workshop on Model-Driven Engineering on and for the Cloud Co-Located with the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014), Valencia, Spain, 30 September 2014.
16. Kim, D.K. Development of Mobile Cloud Applications using UML. *Int. J. Electr. Comput. Eng.* **2018**, *8*, 596–604. [CrossRef]
17. Ordonez, C.; Al-Amin, S.T.; Bellatreche, L. An ER-Flow Diagram for Big Data. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020.
18. Free UML Tool for Fast UML Diagrams. Available online: https://www.umlet.com/ (accessed on 11 November 2023).
19. Sparx Systems Enterprise Architect. Available online: https://sparxsystems.com/products/ea/ (accessed on 11 November 2023).
20. Modelio Open Source—UML and BPMN Free Modeling Tool. Available online: https://www.modelio.org/index.htm/ (accessed on 15 November 2023).
21. Mukhtar, M.I.; Galadanci, B.S. Automatic code generation from UML diagrams: The state-of-the-art. *Sci. World J.* **2018**, *13*, 47–60.
22. Umbrello UML Modeller. Available online: https://sourceforge.net/projects/uml/ (accessed on 20 November 2023).
23. Github. Available online: https://github.com/oleferovich/UML-sec-class-diagram (accessed on 15 December 2023).