# Formal Analysis and Design for Engineering Security (FADES)

Riham Hassan Abdel-Moneim Mansour

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Computer Science and Applications

March 11, 2009
Blacksburg, Virginia

Keywords: Security Engineering, Software Engineering, Goal-Oriented Requirements Analysis, Formal Analysis and Design, Security, Formal methods

# Formal Analysis and Design for Engineering Security (FADES)

Riham Hassan

## ABSTRACT

Engineering secure software remains a significant challenge for today's software organizations as they struggle to understand the implications of security on their systems and develop systems that guarantee specified software security properties. The use of formal methods that are based on mathematical models has long been advocated in the development of secure systems, yet the promise of formal methods has not been realized. This is due to the additional discipline needed to formulate precisely the requirements and due complexities that often confront engineers. Further, the cost of development and the requisite learning curve of formal methods are quite high making them cost prohibitive to apply, especially for large software.

The transition from requirements to design has been one of the most difficult steps in software development. Moreover, effective methods for deriving design from requirements that guarantee retention of the intended security properties remain largely unrealized on a repeatable and consistent basis. If security requirements are formalized and transformed into design using formal methods, the potential for security vulnerabilities would be diminished through better clarity, completeness, and consistency. Therefore, a requirements specification must be systematically transformable to a formal representation, and through effective formal methods the design can be derived such that the security properties are preserved and conveyed.

This dissertation presents the FADES (Formal Analysis and Design for Engineering Security) approach that couples goal-oriented requirements specification with formal design specification to develop secure software in a constructive, provable and cost-effective way. To the best of our knowledge, FADES is the first security engineering approach that provides a systematic and automated bridge between semi-formal security requirements and formal design and implementation. FADES maintains the completeness and consistency of the security requirements specified with KAOS (Knowledge Acquisition for autOmated Specifications) when transformed to B formal specifications. Relaxing formality during requirements analysis enables security requirements to be better organized for producing more complete, consistent and clear requirements. The KAOS requirements model is then transformed to B, a popular formal representation used to derive and refine software systems. Security design specifications and implementation are produced using the B formal method which preserves the requisite security requirement properties.

FADES treats security-specific elements in a systematic and constructive way while considering security early in the development lifecycle. Moreover, employing FADES provides better confidence for security evaluators in the evaluation of trusted software. A side effect of employing formal methods in development is the availability of sufficient traceability information at the various phases of development and maintenance allowing for more accurate impact analysis of security changes.

FADES has been examined empirically both by security engineering experts and practitioners. Results obtained from the controlled experiments compare FADES to other formal methods, and show that FADES preserves security properties while maintaining better consistency, quality, and completeness. This is accomplished at a lower cost and with better results. These results have been evaluated by academic and industry experts working in the area of security and formal methods.

## Dedication

To the people I love most--

*My parents and Hatim, my dear husband*

&

*Omar and Ali, my lovely sons*

## Acknowledgements

First and foremost, I am grateful and thankful to Allah (God) who gave me the guidance and strength to do my doctoral work. Allah is the first to be praised and thanked for everything at all times. I thank Him for giving me patience and helping me through all the tough times of the Ph.D journey. I am always grateful to my beloved parents, brothers and sister who helped me through my life milestones. I am especially thankful to my husband whom without his concrete and moral support, I could have never succeeded in my Ph.D work. I am also grateful to my two sons Omar and Ali who bore a lot with me in the Ph.D journey and I hope that my Ph.D brings them a better mother. The Prophet Mohamed (Peace be upon Him) said: "He who doesn't thank people doesn't thank Allah." In the coming paragraphs, I express my gratitude to the people who helped me complete this work.

I have been fortunate to work with Dr. Shawn Bohner, who is inspirational as a teacher, as a mentor and as a human being. His impact on my way of thinking, my research process and my life experience is clear to me at the end of my Ph.D journey. His respect and appreciation to my ideas and my work were a strong motivation for me to proceed in the Ph.D. I will remain grateful for his guidance and feeling privileged to work for such a wonderful advisor.

I gratefully thank Dr.Sherif El-Kassas, my co-advisor for his sincere support, his guidance based on deep knowledge, and patience through all the years that I have worked with him.

I am grateful to my committee members: for Dr.Mohamed Eltoweissy, for his technical expertise and valuable guidance and comments, for Dr.Scott McCrickard for his encouragement and research guidance, and for Dr. Denis Gracanin for his expertise and insightful courses.

Thanks to Dr. Van Axel Lamsweerde who granted us a free license of the Objectiver software that has facilitated my work in the elicitation and modeling of security requirements using the KAOS framework. I am also thankful to the volunteer participants who participated in my empirical work to validate the work. These thanks go to both the IBM practitioners group, academic and industry experts.

I am so grateful to the VT-MENA professors who helped me achieve my dream in pursuing my Ph.D work in a respectable university like VT. I am especially grateful to Dr. Sedki Riad, the U.S. director of the VT-MENA program and Dr.Yasser Hanafy, the director of the program in Egypt. Without their efforts in making the VT-MENA program a reality, I would have never been able to pursue my Ph.D degree as I could not stay away from Egypt for the duration of the Ph.D. I am also grateful to all the Egyptian professors working in the program.

I would also like to thank all my friends, all over the world who have supported me through encouragement and prayers. I would like to mention special group of friends who went the extra mile and really sacrificed time and effort for me, Aida Zakaria, Randa Abdel-Meguid, my VT-MENA collegues and all my IBM friends, and my friends in the Software Engineering lab in VT.

Yet again, all the thanks and gratitude goes to Allah only (El Hamdulelah).

# Table of Content

## Table of Figures

## Table of Tables

# Introduction

Security is a growing concern as the software community increasingly develops larger and more complex systems. These systems support ever more distributed and integrated capabilities in the public and private sectors. As society increasingly depends on software; the size and complexity of software systems continues to dynamically grow and evolve with ever-richer semantics making them more difficult to structure and understand. Trusted system requirements compound the problem by adding security and privacy dimensions to the mix, and most software efforts become untenable as software organizations attempt to bolt on security mechanisms. One of the major sources of security vulnerabilities has been poor-quality software [61]. Security aspects are usually applied to products late in the development cycle leaving systems vulnerable to attacks. This not only results in ineffective security capabilities as seen in the relentless barrage of cyber attacks, but also the integrity of software systems is placed at risk as the engineering principles used to develop software systems are subverted by the subsequent burden of security requirements [1].

After three decades of adding security capabilities to software systems, two patterns arise. First is a heavy reliance on the latest security gadgets and post-development security evaluations [Common Criteria (CC) Security Evaluations] to provide confidence in trusted systems. The second is an unbalanced dependence on legal remedy for responding to security responsibility claims – citing "the corporation provided security measures that are inline with current practice." This will not be much good when a national disaster is caused by system vulnerabilities that could have been averted by engineering for security.

Engineering security requirements is a message that has recently received more attention from the research community [1] due to the losses caused by poorly secured software products that result from considering security as an afterthought of software development. According to Van Lamsweerde, there are three reasons for considering security requirements after the fact. First, early phases of software development raise the priority of analyzing and elaborating functional requirements over non-functional requirements such as security in order to obtain a functioning product in a short amount of time. The second reason is the lack of a constructive and effective mechanism for elaborating security requirements in a complete, consistent, and clear manner. The third reason is the lack of a precise and well-defined approach to produce design and implementation of security requirements that accurately achieve these requirements while ensuring proper handling of all requirements and allowing for requirements traceability at the different phases of development. The proposed FADES approach to engineer security requirements addresses these problems that hinder the consideration of security requirements at the various stages of software development.

In Figure 1a, Wing depicts a layered approach to build secure systems [9]. The cryptographic layer provides primitives for encryption, decryption, digital signatures, etc. The protocols layer provides means for securing communication including authentication and exchange protocols. The systems and languages layer provides security services built in general-purpose programming languages such as C or Java. The applications layer include applications like online shopping, banking, etc. to guarantee some level of privacy and protection to users' data [9].

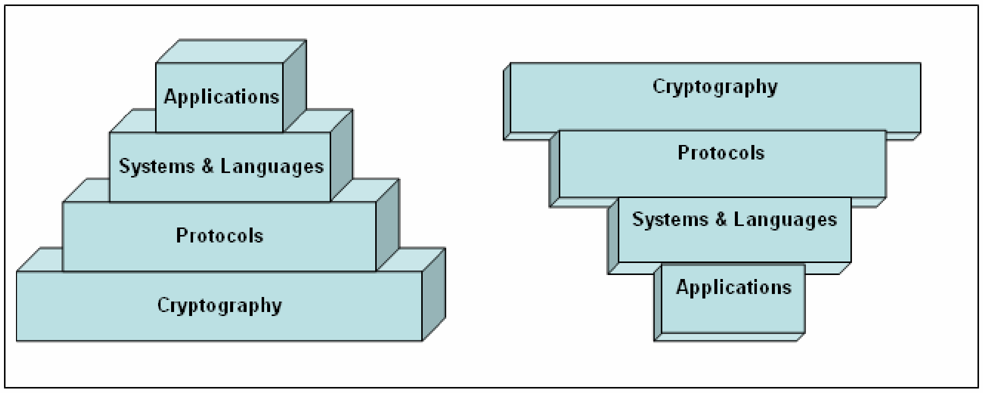


**Figure 1a: System Layers** **Figure 1b: Security Guarantees**

Figure 1b illustrates an ironic observation of the inverse proportionality between the strength of what we can guarantee at each layer in the security-layered approach to the size of the layer (Figure 1b). Wing suggests that there are significant results in cryptography, which can accurately tell us what we can guarantee and what we cannot. At the protocol level, formal methods have proven successful in providing guarantees for authentication protocols while at the systems and languages layer, commercial technology like Active X and Java provide different levels of security, but are still open to public attacks such as denial of service and spoofing. However, at the application layer in terms of security guarantees, there is not much to provide a reasonable level of security [9].

FADES described in this dissertation excels in applications layer to fulfill the essential need for a formal security requirements elaboration method encapsulated in an uncomplicated interface and transformed to a formal language to rigorously derive design specifications that maintain properties of the requirements model.

Security requirements engineering entails developing methods and tools which support the construction of complete, consistent, and clear specifications describing what a software system under development is supposed to do [63]. Candidate solutions including semi-formal modeling notations centered on object-oriented analysis (OOA) techniques like UML and formal specification techniques have been proposed to address the requirements problem.

The graphical notations provided with the semi-formal approaches are easy to use and communicate [33]. Further, the different kinds of diagrams may provide complementary views of the same system; such views can be related to each other through inter-view consistency rules [96]. However, semi-formal approaches to security engineering have limitations in that they can only cope with functional aspects; they generally support highly limited forms of specification and analysis; the semi-formal notation is most often fairly fuzzy since the same model may often be interpreted in different ways by different people [33].

Formal methods are complete and precisely defined, but need mathematical skills for effective use. Therefore most security specification tasks are still carried out with the support of informal specification methods, though this practice may lead to dangerously ambiguous, inconsistent, or incomplete specifications resulting in poorly secure software systems. Despite their advantages in solving security problems, formal methods are in fact still perceived as too cumbersome and complicated to be generally applied, and are relegated to the most critical sections of software development and software systems. Being a signification part of the critical aspects in software, security concerns are suitable candidates for development using formal methods. Nevertheless, a good chance for wider acceptance and use is given, if sufficient approach and tool support encapsulating and hiding the formal part become available.

Employing formal methods in real systems is steadily growing from year to year [97]. Despite of the good news, traditional semi-formal (unlike the goal-oriented KAOS approach) modeling and formal specification techniques suffer from serious shortcomings that explain why they are not fully adequate for the critical phase of requirements elaboration and analysis. These shortcomings are outlined as follows [1]:

- *Limited scope*: The vast majority of techniques focus on the modeling and specification of the software alone. They lack support for reasoning about the composite system made of the software and its environment.
- *Lack of rationale capture*: formal notations do not address the problem of understanding requirements in terms of their rationale with respect to some higher-level concerns in the application domain.
- *Poor guidance*: Constructive methods for building correct models/specifications for complex systems in a systematic, incremental way are by large non-existent. The problem is not merely one of translating natural language statements into some semi-formal model and/or formal specification. Requirements engineering in general requires complex requirements to be elicited, elaborated, structured, interrelated and negotiated.
- *Lack of support for exploration of alternatives:* Requirements engineering is much concerned with the exploration of alternative system proposals. Different assignment of responsibilities among software/environment components yields different software-environment boundaries and interactions. Traditional modeling and specification techniques do not allow such alternatives to be represented, explored, and compared for selection.

The above limitations have been overcome in the FADES approach through the employment of goal-orientation during requirements analysis [1, 3, 98] as described further in chapter 2. There are a number of goal-oriented approaches to requirements engineering. KAOS and i* represent the state-of-the-art as mature approaches in the goal-oriented requirements engineering paradigm [100]. KAOS, the more well-established of the two, uses first-order temporal logic as its formal infrastructure with good tool support

[98]. Further, KAOS is unique in its conceptual ontology: lower level descriptions of the system-to-be are progressively derived from system-level and organizational objectives using a framework that is essentially a taxonomy of concepts instantiated for a particular domain [101].

The KAOS security extension employed in FADES to model and analyze security requirements meet a set of meta-requirements that make such model securely reliable as follows [1]:

- *Early deployment*: In view of the criticality of security requirements, the technique is applicable as early as possible in the requirement engineering process, that is, to declarative assertions as they arise from stakeholder interviews and documents (as opposed to, e.g., later state machine models).
- *Incrementality*: This technique supports the intertwining of model building and analysis and therefore allow for reasoning about partial models.
- *Reasoning about alternatives*: This technique makes it possible to represent and assess alternative options so that a "best" route to security can be selected.
- *High assurance*: This technique allows for formal analysis when and where needed so that compelling evidence of security assurance can be provided.
- *Security-by-construction*: To avoid the endless cycle of defect fixes generating new defects, the requirements engineering process is guided so that a satisfactory level of security is guaranteed by construction.
- *Separation of concerns*: This technique keeps security requirements separate from other types of requirements so as to allow for interaction analysis.

FADES is the first to extend the goal-oriented KAOS framework to formal design and implementation, which brings the benefits of the goal-oriented paradigm to the software security domain. The employment of goal-orientation prior to stepping into formal design paves the road for formal design through performing thorough reasoning about security requirements and organizing them into a well-structured requirements model. Van Lamsweerde argues that goals offer the right kind of abstraction to address the inadequacies of formal and semi-formal methods for requirements engineering (especially for high assurance systems). These systems require compelling evidence that they deliver their services in a manner that satisfies safety, security, fault-tolerance and survivability requirements.

Goal-oriented methods are adequate for requirements engineering that is concerned with the elicitation of goals to be achieved by the software-to-be (WHY issues), the operationalization of such goals into specifications of services and constraints (WHAT issues), and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software available or to be developed (WHO issues) [94]. Positive/negative interactions with the other system goals can be captured in goal models and managed appropriately [3]; exceptional conditions in the environment that might hinder critical goals from being achieved can be identified and resolved to produce more robust requirements [3]; the goals can be specified precisely and refined incrementally into operational software specifications that provably assure the higher-level goals [37, 38, 41]. Requirements implement goals much the same way as programs implement design specifications [1].

Most research efforts in the field of security requirements engineering have been devoted to the requirements specification facet of requirements engineering [42]. A large number of languages have been proposed for requirements specifications, some of which are popular formal languages like Z, VDM, or LARCH. However, these languages are not well suited for capturing requirements models because they are too restricted in scope; they address only the "what" questions [41]. Typically, the data and operations of the system envisioned are specified through first-order assertions like conditional equations or pre/post-conditions and invariants. Another limitation is that such languages have no built-in constructs for making a clear separation between domain descriptions and actual requirements [42]. Van Lamsweerde indicated that recent attempts to design semi-formal languages like KAOS support a wider range of requirements with the ability to address the "why", "who", and "when" questions in addition to the normal "what" questions [42].

FADES addresses the limitations of the security requirements specification languages through employing KAOS to elicit security requirements that uses first order temporal logic to formally capture pre, post conditions and domain invariants. Moreover, KAOS models security aspects as goals resulting in a security goal graph in which the bottom level goals are either security requirements assigned to agents in the software-to-be (the software whose requirements are being modeled) or assumptions to be fulfilled by the interacting environment. This differentiation between requirements and assumptions clearly separates domain descriptions from actual requirements. Further, the bottom level goals of the graph are goals to be assigned to agents allowing the requirements model to extend beyond answering what questions to answer how questions and who is responsible for achieving these goals.

Being a crucial aspect in system development, security must be thoroughly ensured during all development phases. While still mostly relegated to the implementation and testing phases, it should be enforced at earlier stages too, i.e. in the requirements elaboration phase since early detection of possible security vulnerabilities is a key factor towards developing secure systems that are cost effective. Again, FADES is a requirements-driven approach that considers security very early in development while security vulnerabilities are analyzed and resolved. Moreover, the rest of the development phases are guided requirements model using the B formal method refinement mechanism that obtains its initial B model from automatically transforming the KAOS model to B. The employment of B as a design elaboration method to fully develop security-specific elements of software allows for the automatic verification of security implementation from more abstract properties represented in the requirements model. Further, the derivation of acceptance test cases from the requirements model provides means to ensure compliance between the derived implementation and the initial set of security requirements.

Automatic code generation from requirements facilitates the production of large systems with high-quality in a cost-effective manner. Therefore, it has been one of the objectives of software engineering almost since the advent of high-level programming languages, and calls for a "requirements-based programming" capability has become deafening [5]. Several tools and products exist in the marketplace for automatic code generation from a given model. However, they typically generate code, portions of which are never executed, or portions of which cannot be justified from either the requirements or the model [5]. Moreover, existing tools do not and cannot overcome the fundamental inadequacy of all currently available automated development approaches, which is that they include no means to establish a provable equivalence between the requirements stated at the outset and either the model or the code they generate. FADES solves this problem by building a mature and formally analyzed security requirements model using KAOS and transforms this requirements model into B to fully develop security aspects. Discharging B proof obligations provides means to establish a provable equivalence between the security requirements model and the more specified models produced for design and implementation.

While security-in-the-large encompasses development, deployment, and administration of trusted systems in their operational environments, this research focuses largely on the development process. We conjecture that *purposefully engineering security principles into trusted systems during development reduces the risks of manifesting vulnerabilities during deployment, administration, and operation in the trusted environment*. More concisely, while well understood security principles exist to guide organizations and security best practices exist for specific areas of development, there is no cohesive framework of security engineering principles that integrates development activities, artifacts, and practices, with relevant security principles. Lacking an integrated software security engineering process that incorporates security principles from the onset, we continue to employ informal solutions that render many subsequent systems vulnerable to attacks.

From the perspective of building a system devoid of security vulnerabilities, formal methods have been hoisted up as a reasonable solution. However, formal methods come with some challenges for large, complex systems. First, the application of formal methods entails substantial requirements formulation in a precise provable and correct representation, unambiguous, and consistent. Even for moderate systems, this goal is not readily achieved from a cost and time perspective. Secondly, once formal requirements set exists, translating them into a design that preserves the requisite security properties can be arduous and error-prone and arduous. Thirdly, the availability of engineers with the requisite experience to render a system definition and design specification formally is relatively low and they are often more expensive to employ.

Combining the increasing need for secure systems and the challenges of formal methods-based solutions, we have a dilemma. Do we bite the bullet and apply formal methods to engineering secure systems or do we continue down the informal path and live with the resulting vulnerabilities? Our research leads us to believe that there is a rational middle ground. First, assuring that all vulnerabilities are covered is unrealistic both technically and economically unless we are willing to accept systems of low size and complexity. Secondly, requirements engineering with formal methods is precise and provides a good platform for better completeness, but it still lacks the guarantees that completeness is achieved. Therefore, the argument for completeness turns to a tradeoff between approaches for time taken for getting to an acceptable level of completeness before proceeding on to design.

The requirements-driven goal-directed FADES approach proposed in this dissertation as a step towards the development of highly secure software is less precise and formal than starting from a totally formal approach, but it has three key mechanisms that render near-formal results. First, it employs KAOS (Knowledge Acquisition for autOmated Specifications) [1, 3], which is a proven goal-directed framework to elaborate security requirements. KAOS is natural for identifying and reasoning about the requisite requirements – making it an effective mechanism for facilitating completeness with additional formality in areas where it is essential. Second, the obstacle analysis mechanism of KAOS provides a good capability to reduce the presence of vulnerabilities by concentrating on the most appropriate alternatives for avoiding or eliminating the obstacles. Third, the test-case generation from the requirements model assures better alignment with the requirements and confidence in the specification from a verifiability perspective.

FADES might have a broader impact in two dimensions. The first is that it could enhance the infrastructure for research as some of the case studies used to demonstrate and verify the approach are industry-oriented. This collaboration between the proposed work and industry broadens its impact and extends its applicability and practicality. The second dimension in which FADES has a broader impact is that it widens dissemination to enhance scientific and technological understanding. This is achieved through the results obtained from verifying the approach through a controlled experiment, which are beneficial to software engineers in general and to security engineers in specific who lack a constructive and systematic approach to capture and develop security concerns in software products.

## Document Organization

Figure 2 illustrates the tasks carried out to accomplish this research work as presented in this dissertation. From now on in the document, the word "we" is used to refer to me and both my advisors (Dr. Shawn Bohner and Dr. Sherif El-Kassas) who graciously collaborated with their guidance, discussions and advice to get this research work done successfully. The document uses the term "software-to-be" to refer to the software system whose requirements are being modeled.

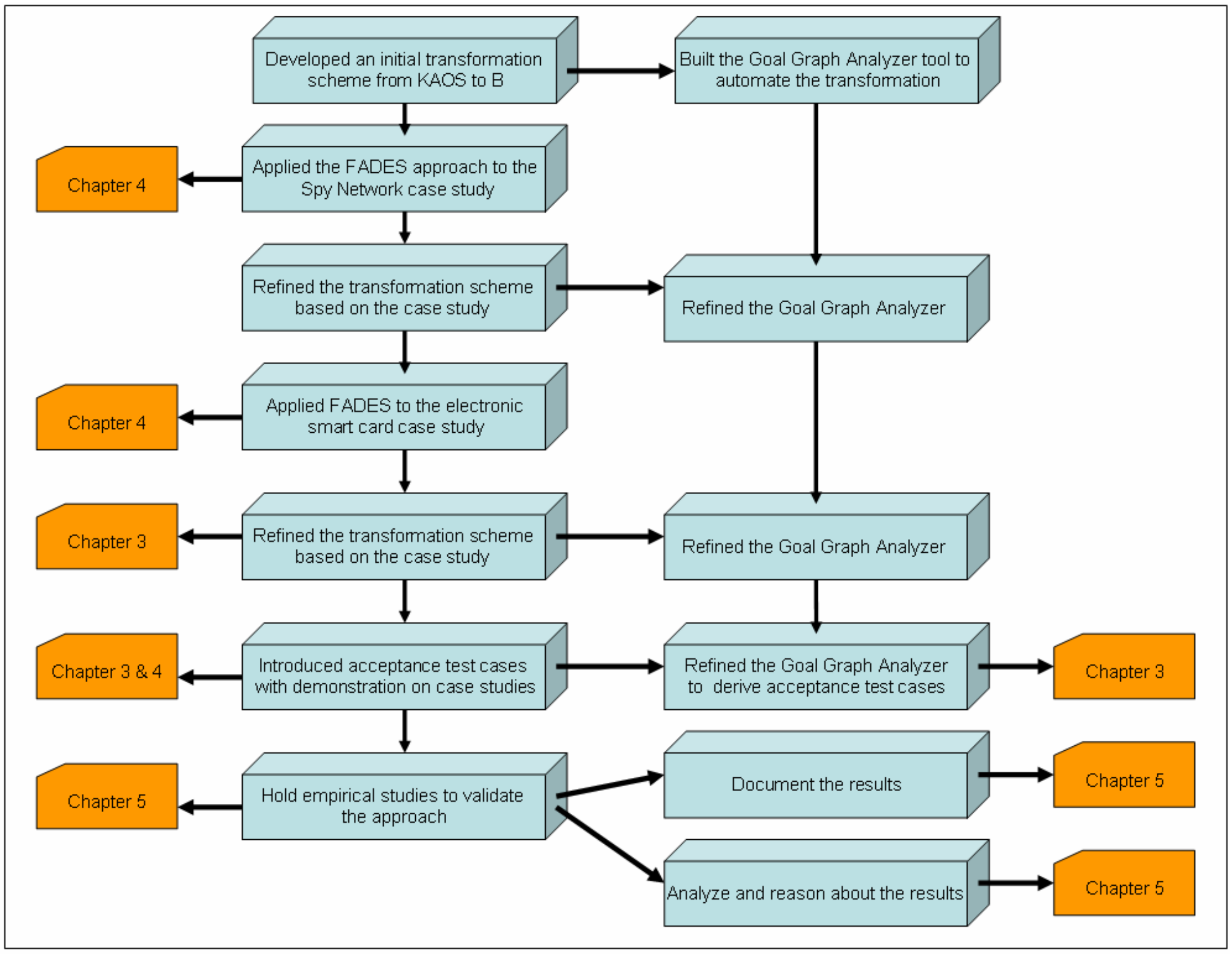

Developed an initial transformation scheme from KAOS to B
Built the Goal Graph Analyzer tool to automate the transformation
Chapter 4
Applied the FADES approach to the Spy Network case study
Refined the transformation scheme based on the case study
Refined the Goal Graph Analyzer
Chapter 4
Applied FADES to the electronic smart card case study
Chapter 3
Refined the transformation scheme based on the case study
Refined the Goal Graph Analyzer
Chapter 3 & 4
Introduced acceptance test cases with demonstration on case studies
Refined the Goal Graph Analyzer to derive acceptance test cases
Chapter 3
Chapter 5
Hold empirical studies to validate the approach
Document the results
Chapter 5
Analyze and reason about the results
Chapter 5

**Figure 2: Tasks for Defense**

Post-proposal tasks started by developing a mature transformation scheme from KAOS to B in which the research efforts both in the direction of model driven development [84, 85, 86] and the transformations done from KAOS to other formal languages like VDM++ [4] or from UML to B [25. 26, 27] were surveyed. An initial transformation scheme was developed and refined later when FADES was applied to cases studies. The automation of the transformation step has come along with the development of the transformation rules leading to the construction of the Goal Graph Analyzer tool that automatically parses the requirements model and constructs an equivalent model in the B language for further refinement to reach the ultimate goal of deriving design and implementation.

The next step was to demonstrate the approach on some relevant case studies. These were carefully selected with the following criteria:

- *Reasonable size*: large enough to be convincing, yet small enough to be manageable within a reasonable Ph.D research timeline.
- *Common security requirements*: representative of software categories with a common set of security requirements. For example, the Spy Network case study belongs to the category of communication-based systems that share a common set of security requirements. This means that the applying FADES to the spy network system shows feasibility of application to other software systems in the same category of communication-based systems.
- *Industry-related*: coming from industrial projects or research projects that serve industrial purposes. The rational behind this criterion is that our best case selection was to apply FADES to real projects in an enterprise to analyze its strengths and limitations, but money and time for doing that was not available and case studies have been the alternative. Selecting case studies with industrial background bring this research work close to the ultimate goal of studying FADES in true industrial environment.
- *Availability of sufficient information about security requirements*: most case studies available in publications might not be security-oriented or might not give sufficient details about security requirements. Therefore, an extensive search was required to find case studies that could be directly used with sufficient information.

Based on these criteria, two case studies were selected namely the spy network and the electronic smart card systems. The spy network system has been derived from two commercial software systems as indicated in Chapter 4 [40]. The spy network system has a close match to the above selection criteria with an extra feature of being already modeled in KAOS. The electronic smart card case study is derived from an industry-oriented research project for electronic smart cards. The security requirements of the electronic smart card system has been specified, designed and implemented in Z.

The availability of a case study like the electronic smart card system that focuses on employing formal methods to build secure systems has shown an opportunity for comparing FADES to other formal methods when both are applied to the same case study. In the proposal of this work, it was planned to compare FADES to the Genetic Software Engineering (GSE) method that was shown later to be unsuitable for comparison with FADES. The GSE method addresses functional requirements only with no constructs provided with the method to deal with non-functional requirements. This makes the GSE method inappropriate to apply for analyzing security requirements that are typically classified as non-functional requirements. Therefore, the application of the Z formal method to the security requirements of the electronic smart card case study has been more tempting for comparison with FADES. Both FADES and Z employ formal languages to design and implementation of security requirements, but FADES employs a different method, which is KAOS for specifying and elaborating security requirements. The comparison between FADES and Z shows how the integration of KAOS and the B formal method in FADES might result in better security as illustrated in Chapter 4.

After the application of FADES to case studies as illustrated in Chapter 4, the transformation rules from KAOS to B have needed some modifications that were also reflected to the Goal Graph Analyzer tool that automates the transformation.

Obtaining positive results from the case studies has led to the next step, which is proving equivalency of the transformation. To prove equivalency between the KAOS model and the initial B model resulting from the transformation, we have investigated the following two options;

- A formal mathematical proof.
- Acceptance test cases.

Consulting Nakagawa and the Japanese team who developed the formal specification generator for KAOS [4] that generates a VDM++ model from KAOS specifications about the feasibility of a formal mathematical proof of equivalency between KAOS and B, they reported that "we simply assumed the transformation rules are correct". Further, they reported their experience in trying to develop the mathematical proof as infeasible since the KAOS model is a requirements model involving undefined parameters making the proof very hard to develop within a reasonable timeframe. Moreover, the automation of the transformation reduces its error-proneness since automation rigidly applies the transformation rules. Further, the fact that both KAOS and B employ first-order predicate logic to express system constraints and conditions facilitates the transformation and reduces the probability of in-equivalency between the KAOS model and the transformed B model. While FADES bridges the gap between requirements and design using a similar transformation scheme to the one developed by the Japanese team, FADES provides an extra verification step through deriving acceptance test cases to assure transformation correctness.

Since the equivalency proof appears infeasible within a foreseeable timeline, an automated test case generation approach was considered. Test cases that are directly derived from the requirements model are more powerful in detecting compliance errors between implementation and requirements [28]. Therefore, the acceptance test case option has been more feasible to realize any errors in the derived implementation that might result from the transformation scheme. A set of acceptance test cases are automatically derived from the KAOS requirements model to verify that the derived design and implementation meet the security requirements objectives. Acceptance test cases show scenarios of operation sequences as specified in the requirements model to accomplish a certain requirement. The results of verifying the feasibility of the acceptance test cases idea have shown that the major two sources of problems encountered in implementation are inconsistencies either in the requirements model or in the design decisions made during the B refinement steps, but not in the transformation rules.

Finally, a validation mechanism for FADES was needed through one of the following two options:

- Apply the approach to projects in an industrial corporation.
- Apply and evaluate FADES in a controlled environment involving a small group of software engineering practitioners domain experts in formal methods and security areas.

The first option is more demonstrative and gives more meaningful results, but appears infeasible with the available resources for the timeline. The second option sounded more realistic, so we have designed two experiments, one to apply FADES versus other formal methods by practitioners and the other to evaluate the approach by academic and industry experts. The practitioners' experiment collected their feedback through allowing them to experience the approach and apply it to part of the security requirements of the electronic smart card system. The experts' experiment focused on domain experts to evaluate the effectiveness of the different features of FADES and report their expectations about the scalability of the results obtained in the practitioners' experiment. Chapter 5 describes the two experimental studies.

Our research work has obtained good feedback from the research community through our four conference publications indicated in the References section [46-49]. The results have been sent to the IEEE Transaction on Software Engineering for scientific archival.

## Problem Statement and Hypothesis

Security is often considered after the fact and in an ad-hoc manner resulting in insecure software products. Security requirements play a crucial role in designing secure software. With poor specification of security requirements, many security issues become vulnerabilities that are pushed to maintenance for resolution [2]. In practice, it has not been clear how to engineer security systematically throughout the software lifecycle [1].

Traditionally, security requirements have been engineered using one of two extremes namely informal ad-hoc practices or formal methods. The former is convenient to learn and apply but results in vulnerable software highly exposed to cyber attacks and the software systems integrity is places at risk. The later is more precise and has been hoisted up as a reasonable solution to engineer secure software. However, formal methods come with some challenges, especially for large and complex systems. First, the application of formal methods entails substantial requirements formulation in a precise, provable and correct representation, unambiguous and consistent. Even for moderate systems, this goal is not readily achieved from a cost and time perspective. Secondly, once formal requirements set exists, translating them into a design that preserves the requisite security properties can be arduous and error-prone. Thirdly, the availability of engineers with the requisite experience to render a system definition and design specification formally is relatively low and they are often more expensive to employ.

The main problem addressed by FADES is the application of appropriate rigor in elaborating and refining security requirements to produce systems that possess and guarantee the requisite security properties necessary for the production of secure software. There is a need for an appropriate approach that achieves the balance between preserving and guaranteeing security properties while allowing the flow of requirements engineering to proceed.

FADES provides a security requirements-driven software development approach intended for highly secured software in a cost-effective manner. The integration of semi-formal and formal methods in FADES balances the tradeoff between complexity of rigid formality and expressiveness of semi-formal approaches. Our hypothesis is that if the semi-formal requirements approach KAOS is integrated with a formal design specification method, B, deriving design specifications from security requirements should provably preserve the requisite security properties while maintaining consistency and completeness.

With some investigation, KAOS has been promising in that it could be extended with an extra step of formality in order to fully implement security requirements while preserving the requirements security properties [47]. Further, extending KAOS with more formality in a development framework like B allows for tracing requirements at the various steps of development; that is, during both the design and implementation.

# Chapter 1: State of the Art Survey

FADES is a security engineering approach employing formal methods to analyze, design and implement security-specific elements of software with the aim of producing highly secured software products. FADES can therefore be positioned in the intersection of three areas of software research namely software engineering, security and formal methods as illustrated in Figure 3. This chapter surveys research efforts of employing formal methods to produce highly secured software, elaborating and modeling security requirements, deriving design specifications and implementation from requirements using formal and semi-formal methods and checking consistency between requirements and design.

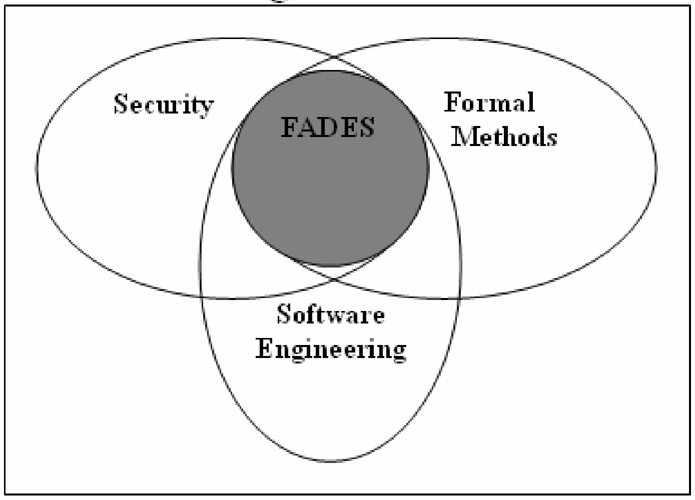


**Figure 3: FADES Position in Software Research**

Clarke, et. al. defined formal methods as “mathematically based languages, techniques and tools for specifying and verifying software systems” [8]. They have argued that using formal methods does not necessarily guarantee correctness, but they can significantly elevate our understanding of the system through detection of inconsistencies, ambiguities and incompleteness. Using formal methods for producing highly assured software has long been accepted and advised for secure systems [5]. Even contemporary security evaluation suggests formal specification and controlled transformation as evidenced in the Common Criteria Evaluation Assessment Levels (EAL) 5, 6 and 7 requiring formalism. Yet, formal methods are rarely exercised in the domain of security engineering.

The Common Criteria (CC) is an international standard for evaluation of software security. It provides a framework for security users to specify their security requirements while software vendors implement and make claims about the security attributes of their products that are evaluated by the concerned parties like test laboratories to verify the claims. The CC defines seven Evaluation Assurance Levels (EAL 1 to EAL 7) to measure the degree of compliance between the product and the claimed security functionality. Each EAL covers the complete development of a product with a given level of strictness with EAL 1 being the most basic and hence the cheapest to implement and evaluate and EAL 7 being the most rigorous and expensive [90]. The first four EAL levels of the CC do not require formal evidence for assuring the security functionality of the products while EAL levels 5, 6 and 7 have requisites of providing formal artifacts of development to assure the claimed security attributes. This means that products assured at EAL 5-7 provides more confidence to the users in the security claims of the product.

EAL1 is applicable where some confidence in correct operation is mandated while security threats are not considered serious [90]. Evaluation at EAL1 should show evidence of compliance between the target of evaluation functions and its documentation using, for example, testing. EAL2 requires the interference of the developers in delivering design and test results and is applicable where low to moderate level of security is required. EAL2 provides security assurance by analyzing the security functions using requirements specifications, guidance documentation and the high-level design of the target of evaluation [90]. EAL3 thoroughly investigate the target of evaluation without substantial re-engineering. EAL3 provides security assurance using requirements specifications, documentation and the high level design; however, the analysis is supported by testing of security functions and evidence of the developer testing. EAL4 is applicable to moderate and high level security and it provides assurance through analysis of security functions using complete requirements specifications, documentation, high-level and low-level design, and subset of the implementation to understand security behavior [90].

EAL5 is the first assurance level requiring formal evidence of development to assure high-level security of the target of evaluation. EAL5 requires complete specification, documentation, high-level and low-level designs and all of the implementation. Assurance is further gained through a formal model of the security policy and a semi-formal presentation of the requirements specifications and the high-level design and a semi-formal evidence of the compliance between the two. EAL6 gains assurance through a formal model of security, a semiformal presentation of the security requirements specifications, high-level and low-level designs and a semiformal evidence of the compliance between the specifications and the two designs. Further, a modular and layered design is required. EAL7 is applicable to extremely high-risk applications where the assets need maximum protection. EAL7 assures security through a formal model of the security policy, formal presentation of the security requirements specifications, high-level design, a semiformal presentation of the low-level design and formal and semiformal evidence of compliance between specifications and both the high-level and low-level designs. These definitions imply that the software products developed with FADES could be assured at EAL5, EAL6 and EAL7. These three evaluation assurance levels especially EAL6 require semiformal presentation of the requirements specifications, which are provided in FADES in the semiformal form of the KAOS goal graph. Further, EAL6 and EAL7 require formal presentation of design, which is provided in FADES using the B formal method and a semiformal evidence of compliance between specifications and design, which is provided in FADES using the acceptance test cases generated from the specifications model (KAOS goal graph). The employment of formal methods in FADES provides the required formal evidence of development for the assurance of the resulting target of evaluation at EAL5-7.

There are a number of success stories for applying formal methods to security-critical systems at the application layer; however, there are few if any formal methods-based approaches that handle security-specific elements. Stepney used the Z formal language to construct a security requirements specifications model that has been further refined to derive design and implementation specifications for a money exchange system using smart cards [6]. Clarke surveyed several applications of formal methods to security aspects [8]. Oxford University and IBM Hursley Laboratories collaborated in the 1980s on employing Z to formalize part of IBM's Customer Information Control System (CICS), an online transaction processing system with thousands of installations worldwide [8]. IBM reported measurements collected throughout development to indicate an overall improvement in product quality, a reduction in the number of errors discovered, earlier detection of errors, and an estimated 9% reduction in total development cost. Sabatier and Lartigue reported on industrial smart card application in which they designed the transaction mechanism to provide secure means for modifying data that is permanently stored in smart cards [10]. They demonstrated how the use of the B method increased confidence and provided mathematical proof that the design of the transaction mechanism fulfills the security requirements.

A number of approaches have been proposed in the literature for elaborating and modeling security requirements in a way comparable FADES. Misuse cases that complement UML are able to capture attacker features at requirements engineering time. Misuse cases are defined as "the inverse of UML use cases specifying functions that the system should not allow" [15, 16]. Misuse cases refer to scenarios that result in loss for the organization or some specific stakeholder. The concept of mis-actor is associated with the misuse cases. A mis-actor is defined as "the inverse of a UML actor, who is someone-intentionally or accidentally initiates misuse cases and whom the system should not support in doing so" [15]. Misuse cases can be normally related to normal use cases through "includes", "extends", "prevents", and "detects" relations. The relation of "includes" or "extends" from a misuse case to an ordinary use case indicates a misuse of one of the functions of the ordinary use cases. For example, a denial of service (DoS) attack needs not include illegal actions, just flooding the system with a heavy burden of publicly available registration requests. The "prevents" and "detects" have been introduced specifically for misuse cases, which relate ordinary uses cases to misuse cases outlining functions that prevent or detect misuse [15].

Firesmith in [17] has differentiated between misuse cases and security use cases. Firesmith thinks that misuse cases provide means for analyzing security threats but are inappropriate for the analysis and specification of security requirements. Instead, security use cases should define possible threat scenarios from which the application should be protected. Figure 4 shows the functionality of both misuse cases and security use cases. Security use cases could then be integrated with the rest of UML artifacts such as class and interaction diagrams in order to provide a software engineering approach that captures security requirements early on and integrates them with the rest of the system development phases through UML [17].

Unlike FADES, misuse cases and security requirements use cases handle security concerns only during the requirements and analysis phases while FADES covers the different stages of development for security requirements to produce a complete implementation. Misuse cases and security use cases inherit the simplicity and popularity as well as the semantic inconsistencies of UML. Further, they lack the rigor and requirements traceability features in FADES.

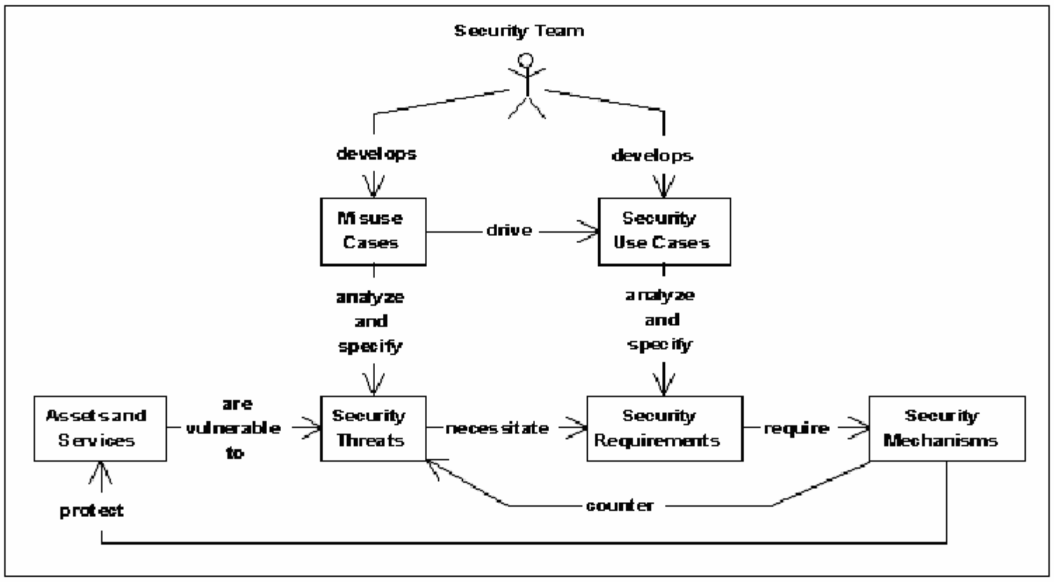


**Figure 4: Misuse cases vs. security use cases [17]**

Liu et. al. have extended the i* agent-oriented requirements modeling language to handle security and privacy concerns [18, 19]. The i* is an agent-oriented framework for modeling and redesigning intentional relationships among actors that are strategic to the software being modeled [87]. The i* framework is concerned with the early phase of requirements engineering with the notion of strategic actor being a central concept. Actors have properties like goals, beliefs, abilities, and commitments. The framework focuses on analyzing the strategic implications that each actor is concerned with in order to meet that actors' interests. This is achieved through modeling intentional relationships among actors, rather than input/output data flow, in which actors depend on each other to achieve goals, perform tasks, or employ resources. Modeling software from the agent perspectives has shown potential in extending the i* framework to model security aspects that originate from human concerns and intents; therefore, should be modeled through social concepts [18].

This i* security extension provides analysis techniques to deal with security requirements. The first technique is the attacker analysis that identifies potential system attacks. The attacker analysis theme is that "all actors are assumed guilty until proven innocent" [18]. Ordinary system actors (roles, positions or agents) are considered among potential attackers to the system or to other actors. The second analysis technique is the dependency vulnerability analysis that detects vulnerabilities in terms of organizational relationships among stakeholders whose dependency relationships bring vulnerabilities to actually attack the system through the manipulation of their malicious intents. Detailed analysis of vulnerability with the i* dependency modeling capability to trace the potential failure of each dependency to a dependent and to its dependents. Countermeasure analysis proposes proactive actions to resolve vulnerabilities and threats. Finally, access control analysis fills in the gap between security requirements and their realization in implementation. Access control analysis uses i* models to embed a proposed solution to system design. The i* role-based requirements analysis with i* facilitates the transition from requirements to design since it fits naturally to the role-based access control methodology software design.

The i* security extension is close in spirit to KAOS security extension employed in FADES. The main difference is that the i* security extension starts with agents involved in the system rather than goals being threatened as in KAOS. Further, the i* security extension identifies insider attackers only, that is, system stakeholders that were identified before in the primal model and might be suspect while KAOS identifies possible attacks regardless they can be performed by insider or outsider attackers. In the i* security extension, the malicious goals owned by attackers are not modeled explicitly and the methodology provides no formal techniques for building threat models. On the other hand, the KAOS security extension provides a formal procedure for generating attacks and countermeasure to such attacks, which makes KAOS more reliable especially in the security requirements context.

Both the KAOS security extension and the i* security extension provide constructs to effectively reason about security requirements. However, the KAOS security extension has been favored over the i* one to employ in FADES because the i* framework is not based on first-order predicate logic like KAOS making it difficult to transform to a formal language like B.

There are a number of approaches proposed in the domain of deriving design from requirements both formally and semi-formally. Some of these approaches extended the KAOS framework for further stages in development like architecture and design. All the approaches that extended KAOS exploited the key features of goal-orientation namely the ability to explore alternatives in specifications, responsibility assignment, goal formalization, adapting different levels of formality, and modeling the software and its environment. However, the approaches that extended KAOS focused on the functional and non-functional requirements of the system and none of them addressed the security aspects like FADES. This distinguishes FADES in being the first to employ goal-orientation for the design and implementation of security-specific aspects of software system.

Nekagawa et. al. have proposed a formal specification generator that transforms KAOS requirements specifications into VDM++ specifications to develop software systems [4]. Requirements are elaborated and analyzed using KAOS resulting in the construction of a requirements model, which is given as an input to the generator to produce the VDM++ specification. Missing parts in the requirements model are commented during the generation process to prompt developers to augment the VDM++ specification. The generated specification contains implicit operations consisting of pre and post-conditions, inputs, and outputs while the body of operations is left for developers to add to create an explicit specification. Test cases are developed to verify the formal specification using the VDM tools [4]. An overview of the VDM++ generator steps and artifacts is illustrated in Figure 5. Like FADES, the VDM++ generator formally derives design from requirements modeled with KAOS. Nevertheless, the VDM++ generator only derives high level design with no specific focus on security aspects.

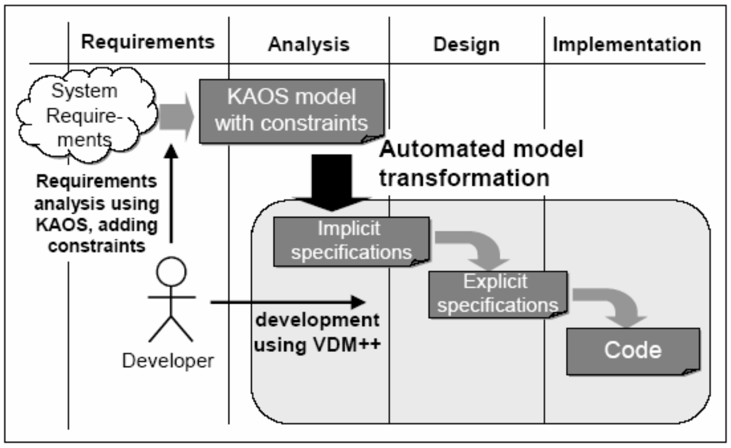


**Figure 5: Overview of Development Process**

Jiang, et. al., presented a case study of a real world industrial application that produced several versions of conceptual schema design for a biological database during its evolution [94]. The case study compares two different methods for designing a database. The case study authors started with an analysis of the original conceptual schema and its evolving design. They then revisited the design process using KAOS in order to construct a goal model of the problem domain. The case study has been used as a proof of concept for the authors' work in devising an extended database design methodology in which stakeholder goals drive the design process in a systematic way. This research direction has been motivated by the authors' belief in goal-oriented capabilities of making stakeholders' goals explicit, and exploring a space of design alternatives that lead to a set of data requirements specifications, each of which corresponds to a particular choice to fulfill the top-level goals

The comparison of the design choices for the biological database as originally made by its designers in successive versions and the design recommendations suggested by the goal analysis shows that the goal-driven approach results in a design spaces that:

- Includes original schema built through the evolution of the application.
- Suggests additional alternatives that lead to more comprehensive design.
- Supports systematic evaluation of design alternatives.
- Generates schemas with rich and explicit data semantics.

The authors' interpretation to the above results is that goal analysis allows the cognitive process that took place in the actual design of the database to be explicit when goals are declared as opposed to being implicit in the traditional method [94]. This led to schema design that better responds to the purpose of the application. The schema resulting from the applying goal analysis provides justification for the design choices and suggests additional alternatives that lead to more comprehensive design in terms of the coverage of the stakeholder goals. Further, an important property of the goal model is that all the alternatives exist to be examined, regardless whether they are selected or not. Goal analysis provides a systematic way for evaluating design alternatives through use of soft goals that are goals without clear-cut criteria for their satisfaction and usually used to model non-functional requirements of a software system [94]. Design alternatives represent different information needs that may have positive or negative contributions to the fulfillment of the soft goals. Moreover, goal-oriented database design provides direct trace from intentions to requirements to schemas. The knowledge captured during design can be used to attach explicit meaning to the elements in the schema and propagate to the data organized by the schema. The justification for the results has been demonstrated with examples from both the traditional schema and the goal-oriented one.

Brandozzi and Perry proposed a method to transform the requirements specification for a software system into an architectural specification [64]. This approach has chosen KOAS as a goal-oriented requirements engineering methodology to specify requirements and transforms such requirements specifications to APL (Architecture Prescription Language) in order to derive an architecture prescription from the KAOS requirements model. The authors justified their choice of KAOS by their belief that goal-oriented specifications, among all kinds of requirements specifications, are nearer to the way human thinks and are easy to understand by all stakeholders. Another reason is that goal-oriented specifications are particularly suitable to be transformed to formal languages like APL. The construction of requirements in the form of a directed-acyclic graph provides analytical capability to the requirements model and facilitates its transformation to a formal language. The approach tries to find a solution to the transition from requirements to architecture, which has been traditionally one of the most difficult aspects of software engineering [64]. Such transformation is difficult because it transforms the question of what we want the system to do into a basic framework for how to do it.

This approach takes as input goal oriented requirement specifications and returns as output an architecture prescription. An architecture prescription lays out the space for the system structure by restricting the architectural elements (processes, data, and connectors), their relationships (interactions) and constraints that can be used to implement the system [64]. The main advantages of an architecture prescription over a typical architecture description are that it can be expressed in the problem domain language and it is often less complete, and hence less constraining with respect to the remaining design of the system. An architectural prescription concentrates on the most important and critical aspects of the architecture and these constraints are most naturally expressed in terms of the problem space (or business domain, the domain of the problem). An architecture description, on the other hand is a complete description of the elements and how they interface with each other and tends to be defined in terms of the solution space rather than the problem space (or in terms of components such as GUIs, Middleware,

The rules for transforming KAOS constructs to APL constructs are as follows: each object in the requirements generally corresponds to a component in the architecture. More specifically, and agent object, and active object, corresponds to either a process or a connector. By definition, a process (thread, task) is an active component. What might not be immediately apparent is that also a connector can be an active component. An example of this type of connector is a software firewall. A software firewall is an active entity that checks whether the processes that want to interact satisfy some conditions or not, and allows or denies the interaction among them accordingly.

The events relevant to the architecture of the system are those either internal to the software system or those in the environment that have to be taken into account by the software system. The receiving of a message by a process is an example of internal event. The triggering of an interrupt by a sensor is an example of external event. An event is generally associated to a connector.

An entity, or passive object, corresponds to a data element, which has a state that can be modified by active objects. For example, the speed of a train is a variable (entity) that can be modified by a controller (agent). A relation corresponds to another type of data element that links two or more other objects and that can have additional attributes. An example of relation data is a data structure whose attributes are the type of train, its current speed and its maximum speed (additional attribute). A goal is a constraint on one or more of the components of a software system. Additional components may be derived to satisfy a non-functional goal. An example of a constraint deriving from a goal is that a component of the software system of an ATM has to check if the password typed by the user matches the password associated in the system to the ATM card inserted.

The transformation of KAOS to an architecture prescription is closely related to FADES in that both approaches model requirements using the KAOS framework with FADES being more focused on security requirements. However, FADES achieves more in terms of software implementation since requirements are produced in an implementable form while the other approach transforms requirements to architecture only. Further, FADES verifies that the derived implementation maintain the security properties specified in the requirements through the correctness-by-construction guarantees associated with the B formal method. In the other approach, requirements are transformed to APL, which is an architectural prescription language with no such rigid formality that guarantees the fulfillment of the requirements specified in the requirements model by the generated architecture.

Van Lamsweerde extended the KAOS framework to systematically derive architectural design from functional and non-functional requirements so that the compliance between architecture and requirements is guaranteed by construction [95]. Software specifications are first derived from the KAOS requirements model followed by deriving an abstract architectural draft from functional specifications. This draft is refined to meet domain-specific architectural constraints. The resulting architecture is then recursively refined to meet the non-functional goals modeled and analyzed during the requirements engineering process [95]. Van Lamsweerde provides means to bridge the gap between requirements and architecture using a rigorous architectural design process that relies on the use of precise descriptions of the software components and their interactions. Although this approach is still in its preliminary stages, Van Lamsweerde has specified some ideal meta-requirements on the derivation process in which the derivation should be:

- Systematic so that active guidance could be provided to architects.
- Incremental to allow for reasoning on partial models.
- Leading to (at best) provable or (at least) arguably "correct" and "good" architectures in order to demonstrate that the derived architecture indeed meets the functional requirements and achieves the non-functional ones.
- Highlighting different architectural views like a security view, a fault tolerance view, etc.

The KAOS framework is used to formulate requirements in terms of objects in the real world of the software-to-be, and in a vocabulary accessible to stakeholders. Required relations between objects are captured in order to model the environment in which these relations are monitored and controlled by the software. The next step after constructing the KAOS requirements model for the software is to derive software specifications that are formulated in terms of objects manipulated by the software, and in a vocabulary accessible to programmers while capturing required relations between input and output software objects. The derivation of software specifications from requirements follows the below rules:

1. All goals assigned to software agents are translated into the vocabulary of the software-to-be by introducing software input-output variables.
2. Relevant elements of the domain object model are mapped to their images in the software's object model.
3. Accuracy goals modeling non-functional requirements that mandate the mapping to be consistent are introduced, that is, the state of software variables and database elements must accurately reflect the state of the corresponding monitored/controlled objects they represent.
4. Assign input/output agents to the accuracy goals introduced in step 3- typically, sensors, actuators or other environment agents.

The derived software specifications are assumed to be non-conflicting as conflicts have been managed upstream in the requirements engineering process [3]. The software specifications are used to obtain a first architectural draft from data dependencies among the software agents assigned to functional requirements. These agents become architectural components that are statically linked through dataflow connectors. The procedure for deriving dataflow architecture from the software specifications is as follows:

1. For each functional goal, a component is defined to regroup a software agent responsible for achieving the goal with the various operations operationalizing the goal and performed by the agent. The agent's interface is defined by the sets of variables the agent monitors and controls.
2. For each pair of components C1 and C2, a dataflow connector is derived from C1 o C2 labeled with variable d if and only if d is among C1's controlled variables and C2's monitored variables.

The initial abstract architecture obtained with the above construction rules defines the refinement space in which different alternatives of component refinement exist. The refinement space is first globally constrained by architectural requirements and then different alternatives are explored to refine components and connectors. Van Lamsweerde proposed imposing "suitable" architectural styles in order to refine the dataflow architecture, that is, styles to be documented by applicability conditions such as domain properties and the soft goals they are addressing [95].

Once the abstract dataflow architecture is refined to meet architectural constraints, it gets ready for further refinement to achieve the non-functional goals (quality of service and development goals). Many of these goals impose constraints on component interaction; for example, security goals restrict interactions to limit information flows along the channels; accuracy goals impose interactions to maintain a consistent state between related objects and so forth. The refinement of the architecture to accommodate non-functional goals proceeds as follows:

1. For each non-functional terminal in the goal refinement graph G,
   a. All specific connectors and components that G may constrain are identified.
   b. If necessary, G is instantiated to those connectors and components.
2. For each non-functional goal-constrained connector or component, it is refined to meet the instantiated non-functional goal associated with it. Architectural refinement patterns might be used to drive the refinement process as described in [95].

Van Lamsweerde characterizes his architectural extension of KAOS as systematic and incremental in a mix of qualitative and formal reasoning to attain software architectures that meet both functional and non-functional requirements. He argues that deriving architecture that is based on goal-oriented requirements analysis allows for making use of the capabilities of the goal-oriented paradigm. The derived architecture is quite able to accommodate non-functional requirements as well as functional requirements, which is not the case in other paradigms like object-oriented analysis and formal methods. Further, the ability to explore the different alternatives for answering "WHAT" questions lies at the core of goal-orientation allowing for constructive guidance to software architects in their design task.

This extension to KAOS to derive architecture from requirements is similar to FADES in some aspects like employing goal-oriented approaches during requirements analysis in order to derive an architecture using a constructive procedure in the form of transformation rules. However, the generated architecture from Van Lamsweerde's extension to KAOS is not formal like FADES or APL [94]. Further, FADES goes further beyond design and produces requirements into an implementable form while Van Lamsweerde's extension of KAOS generates architectural design with no specific focus on security aspects.

Liu and Yu explored integrating the goal-oriented language GRL and Use Case Maps (UCM) in order to derive architectural design from functional and non-functional requirements [20]. The goal-oriented language GRL is used to support requirements modeling using goal and agent-oriented techniques, and to guide the architectural design process. UCM is a scenario-oriented architectural notation used to express the architectural design at each stage of development. UCM support of scenario orientation allows for visualizing the behavioral aspects of the architecture at varying degrees of abstraction and levels of detail. In the integrated GRL and UCM approach, GRL models are constructed using business goals and non-functional requirements that are refined and operationalized, until some concrete design decisions are launched. These design decisions are further elaborated into UCM scenarios that ask "how" questions instead of "what" questions. Moreover, UCM scenarios describe the behavioral features and architectures of the system in the restricted context of achieving some implicit purpose(s), which basically answers the "what" questions followed by the "why" questions such as [20]:

1. "What the system should do as providing an in-coming call service?"
2. "What is the process of wireless call transmitting?"
3. "Why to reside a function entity in this network entity instead of the other?"

The GRL-UCM integrated approach aims to derive an architectural design from requirements through eliciting, refining and operationalizing requirements incrementally until a satisfying architectural design is launched. The general steps of the process are illustrated in Figure 6. Unlike FADES, the integrated GRL and UCM approach only derives high level architectural design without involving any rigor in the derivation. Further, the integrated approach aims at handling system requirements with no specific focus on security requirements.

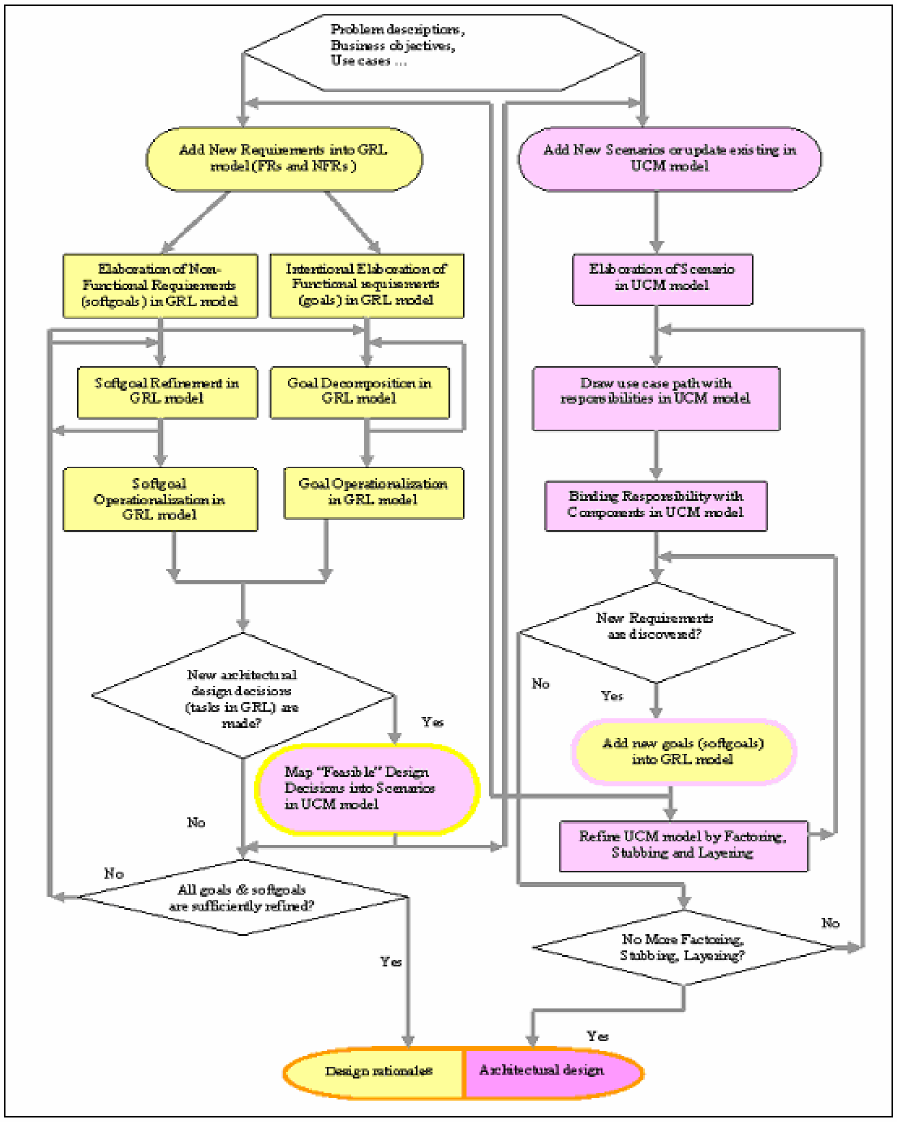

Problem descriptions, Business objectives, Use cases ...
Add New Requirements into GRL model (FRs and NFRs)
Add New Scenarios or update existing in UCM model
Elaboration of Non-Functional Requirements (softgoals) in GRL model
Intentional Elaboration of Functional requirements (goals) in GRL model
Elaboration of Scenario in UCM model
Softgoal Refinement in GRL model
Goal Decomposition in GRL model
Draw use case path with responsibilities in UCM model
Softgoal Operationalization in GRL model
Goal Operationalization in GRL model
Binding Responsibility with Components in UCM model
New Requirements are discovered?
No
Yes
New architectural design decisions (tasks in GRL) are made?
Yes
Add new goals (softgoals) into GRL model
Map "Feasible" Design Decisions into Scenarios in UCM model
No
Refine UCM model by Factoring, Stubbing and Layering
No
All goals & softgoals are sufficiently refined?
No
No More Factoring, Stubbing, Layering?
Yes
Yes
Design rationales
Architectural design

**Figure 6: Integration of goal-oriented and scenario-based modeling**

Mylopoulos et. al. proposed a requirements-driven software development method called TROPOS that supports agent-orientation of software systems [11]. TROPOS adopts the i* modeling framework described in [19], which offers the concepts of actor, goal, and dependency. These concepts are used to model early and late requirements, architectural and detailed design. Tropos main theme is that building a software system that operates in a dynamic environment requires explicit modeling and analysis of this environment in terms of actors, their goals and dependencies on other actors [11]. Tropos supports four phases of software development:

- Early requirements, concerned with understanding the problem by studying an organizational setting; the output of this phase is an organizational model which includes relevant external actors, their respective goals and their interdependencies.

- Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies.
- Detailed design, where behavior of each architectural component is defined in further detail.

Like FADES, TROPOS formally derives detailed design specifications from requirements. Unlike FADES, Tropos does not focus on security aspects but rather on system requirements.

Dromey proposed the GSE method, which formally derives high level architectural design from a set of functional requirements using behavior trees [12]. GSE derives design from requirements through the admission of the prospect that individual functional requirements are regarded as fragments of behavior while a design that satisfies a set of functional requirements is regarded as integrated behavior. Individual functional requirements are formally modeled using behavior trees representation in the GSE approach to enable the transition from requirements to design. Dromey believes that the behavior tree notation solves a fundamental problem of going from a set of functional requirements to a design satisfying those requirements since it provides a clear, simple, constructive and systematic path for this transition [12]. Behavior trees of individual functional requirements may be composed, one at a time, to create an integrated design behavior tree. From this problem domain representation, a direct and systematic transition to a solution domain representation is feasible. The solution domain is represented in the form of component architecture of the system and the behavior designs of the individual components that make up the system. Unlike FADES, GSE only considers functional requirements, which makes it inappropriate to security requirements that are typically classified as non-functional. Further, GSE derives high level architectural design as opposed to implementation specifications in FADES.

Hinchey et. al proposed R2D2C (Requirements to Design to Code) approach, which "offers a mechanical transformation of requirements expressed in restricted natural language or in other appropriate graphical notations into a provably equivalent formal model that can be used as the basis for code generation and other transformations" [13]. Requirements might be expressed using scenarios in constrained (domain-specific) natural language, or in a range of other notations (including UML use cases). Requirements are then transformed to a formal model guaranteed to be equivalent to the requirements stated at the outset. The formal model can be expressed using a variety of formal methods and could be subsequently used as a basis for code generation. Currently, R2D2C is using CSP, Hoare's language of Communicating Sequential Processes, which is suitable for various types of analysis and investigation. CSP could be used for full formal implementations and automated test case generation, etc.

R2D2C involves a number of phases, which are reflected in the system architecture described in Figure 7. The following describes each of these phases.

- *D1 Scenarios Capture*: Engineers, end users, and others write scenarios describing the functionalities that should be offered by the intended system. The input scenarios may be represented in a constrained natural language using a syntax-directed editor, or may be represented in other textual or graphical forms.

- *D2 Traces Generation*: Traces and sequences of atomic events are derived from the scenarios defined in D1.
- *D3 Model Inference*: An automatic theorem prover is used to infer a formal model, expressed in CSP – in this case, ACL2, using the traces derived in phase 2. Concurrency laws need to be deeply embedded in the theorem prover to provide it with sufficient knowledge of concurrency and of CSP to perform the inference.
- *D4 Analysis*: different types of analysis could be performed based on the formal model making use of the currently available commercial or public domain tools, and specialized tools that are planned for development. CSP allows for model analysis at different levels of abstraction using a variety of possible implementation environments.
- *D5 Code Generation*: Existing techniques of automatic code generation from formal models are reasonably well understood and could be applied in the R2D2C approach whether using a tool specifically developed for the purpose, or existing tools such as FDR or converting to other notations suitable for code generation (e.g., converting CSP to B) and then using the code generating capabilities of the B Toolkit.

The generated code might be code in a high-level programming language, low-level instructions for (elector-) mechanical device, natural language business procedures and instructions or the like [13].

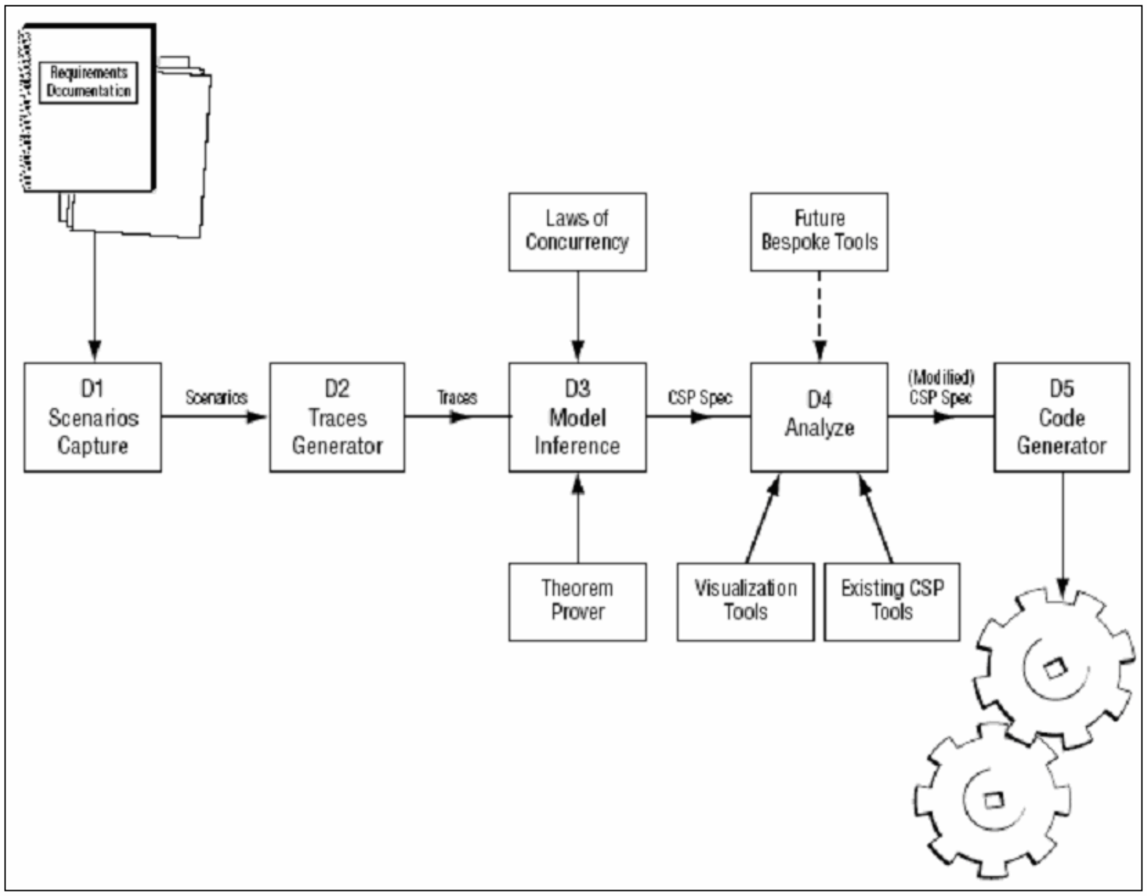


**Figure 7: The Entire Process with D1 thru D5 Illustrating the Development Approach**

R2D2C, the way it is described above, requires significant computing power due to the employment of an automated theorem prover performing significant inferences based on traces input and its knowledge of the concurrency laws. For more applicability of the approach, there is a reduced version of R2D2C called the shortcut version, in which the use of a theorem prover is avoided while maintaining the same level of validity of the approach.

R2D2C has the same spirit as FADES in terms of building a formal model of requirements from which design and code can be automatically generated. R2D2C employs CSP while FADES employs KAOS security extension and the B method. Software projects fully developed using B have not reported performance problems that obstruct the applicability of B to real software due to the level of maturity and stability of the commercial B tools. R2D2C also considers transforming the requirements model specified in CSP to B in order to make use of the code generation capabilities in B tools. Unlike FADES, R2D2C captures requirements only in scenarios, which cannot capture all types of requirements especially security requirements that might not always be suitable for representation in scenarios.

There are a number of approaches proposed in literature to check for consistency and compliance between requirements and design or implementation. This category of approaches is comparable to FADES since FADES allows for the generation of acceptance test cases from the requirements in order to check for the consistency between the derived implementation specifications and the requirements model.

The Ontology for Software Specification and Design (OSSD) approach integrates KAOS and UML to be able to detect errors in software designs against the original requirements [21]. This is achieved through integrating UML with the KAOS framework for elaboration requirements in order to help automate the detection of inconsistencies in UML designs, which enhances the quality of the original design and ultimately integrating the multiple views of UML [21]. OSSD is based on extracting structure, data and relationships from UML design models; abstracts them into an ontology-based integrated model; and creates a specification level representation of the original UML design in a formal, agent-oriented requirements modeling language, which is KAOS.

A simple set of mappings is used to transform the OSSD model to an equivalent KAOS model in order to produce requirements specification that is used as input to an appropriate verification tool in order to detect inconsistencies between the specifications resulted from the OSSD approach and the original requirements. The original UML design is then manually updated based on the results of the verification processing.

“The transformation from UML to OSSD can be summarized as a combined lexical and semantic analysis of the UML Model diagrams, followed by the utilization of multiple mapping tables that enable the creation of an instance of the OSSD model” [21]. The Upper Merged Ontology (SUMO), WordNet, Browser helps with the categorization of terminology used in the UML diagrams. The first step in the approach is to identify the Object, Attribute, Relation and Behavior Constructs of the OSSD Model using UML class diagrams. Behavior and behavior constraints are refined through the processing of the UML Sequence Diagrams. The processing of the UML State Machine Diagram refines behavior constraints and identifies the States and Transitions in the OSSD Model. Lastly, the OSSD processing of the UML Use Case Diagram identifies the Goals associated with Objects and Behavior in the OSSD Model. The OSSD is depicted in Figure 8.

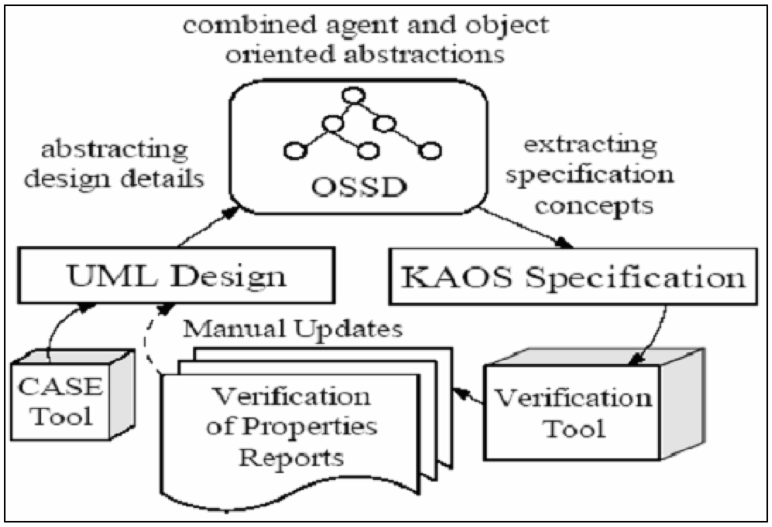

combined agent and object oriented abstractions
abstracting design details
OSSD
extracting specification concepts
UML Design
KAOS Specification
Manual Updates
CASE Tool
Verification of Properties Reports
Verification Tool

**Figure 8: OSSD Approach**

OSSD has the same objective as FADES in producing high quality design; however, OSSD does not have a specific focus on security and it does not illustrate how to deal with UML semantic inconsistencies. Further, OSSD contemplates only on the design phase of software engineering with no strong focus on either requirements or implementation like FADES.

Liu et. al. proposed a formalization of UML that defines both a UML model of requirements and another of design as a pair of class diagram and a family of sequence diagrams [22, 23, 24]. Both models of requirements and design are then given unified semantics. The approach then defines consistency between a design class diagram and the interaction diagrams and shows how the removal of inconsistency can be treated as a model refinement. Finally, the approach formally defines the correctness of UML model of design with respect to the model of requirements. This approach supports a "use case, step-wised and incremental development in building models for requirements analysis" [22].

In this formalization approach of UML, class models and use cases are used to capture requirements. The class model is relatively conceptual, which means that classes do not have methods and the associations among conceptual classes are undirected. The functional requirements are described by use cases and each use case is represented by a sequence diagram called a system sequence diagram. On the other hand, the design model consists of a design class model and a family of sequence diagrams. Classes in this class model now have methods and a method of a class may call methods of other classes. Therefore, the specification of these methods must agree with the object interactions in the sequence diagrams.

The formalization of both the requirements model and the design model allows for checking consistency between requirements and design [24]. However, the approach uses UML to build the requirements model, which suffers from the inherent inconsistencies and informality of UML that might result in building a low quality requirement model. Moreover, the formalization approach allows for eliciting and analyzing functional requirements while marinating the limitations of UML to capture non-functional requirements such as security.

Ledang and Souquieres proposed translating UML specifications to formal B specifications in order to rigorously analyze UML specifications via their corresponding B formal specifications [25, 26, 27]. This approach suggests a formalization of each UML construct as follows:

- Use case translation to B: each use case is modeled as a B operation. To express in B the pre- and post-conditions of use cases, each use case and its involved classes are modeled in the same abstract machine. By structuring use cases, they are organized into levels. The use cases at level one corresponds to "user-goal" use cases. The use cases, which are the included cases of the ones at level one, are said at level two and so on. The bottom level of use cases is composed of basic operations of classes.
- Modeling class operations: each class operation is modeled as a B operation in an abstract machine. As for use cases, the class operation and its involved data are grouped in the same abstract machine. In addition, the calling-called class operation dependency to arrange derived B operations into abstract machines is used.
- Modeling state charts: this happens in two stages
    - Creating a B abstract operation for each event. In the B abstract operation, the expected effects of the event is directly specified on the data derived from class data related by the event. Consequently, an event and its related data are modeled in the same abstract machine.
    - Implementing (or refining) the B operation in the first step by calling B operations for the triggered transition and actions.

Several kinds of analysis on UML specifications can be done after the translation to B such as the following:

i. The consistency of class invariant.
ii. The conformity of object and state chart diagrams regarding the class diagrams.
iii. The conformity of class operations, use cases regarding the class invariant.
iv. The class operation calling-called dependency.
v. The use case structuring.

The transformation of UML to B is close in methodology to the FADES though it is different in its objective. That is, the main objective of translating UML to B is to allow for formal analysis of UML specifications through their corresponding B specifications. The other objective of the UML formalization is to use UML specifications as a tool for building B specifications, so the development of B specifications becomes easier. On the other hand, FADES transforms KAOS requirements model to B in order to allow for the refinement of security requirements, which are critical, into an implementation with a high confidence that the generated implementation meets the security requirements and preserves the security properties. Unlike FADES, the UML formalization falls short when it is applied to security requirements as UML does not have specific constructs to model security specifications.

Blackburn et.al. introduced the Test Automation Framework (TAF), which is a model-based verification approach that has been effective in detecting and correcting requirement defects early in the development process [28-31]. The TAF main objective is to reduce the manual test development effort and reduce rework though the integration of various government and commercially available model development and test generation tools to support defect prevention and automated testing of systems [28].

TAF supports modeling methods that focus on representing requirements, like the Software Cost Reduction (SCR) method, as well as methods that focus on representing design information, like SimulinkB or MATRIXx, which supports control system modeling for aircraft and automotive systems [29]. Blackburn uses model translation means to convert requirement-based or design-based models to a form understandable by T-VEC, the test generation component of TAF, to produce test vectors [30]. Test vectors include both inputs and the expected outputs along with requirement-to-test traceability information. T-VEC also supports test driver generation, requirement test coverage analysis, and test results checking and reporting. The test drivers are then used to test the implementation functionalities during the testing phase [31].

Like FADES, TAF derives test cases from the requirements model in order to verify the consistency and compliance between requirements and implementation and to provide sufficient traceability information. However, TAF is stronger in generating different types of test cases including unit testing, integration testing and acceptance testing while FADES only generates acceptance test cases. This difference is due to the fact that TAF is a specialized testing framework while FADES provides the generation of acceptance test cases as an extra verification step to ensure that the derived implementation meets the initial set of security requirements. It might be part of FADES' future work to apply deeper analysis to the KAOS requirements model to generate other type of test cases.

Table 1 categorizes the various approaches outlined above and summarizes their similarities and differences. Further, Table 1 positions FADE in its proper location compared to other relevant approaches

| **Approach Type** | **Approach** | **Formality Entailed** | **Security Requirements Support** | **Level of Details of The Derived Design** | **Requirements Traceability** |
|---|---|---|---|---|---|
| Elaboration and modeling of security requirements | Misuse & security use cases | Informal | Introduced in UML to capture security requirements | High-level design | Partial traceability |
| | i* security extension | Informal | Extends i* to support security | No derived design | Not applicable |
| Deriving design from | VDM++ generator | Formal | Supports non-functional requirements but no specific | High-level design | Support requirements traceability |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| requirements |  |  | support of security |  |  |
|  | GRL & UCM integration | Informal | Supports non-functional requirements but no specific support of security | High-level architectural design | Supports requirements traceability |
|  | TROPOS | Formal | Supports non-functional requirements but no specific support of security | Detailed design | Supports requirements traceability |
|  | GSE | Formal | No support for non-functional requirements and security | High- level architectural design | Support requirements traceability |
|  | R2D2C | Formal | Not specific to security | Detailed design & implementation with performance problems | Supports requirements traceability |
|  | Biological database schema design | Formal | Supports database requirements | Database conceptual schema design | Supports requirements traceability |
|  | KAOS to APL | Formal | Functional and non-functional requirements but no specific support of security | Architecture | Supports requirements traceability |
|  | Lamsweerde extension of KAOS to architecture | Formal | Functional and non-functional requirements but no specific support of security | Architecture | Supports requirements traceability |
|  | FADES | Formal | Specific to security | Detailed design & implementation specification | Supports requirements traceability |
| Consistency check between requirements and design | OSSD | Formal | Not specific to security | No design derivation | Supports requirements traceability |
|  | UML formalization | Formal | Not specific to security | No design derivation | Supports requirements traceability |
|  | UML to B | Formal | Not specific to security | No design derivation | Supports requirements traceability |
|  | TAF | Formal | Not specific to security | Derivation of test drivers | Supports requirements traceability |

**Table 1: Classification of Related Work**

# Chapter 2: Background

FADES is a requirements-driven software engineering approach that formally derives design and implementation specifications from a set of security requirements. Therefore, FADES needs a requirements elaboration method and a design elaboration method in order to cover all the stages of development until implementation is obtained. FADES employs the KAOS security extension for eliciting, modeling and analyzing security requirements while it employs the B formal development method for deriving design and implementation specifications. This chapter outlines the underlying methods adopted by FADES namely the KAOS security extension and the B formal method.

## 2.1 KAOS – Goal-Oriented Requirements Engineering

Van Lamsweerde in [3, 33] has described KAOS as a general approach for eliciting, analyzing and modeling functional and non-functional requirements of software systems based on first-order temporal logic. Van Lamsweerde in [1, 32] has then extended KAOS to handle security requirements. Section 2.1 describes the general KAOS framework and section 2.2 illustrates the KAOS security extension.

KAOS is a requirement engineering method concerned with the elicitation of goals to be achieved by the envisioned system, the operationalization of such goals into specifications of services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software [33]. KAOS employs some techniques based on a temporal logic formalization of goals and domain properties with the aim of deriving more realistic, complete, and robust requirements specifications. The key concept in KAOS is to handle exceptions at requirements engineering time and at the goal level, so that more freedom is left for resolving them in a satisfactory way.

The KAOS framework supports the whole process of requirements elaboration, from the high level goals to be achieved to the requirements, objects, and operations to be assigned to the various agents in the composite system [35]. The methodology provides a specification language, an elaboration method, meta-level knowledge used for local guidance during method enactment, and tool support [38]. The following subsections will describe each of the methodology steps.

### 2.1.1 Concepts and Terminology

KAOS specification language provides constructs for capturing various kinds of concepts that appear during requirements elaboration namely goals, constraints, agents, entities, relationships, events, actions, views and scenarios. There is one construct for each type of concept. The types are defined first followed by the constructs for specifying their instances [37].

- Objects: an object is a thing of interest in the domain whose instances may evolve from state to state [40]. They can be:
    - Agents: active objects
    - Entities: passive objects
    - Events: instantaneous objects
    - Relationships: depend on other objects
- Operations: an operation is an input-output relation over objects. Operation applications define state transitions. Operations are characterized by pre/post and trigger conditions [36]. A distinction is made between domain pre/post conditions, which capture the elementary state transitions defined by operation applications in the domain, and required pre/post conditions, which capture additional strengthening to ensure that the requirements are met [41].
- Goal: it's an objective for the system. In general, a goal can be AND/OR refined till we obtain a set of goals achievable by some agents by performing operations on some objects [39]. The refinement process generates a refinement directed acyclic graph. AND-refinement links relate a goal to a set of subgoals (called refinement); this means that satisfying all subgoals in the refinement is a sufficient condition for satisfying the goal. OR-refinement links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is a sufficient condition for satisfying the goal. Goals often conflict with others. Goals concern the objects they refer to [35].
- Requisites, requirements and assumptions: the leaves obtained in the goal refinement graph are called requisites. The requisites that are assigned to the software system are called requirements; those assigned to the interacting environment are called assumptions [32].
- Domain property: is a property about objects or operations in the environment which holds independently of the software-to-be. Domain properties include physical laws, regulations, constraints imposed by environmental agents, indicative statements of domain knowledge [34]. In KAOS, domain properties are captured by domain invariants attached to objects and by domain pre/post conditions attached to operations [42].
- Scenario: is a domain-consistent sequence of state transitions controlled by corresponding agent instances; domain consistency means that the operation associated with a state transition is applied in a state satisfying its domain precondition together with the various domain invariants attached to the corresponding objects, with a resulting state satisfying its domain post condition.

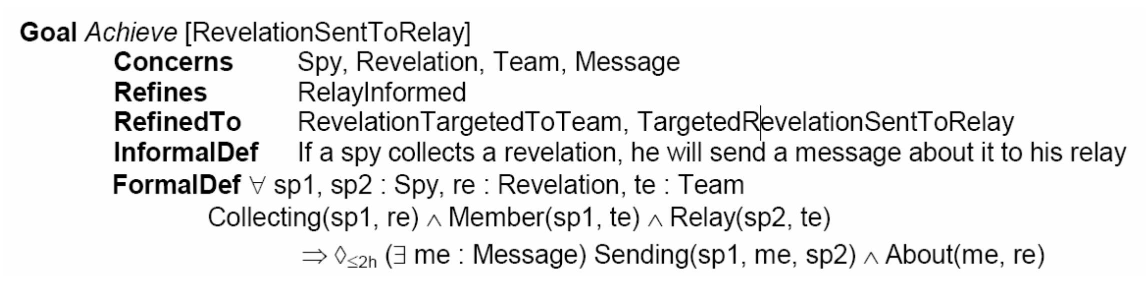
Goals are classified according to the category of requirements they will drive about the agents concerned [69]. Functional goals result in functional requirements. For example, SatisfactionGoals are functional goals concerned with satisfying agent requests; InformationGoals are goals concerned with keeping agents informed about object states. Likewise, nonfunctional goals result in nonfunctional requirements. For example, AccuracyGoals are nonfunctional goals concerned with maintaining the consistency between the state of objects in the environment and the state of their representation in the software; other subcategories include SafetyGoals, SecurityGoals, PerformanceGoals, and so on [36].

Goal refinement ends when every subgoal is realizable by some individual agent assigned to it, that is, expressible in terms of conditions that are monitorable and controllable by the agent. A requirement is a terminal goal under responsibility of an agent in the software-to-be; an expectation is a terminal goal under responsibility of an agent in the environment (unlike requirements, expectations cannot be enforced by the software-to-be) [76].

### 2.1.2 The Specification Language

Each construct of the KAOS specification language has a two-level generic structure: an outer semantic net layer for declaring a concept, its attributes and its various links to other concepts; an inner formal assertion layer for formally defining the concept [36]. The declaration level is used for conceptual modeling (through a concrete graphical syntax), requirements traceability (through semantic net navigation) and specification reuse (through queries). The assertion level is optional and used for formal reasoning [39].

The generic structure of a KAOS construct is instantiated to specific types of links and assertion languages according to the specific type of the concept being specified. For example, consider the following goal specification for a secret message sending system [40]:



**Goal** *Achieve* [RevelationSentToRelay]
**Concerns** Spy, Revelation, Team, Message
**Refines** RelayInformed
**RefinedTo** RevelationTargetedToTeam, TargetedRevelationSentToRelay
**InformalDef** If a spy collects a revelation, he will send a message about it to his relay
**FormalDef** $\forall$ sp1, sp2 : Spy, re : Revelation, te : Team
Collecting(sp1, re) $\wedge$ Member(sp1, te) $\wedge$ Relay(sp2, te)
$\Rightarrow \Diamond_{\leq 2h}$ ($\exists$ me : Message) Sending(sp1, me, sp2) $\wedge$ About(me, re)

The declaration part of this specification introduces a concept of type "goal", named RevelationSentToRelay, stating a target property that should eventually hold ("Achieve" verb), referring to objects such as Spy or Revelation, refining the parent goal RelayInformed, refined into subgoals RevelationTargetedToTeam and TargetedRevelationSentToRelay, and defined by some informal statement. The optional assertion part in the specification above defines the goal Achieve [RevelationSent] in formal terms using a real-time temporal logic. In this document, the following classical operators for temporal referencing are used [35]:

| | |
|---|---|
| $\circ$ (in the next state) | $\bullet$ (in the previous state) |
| $\Diamond$ (some time in the future) | $\blacklozenge$ (some time in the past) |
| $\square$ (always in the future) | $\blacksquare$ (always in the past) |
| $\mathcal{W}$ (always in the future unless) | $\mathcal{U}$ (always in the future until) |
| $\mathcal{B}$(always in the past back to) | $\mathcal{S}$ (always in the past since) |

Formal assertions are interpreted over historical sequences of states. Each assertion is in general specified by some sequences and falsified by some other sequences. The notation (H, i) |= P is used to express that assertion P is satisfied by history H at time position i (i € T), where T denotes a linear temporal structure assumed to be discrete for sake of simplicity. We will also use the notation H |= P for (H, 0) |= P.

States are global; the state of the composite system at some time position i is the aggregation of the local states of all its objects at that time position. The state of an individual object instance ob at some time position is defined as a mapping from ob to the set of values of all ob's attributes and links at that time position. In the context of KAOS requirements, a historical sequence of states defines a behavior produced by a scenario [38].

The semantics of the above temporal operators is then defined as follows [33]:

| | |
|---|---|
| $(H, i) \models \circ P$ | iff $(H, next(i)) \models P$ |
| $(H, i) \models \Diamond P$ | iff $(H, j) \models P$ for some $j \geq i$ |
| $(H, i) \models \square P$ | iff $(H, j) \models P$ for all $j \geq i$ |
| $(H, i) \models P\mathcal{U}Q$ | iff there exists a $j \neq i$ such that $(H, j) \models Q$ and for every k, $i \leq k < j$, $(H, k) \models P$ |
| $(H, i) \models P\mathcal{W}Q$ | iff $(H, i) \models P\mathcal{U}Q$ or $(H, i) \models \square P$ |
| $(H, i) \models P\mathcal{S}Q$ | iff there exists a $j \leq i$ such that $(H, j) \models Q$ and for every k, $j < k \leq i$, $(H, k) \models P$ |
| $(H, i) \models P\mathcal{B}Q$ | iff $(H, i) \models P\mathcal{S}Q$ or $(H, i) \models \blacksquare P$ |

Note that $\square$ P amounts to $P\mathcal{W}$ false. We will use the standard logical connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implies), $\leftrightarrow$ (equivalent), $\Rightarrow$ (strongly implies), $\Leftrightarrow$ (strongly equivalent), with

$$P \Rightarrow Q \text{ iff } \square (P \rightarrow Q)$$
$$P \Leftrightarrow Q \text{ iff } \square (P \leftrightarrow Q)$$

Note thus that there is an implicit outer $\square$- operator in every strong implication.

Beside the agent-related classification of goals introduced in section 2.1.1, goals in KAOS are also classified according to the pattern of temporal behavior they capture:

| | |
|---|---|
| Achieve: | $C \Rightarrow \Diamond T$ |
| Cease: | $C \Rightarrow \Diamond \neg T$ |
| Maintain: | $C \Rightarrow T\mathcal{W} N$, $C \Rightarrow T$ |
| Avoid: | $C \Rightarrow \neg T\mathcal{W} N$, $C \Rightarrow \neg T$ |

In these patterns, C, T, and N denote some current, target, and new condition respectively.

In requirements engineering, it is needed to introduce real-time restrictions. Bounded versions of the above temporal operators are therefore introduced in the following style:

| | |
|---|---|
| $\lozenge_{\leq}d$ | (some time in the future within deadline d) |
| $\square_{\leq}d$ | (always in the future up to deadline d) |

To define such operators, the temporal structure T is enriched with a metric domain D and a temporal distance function dist: $T \times T \longrightarrow D$, which has all desired properties of metrics. We will take:

| | |
|---|---|
| T: | the set of naturals |
| D: | { d \| there exists a natural n such that d = n × u}, where u denotes some chosen time unit |
| dist(i, j): | \| j - i \| × u |

Multiple units can be used – e.g., s (second), m (minute), d (day), etc; these are implicitly converted into some smaller unit. The o-operator then yields the nearest subsequent time position according to this smallest unit.
The semantics of the real-time operators is then defined accordingly, e.g.,

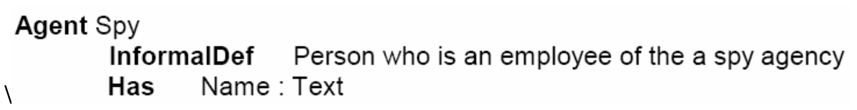
| | |
|---|---|
| $(H, i) \models \lozenge_{\leq}d\ P$ | iff $(H, j) \models P$ for some $j \neq i$ with $dist(i, j) \leq d$ |
| $(H, i) \models \square_{<}d\ P$ | iff $(H, j) \models P$ for all $j \neq i$ such that $dist(i, j) < d$ |

In the above goal declaration of RevelationSentToRelay, the conjunction of the assertions formalizing the subgoals RevelationTargetedToTeam and TargetedRevelationSentToRelay must entail the formal assertion of the parent goal RevelationSent they define together. Every formal goal refinement thus generates a corresponding proof obligation [34].

In the formal assertion of the goal RevelationSentToRelay, the predicate Sending(sp1, me, sp2) means that, in the current state, an instance of the Sending relationship links variables sp1, me, and sp2 of sort Spy, Revelation, respectively. The Sending relationship and Revelation entity are defined in other sections of the specification, e.g.,

```
Entity Revelation
        InformalDef    Secret information about the enemy
        Has     Content : Text

Relationship Sending
        InformalDef    A spy is sending a message to another spy
        Links   Spy {Role Sends}
                Message {Role Sent}
                Spy {Role To}
```

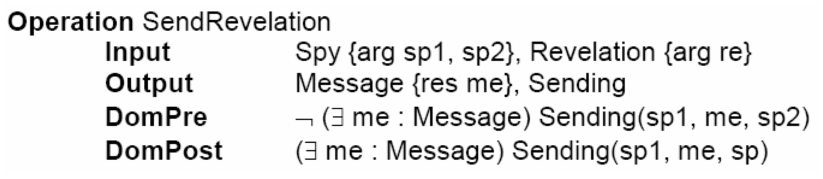
The Spy type might in turn be declared by



```
Agent Spy
        InformalDef    Person who is an employee of the a spy agency
        Has      Name : Text
```

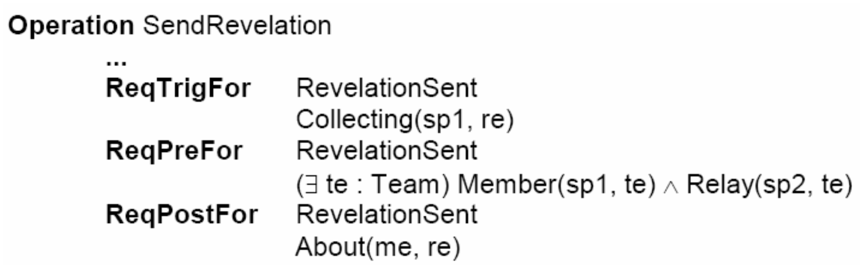
\

In the declarations above, Name is declared as an attribute of the entity Spy.
As mentioned earlier, operations may be specified formally by pre and post conditions in the state-based style, e.g.,



```
Operation SendRevelation
        Input          Spy {arg sp1, sp2}, Revelation {arg re}
        Output         Message {res me}, Sending
        DomPre         ¬ (∃ me : Message) Sending(sp1, me, sp2)
        DomPost        (∃ me : Message) Sending(sp1, me, sp)
```

The pre and post condition of the operation SendRevelation above are domain properties; they capture corresponding elementary state transitions in the domain, namely, from a state where no message is sent to a state where a message is sent. The software requirements are found in the terminal goals assigned to agents in the software –to-be, and in the additional pre, post and trigger conditions that need to strengthen the corresponding domain conditions in order to ensure all such goals. Assuming the RevelationSentToRelay goal is assigned to the spy collecting the revelation one would derive from the above formal assertion for that goal:



```
Operation SendRevelation
        ...
        ReqTrigFor     RevelationSent
                       Collecting(sp1, re)
        ReqPreFor      RevelationSent
                       (∃ te : Team) Member(sp1, te) ∧ Relay(sp2, te)
        ReqPostFor     RevelationSent
                       About(me, re)
```

The trigger condition captures an obligation to trigger the operation as soon as the condition gets true and provided the domain precondition is true. The specification will be consistent provided the trigger condition and required precondition are together true in the operation's initial state.

## 2.1.3 The Elaboration Method

Figure 9 outlines the major steps that may be followed to elaborate KAOS specifications from high-level goals.

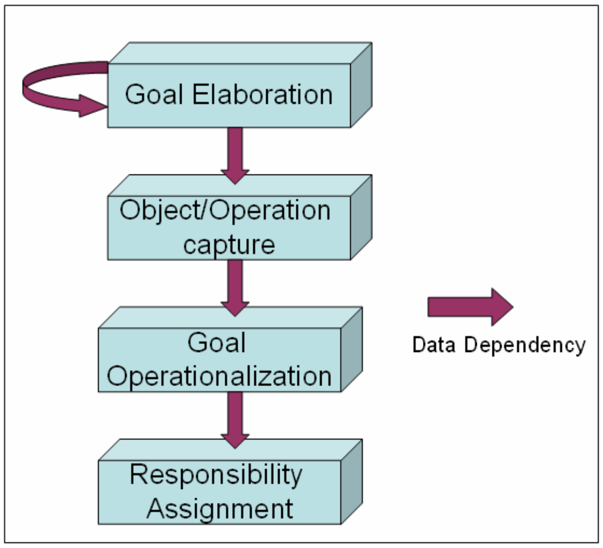


**Figure 9: KAOS requirements elaboration**

- *Goal elaboration*. Elaborate the goal AND/OR structure by defining goals and their refinement links until assignable goals are reached. The process of identifying goals, defining them precisely, and relating them through refinement links is in general a combination of top-down and bottom-up subprocesses; offspring goals are identified by asking HOW questions about goals already identified whereas parent goals are identified by asking WHY questions about goals and operational requirements already identified [39].
- *Object capture*. Identify the objects involved in goal formulations, define their conceptual links, and describe their domain properties by invariants [34].
- *Operation capture*. Identify object state transitions that are meaningful to the goals. Goal formulations refer to desired or forbidden states that are reachable through state transitions; the latter correspond to applications of operations. The principle is to specify such state transitions as domain pre- and post conditions of operations thereby identified and to identify the agents that could perform these operations [33].
- *Operationalization*. Derive strengthened pre-, post-, and trigger conditions on operations and strengthened invariants on objects, in order to ensure that all terminal goals are met. A number of formal derivation rules are available to support the operationalization process [3].
- *Responsibility assignment*. 1) Identify alternative responsibilities for terminal goals; 2) make decisions among refinement, operationalization, and responsibility alternatives, so as to reinforce nonfunctional goals e.g., goals related to reliability, performance, cost reduction, load reduction, etc; 3) assign the operations to agents that can commit to guarantee the terminal goals in the

alternatives selected. The boundary between the system and its environment is obtained as a result of this process and the various terminal goals become requirements or assumptions dependent on the assignment made [38].

The steps above are ordered by data dependencies; they may be running concurrently, with possible backtracking at every step.

### 2.1.4 Obstacle Analysis and Resolution

Obstacles were first introduced in [3] as means for identifying goal violation scenarios. First-sketch specifications of goals, requirements, and assumptions tend to be too ideal; they are likely to be occasionally violated in the running system due to unexpected agent behavior. The objective of obstacle analysis is to anticipate exceptional behaviors in order to derive more complete and realistic goals, requirements, and assumptions. A defensive extension of the goal-oriented process model outlined above is depicted in Figure 10. During elaboration of the goal graph by elicitation and by refinement, obstacles are generated from goal specifications. Such obstacles may be recursively refined as indicated by the right circle arrow in Figure 10.

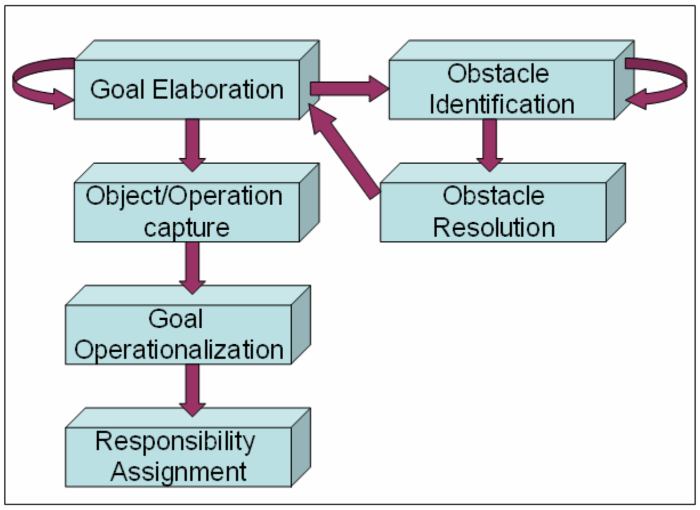


**Figure 10: Obstacle analysis in goal-oriented requirements elaboration**

In declarative terms, an obstacle to some goal is a condition whose satisfaction may prevent the goal from being achieved. An obstacle O is said to obstruct a goal G in some domain characterized by a set of domain properties Dom if and only if

$$\{O, Dom\} \models \neg G \qquad \textit{obstruction}$$

$$Dom \not\models \neg O \qquad \textit{domain consistency}$$

Obstacle analysis consists in taking a pessimistic view at the goals, requirements and expectations being elaborated. The principle is to identify as many ways of obstructing them as possible in order to resolve each such obstruction when likely and critical so as to produce more complete requirements for more robust systems. Formal techniques for generation ad AND/OR refinement of obstacles are detailed in [3, 34].

The basic technique amounts to a precondition calculus that regresses goal negations $\neg G$ backwards through known domain properties Dom. Formal obstruction patterns may be used as a cheaper alternative to shortcut formal derivations. Both techniques allow domain properties involved in obstructions to be incrementally elicited as well [33].

Obstacles that appear to be likely and critical need to be resolved once they have been generated. Resolution tactics are available for generating alternative solutions, notably, goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation [3]. The selection of preferred alternatives depends on the degree of criticality of the obstacle, its likelihood of occurrence and on high-priority soft goals that may drive the selection. The selected resolution may then be deployed at specification time, resulting in specification transformation, or at runtime through obstacle monitoring.

Obstacle resolution results in a goal structure updated with new goals and/or transformed versions of existing ones. The new goal specifications obtained by resolution may in turn trigger a new iteration of goal elaboration and obstacle analysis. Goals obtained from obstacle resolution may also refer to new objects/operations and require specific operationalizations [37].

## 2.2 KAOS Security Extension

The KAOS requirements engineering approach described in section 2.1 is customized in [1, 32] to elicit, capture, and analyze security requirements. The general idea it to build two models iteratively and concurrently:

- A model of the system-to-be that covers both the software and its environment and inter-relates their goals, agents, objects, operations, requirements and assumptions.
- An anti-model derived from the model that exhibits how specifications of model elements could be maliciously threatened, why and by whom.

Security requirements are elaborated systematically by iterating the following steps:

1. Instantiate specification patterns associated with property classes such as confidentiality, privacy, integrity, availability, authentication or non-repudiation.
2. Derive anti-model specifications threatening such specifications.
3. Derive alternative countermeasures to such threats and define new requirements by selection of alternatives that best meet other quality requirements from the model.

As stated above, this security requirements methodology builds on the KAOS goal-oriented framework developed for generating and resolving obstacles to requirements achievements. The extension to malicious obstacles allows the obstacle refinement process to be guided by the attacker's own goals and by the target of deriving observable vulnerabilities and implementable threats. The extension also includes security-specific operators for resolving malicious obstacles.

## 2.2.1 Specification Patterns for Security Goals

The preliminary elicitation of security-related goals is driven by application-specific instantiations of generic specification patterns. The patterns are associated with specializations of the SecurityGoal meta-class, namely, Confidentiality, Integrity, Availability, Privacy, Authentication and Non-repudiation goal subclasses [1]. Patterns refer to meta-classes from the language meta-model (such as Object and Agent). For each subclass of SecurityGoal, the instantiation of the corresponding specification pattern to "sensitive" attributes/associations from the object model yields corresponding candidates for application-specific security goals (the latter may then need to be refined if necessary) [32]. To support formal analysis, the specification patterns may be formalized in a first-order, real-time linear temporal logic augmented with epistemic constructs for security-related predicates.

In particular, the epistemic operator KnowsVag is defined on state variables as follows:

$$\mathrm{KnowsV}_{ag}(v) \equiv \exists x\colon \mathrm{Knows}_{ag}(x = v) \qquad \text{(“knows value”)}$$

$$\mathrm{Knows}_{ag}(P) \equiv \mathrm{Belief}_{ag}(P) \land P \qquad \text{(“knows property”)}$$

Where the operational semantics of the epistemic operator Beliefag(P) is "P is among the properties stored in the local memory of agent ag". Domain-specific axioms must make it precise under which conditions property P does appear and disappear in the agent's memory. An agent thus knows a property if that property is found in its local memory and it is indeed the case that the property holds.

## 2.2.2 Building Intentional Threat Models

As noted in [3], obstacles may obstruct safety or security goals; obstacle refinement graphs then correspond to the popular fault trees used for modeling or documenting hazards in safety-critical systems and to the popular threat trees used for modeling or documenting potential attacks in security-critical systems. There are two significant differences, however.

- Obstacle directed acyclic graphs are goal-anchored as their root is a goal negation; the analyst thus knows exactly where to start the analysis from - it is often much easier to concentrate first on what is desired rather than on what is not desired [32].
- Obstacles can be formally generated by regressing goal negations through domain properties and other goal assertions, or by using formal obstruction and refinement patterns [32].

In the context of security engineering, standard obstacle analysis appears to be too limited for handling malicious obstacles; the reasons are the following.

- The goals underlying malicious obstacles are not captured; one can therefore not use them for driving the obstacle refinement process.
- There is no modeling of attacker agents and their capabilities in terms of operations they can perform and objects they can monitor/control; one can therefore not reason about them.
- Software vulnerabilities are not explicit in standard models; one can therefore not reason about them or, better, derive them.
- The outcome of the obstacle likelihood and criticality assessment process may be quite different; for example, standard obstacle analysis for a ground collision avoidance component of an air traffic control system might miss the obstacle of multiple planes crashing into adjacent buildings at almost the same time, or assess it to be extremely unlikely, whereas the explicit incorporation of terrorist agents and their goal of causing major damage to symbols of economic power might result in totally different conclusions.

Richer models should thus be built to capture attackers, their goals and capabilities, the software vulnerabilities they can monitor or control, and attacks that satisfy their goals based on their capabilities and on the system's vulnerabilities. Such models are called anti-models while the attacker's own goals are called anti-goals including malicious obstacles to security goals. Anti-goals should of course be distinguished from the goals the system under consideration should satisfy. Anti-models should lead to the generation of more subtle threats and the derivation of more robust security requirements as anticipated countermeasures to such threats [1].

The following steps describe the procedure for building intentional anti-models in a systematic way [32]. This procedure corresponds to a dual version of the goal-oriented requirements elaboration method described in the previous section.

1. Get initial anti-goals by negating relevant Confidentiality, Privacy, Integrity and Availability goal specification patterns instantiated to sensitive objects from the object model.
2. For each such anti-goal, elicit potential attacker agents that might own the anti-goal, from questions such as "WHO can benefit from this anti-goal?" (Application specific specializations of known attacker taxonomies may help answering such questions).
3. For each anti-goal *and* corresponding attacker class(es) identified, elicit the attacker's higher-level anti-goals from questions such as "WHY would instances of this attacker class want to achieve this anti-goal?". Such questions may be asked recursively to elicit more and more abstract anti-goals yielding threat rationales together with other potential threats from alternative refinements of those higher-level anti-goals.
4. Elaborate the anti-goal AND/OR graph by AND refining/ abstracting anti-goals along alternative branches, with the aim of deriving terminal anti-goals that are realizable either by the identified attacker agents or by attackee software agents. The former are anti-requirements assigned to the

attacker whereas the latter are vulnerabilities assigned to the attackee. (This step may be performed informally by asking HOW/WHY questions, or formally by regression through the goal model and the domain theory or by use of refinement patterns and obstruction patterns.)

5. Derive the object and agent anti-models from anti-goal specifications. The boundary between the anti-machine (under the attacker's control) and the anti-environment (which includes the software attackee) are thereby derived together with monitoring/control interfaces [24].
6. AND/OR-operationalize all anti-requirements in terms of potential capabilities of the corresponding attacker agent – the latter may include blind or intelligent searching, eavesdropping, deciphering, spoofing, cookie installation, etc.

In Step 4, an anti-goal is said to be realizable by some agent if it is formulated in terms of conditions monitorable and controllable by the agent. As indicated in Step 4, the AND/OR refinement/abstraction process may be guided by the following techniques:

- Asking "HOW?" and "WHY?" questions about the anti-goals already found.
- Regressing anti-goal specifications through domain properties to find out anti-goal preconditions that can be satisfied in the domain – this corresponds to situations where the attacker exploits features of the domain to achieve her anti-goals.
- Regressing anti-goal specifications through goal specifications from the primal model to find out anti-goal preconditions that can be satisfied by the software – this corresponds to situations where the attacker exploits features of the software itself to achieve her anti-goals.
- Applying formal refinement patterns – notably, the "milestone", "decomposition-by-case", "resolve lack of monitorability" and "resolve lack of controllability" patterns and obstruction patterns specified in [41].

### 2.2.3 Generating Countermeasures

Once intentional anti-models have been built systematically in this way, the next step in the model/anti-model building cycle is to consider alternative countermeasures to the various vulnerabilities and anti-requirements found. A preferred countermeasure is then selected based on

1. The severity and likelihood of the corresponding threat.
2. Non-functional goals that have been identified in the primal goal model. The selected countermeasure yields a new security goal to be integrated in the latter model.

   Alternative countermeasures may be produced systematically using operators similar to those described in [3] for resolving obstacles, e.g.,

   - Goal substitution: develop an alternative refinement of the parent goal to prevent the original subgoal from being threatened by the anti-goal.
   - Agent substitution: replace a vulnerable agent assigned to a threatened goal by a less vulnerable one for the threatening anti-goal.

- Goal weakening: weaken the specification of the goal being threatened so as to make it circumvent the threatening anti-goal;
- Goal restoration: introduce a new goal prescribing appropriate restoration measures in states where the goal has been threatened by the anti-goal.
- Anti-goal mitigation: tolerate the anti-goal but mitigate its effects.
- Anti-goal prevention: add a new goal requiring the anti-goal to be Avoided.

The above list may be extended with security-specific operators such as the following:

- Protect vulnerability: make the derived vulnerability condition unmonitorable by the attacker.
- Defuse threat: make the derived anti-requirement condition uncontrollable by the attacker.
- Avoid vulnerability: add a new goal requiring the software vulnerability condition to be Avoided.

Once alternative anti-goal resolutions have been produced by application of such operators a preferred one has to be selected based on how critical the goal being threatened is and on how well the resolution meets other non-functional goals. The NFR qualitative framework may be used here to support the selection process.

The new security goal thereby retained has to be AND/OR refined in turn until requirements/expectations are reached. A new model/anti-model building cycle may then be required. For anti-goals that are not too severe, resolutions may be deferred from specification time to run time using anti-goal monitoring and intrusion detection technology.

## 2.3 The B Method

B is a method for specifying, designing and coding software systems. It is based on Zermelo-Fraenkel set theory with the axiom of choice, the concept of generalized substitution and on structuring mechanisms (machine, refinement, and implementation) [45]. Proof based development methods like B integrate formal proof techniques in the development of software systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. The relationship between two successive models in this sequence is that of refinement. The essence of the refinement relationship is that it preserves already proved system properties including safety properties and termination. That is why the B method has been employed for industrial development of highly trusted software after decades of academic research on program specification and refinement [43]. B models are accompanied by mathematical proofs that justify them. Proofs of B models convince the user (designer or specifier) that the (software) system is effectively correct. Such proofs are called proof obligations, which guarantee correctness of development. Proof obligations are discharged by the proof tool using automatic and interactive proof procedures supported by a proof engine. According to [7], the B method is characterized with the following:

- It offers a rich collection of set-theoretic data types for an abstract specification of the state of systems.
- It allows the use of standard first-order predicate logic for the specification of operations on the state.
- It uses a relational semantics for statements and supports consistency and correctness proofs of operations by weakest precondition calculation.
- It supports grouping of operations and encapsulation of state variables in modules called machines. The notation for describing abstract machines is known as Abstract Machine Notation (AMN).
- It allows the construction of new machines out of existing ones.

The constructs that determine and change the state of a machine are called substitutions; these correspond to what would be called statements in a programming language. The semantics of substitutions are defined by Generalized Substitutions. The concept of a substitution arises as follows:

- In general, any construct that changes a machine can only do so by changing the state, since the state is the only part of a machine that persists and is changeable.
- The principal construct for changing the state of the machine is the simple substitution x := E, which changes the value of the variable x to the value of the expression E. This construction is recognizable as the assignment statement found in all procedural programming languages. Ultimately, all substitutions affect the state through simple substitutions.

At the most abstract level it is obligatory to describe the static properties of a model's data by means of an "invariant" predicate. This gives rise to proof obligations relating to the consistency of the model. They are required to ensure that data properties which are claimed to be invariant are preserved by the events or operations of the model. Each refinement step is associated with a further invariant which relates the data of the more concrete model to that of the abstract model and states any additional invariant properties of the (possibly richer) concrete data model. These invariants, so-called gluing invariants are used in the formulation of the refinement proof obligations.

The goal of a B development is to obtain a proved model. Since the development process leads to a large number of proof obligations, the mastering of proof complexity is a crucial issue. Even if a proof tool is available, its effective power is limited by classical results over logical theories and we must distribute the complexity of proofs over the components of the current development, e.g. by refinement. Refinement has the potential to decrease the complexity of the proof process whilst allowing for traceability of requirements.

The AMN notation provides clauses related to structuring mechanisms in components like abstract machines, refinements or implementations. The B development process starts with basic components mainly abstract machines and is layered development; the goal is to obtain implementation components through structuring mechanisms like includes, sees, uses, extends, promotes, imports, refines. These clauses allow one to compose B components in the classical B approach and every clause leads to specific conditions for use.

The includes primitive can be used in an abstract machine or in a refinement; the included component allows the including component to modify included variables by included operations; the included invariant is preserved by the including component and is really used by the tool for deriving proofs of proof obligations of the including component. The including component can not modify included variables but it can use them in read access. No interference is possible under those constraints. The uses primitive can only appear in abstract machines and using machines have a read-only access to the used machine, which can be shared by other machines. Using machines can refer to shared variables in their invariants and data of the used machine are shared among the using machines. When a machine uses another machine, the current project must contain another machine including the using and the used machines. The refinement is related to the including machine and the using machine can not be refined. The sees primitive refers to an abstract machine imported in another branch of the tree structure of the project and sets, constants, and variables can be consulted without changed. Several machines can see the same machine. The extends primitive can only be applied to abstract machines and only one machine can extend a given machine; the extends primitive is equivalent to the includes primitive followed by the promotes primitive for every operation of the included machine.

Abstract B machines are basically used for specification of system requirements answering the “what” questions. The B refinement mechanism allows for refining the abstract machines in order to answer the “how” questions of the specification behavior [43]. The refinement might either be to refine the data representing the machine state (data refinement) or to change the procedures processing and changing the machine state (procedural refinement). Data refinement describes the design decisions that have been taken so far with regard to a particular specification through describing the way that the abstract information is to be represented by means of a linking invariant. The linking invariant relates the abstract states to the refinement states and describes how the initialization and the operations work with the new data representation [7, 43]. Procedural refinement, on the other hand, refines only the algorithmic component of an operation [45]. This is like changing the algorithm. The formal definition of refinement in B does not distinguish between procedural and data refinement.

The aim of refinement process is to produce code which meets the original abstract specification. The intermediate stages of this process are refinement machines describing the steps towards this goal, encapsulating design decisions, providing data structures and algorithms, and resolving non-determinism as appropriate. The refinement process aims to reach a point where the refinement description is detailed enough to be understood as instructions to a computer and could be translated directly to code in a suitable programming language. When a machine has reached this point, it is called an implementation machine, which cannot be refined any further [7]. The implementation machine has the same refinement relation to the machine it refines as any refinement machine, and so it will have the same proof obligations as any other refinement machine: that any step can make must be matched by the machine it refines, and that outputs must also match [43]. The implementation machine can be done only once for each development with some constraints such as that the implementation machine has no state and cannot use abstract substitutions like non-determinism choice and parallel composition.

B Models rarely need to make assumptions about the size of a system being modeled, e.g. the number of nodes in a network. This is in contrast to model checking approaches [44]. The price to pay is to face possibly complex mathematical theories and difficult proofs. The re-use of developed models and the structuring mechanisms available in B help in decreasing the complexity. Where B has been exercised on known difficult problems, the result has often been a simpler proof development than has been achieved by users of other more monolithic techniques [10].

The B method can be compared to other formal software development methods, as indicated in [44], in order to show why B has been favored in this research over other formal methods.

Like Z, B is based on the Zermelo-Fraenkel set theory; both notations share the same roots, but we can point to a number of interesting differences. Z expresses state change by use of before and after predicates, whereas the predicate transformer semantics of B allows a notation which is closer to programming. Invariants in Z are incorporated into operation descriptions and alter their meaning, whereas the invariant in B is checked against the state changes described by operations and events to ensure consistency. Finally B makes a careful distinction between the logical properties of pre-conditions and guards, which are not clearly distinguished in Z.

VDM is a method with similar objectives to classical B. Like B it uses partial functions to model data, which can lead to meaningless terms and predicates e.g. when a function is applied outside its domain. VDM uses a special three valued logic to deal with undefinedness. B retains classical two valued logic, which simplifies proof at the expense of requiring more care with undefinedness. Recent approaches to this problem will be mentioned later.

Algorithmic State Machine (ASM) and B share common objectives related to the design and the analysis of (software/hardware) systems. Both methods bridge the gap between human understanding and formulation of real-world problems and the deployment of their computer-based solutions. Each has a simple scientific foundation: B is based on set theory and ASM is based on the algebraic framework with an abstract state change mechanism. An Abstract State Machine is defined by a signature, an abstract state, a finite collection of rules and a specific rule; rules provide an operational style very useful for modeling specification and programming mechanisms. Like B, ASM includes a refinement relation for the incremental design of systems; the tool support of ASM is under development but it allows one to verify and to analyze ASMs. In applications, B seems to be more mature than ASM, even if ASM has several real successes like the validation of Java and the Java Virtual Machine.

# Chapter 3: Formal Analysis and Design for Engineering Security (FADES)

Like many high-assurance applications, there are cost and time reasons to focus the use of formal methods to the key aspects of a system. For large software applications it can be cost-prohibitive to apply formal methods and many of these large systems have relevant security concerns. So, a compromise was made and formal methods were applied to software security aspects of the system in FADES. Further, the requirements elicitation and specification process is complex and the additional complexity of formulating these requirements using a formal method is overwhelming. So, FADES started with Van Lamsweerde's near-formal, goal-directed KAOS framework for identifying, elaborating, organizing, analyzing, and specifying security requirements [1, 3, 32, 33]. KAOS is a proven semi-formal requirements elaboration framework with an underlying formal infrastructure based on first-order temporal logic. From this base, FADES transforms the KAOS requirements specification into B, preserving the security properties so carefully expressed in KAOS. Then, using the B refinement process, FADES systematically elaborate and refine the security requirements into a formal B design specification. To accommodate the fact that KAOS is not fully formal, we introduce the idea of producing a test case suite based on the requirements model to help increase confidence through verification. Further, extending KAOS with more formality in a development platform like B allows for tracing security requirements at the various steps of development; that is during both design and implementation.

## 3.1 Rationale

This section provides an argument for the rationale of the choices of KAOS and B to employ in FADES. The paradigm shift from the traditional approaches including semi-formal and formal methods to goal-orientation has led the requirements engineering research community to argue about the effectiveness and usefulness of the new paradigm. The research effort in the goal-orientation domain has resulted in the major two frameworks namely KAOS and i* that represent the current state of the art in the domain. A number of case studies have been used to demonstrate the effectiveness of the goal-oriented paradigm and illustrate its strengths and limitation. The KAOS framework has been effectively demonstrated on more than 30 industrial projects that report outstanding success stories [3]. The i* framework has been demonstrated on a number of case studies, some of them are quite large systems such as [105]. However, there is no quantitative analysis that precisely estimate the gains obtained from applying goal-oriented approaches over traditional approaches. As Van Lamsweerde and Mylopoulos, et. al. mentioned that preliminary empirical studies and their own experience with goal-orientation shows a strong potential in its application and its extensibility to formal methods [93, 98].

The above mentioned approaches that extend KAOS with extra formality to fill in the gap between requirements and later phases of development such as architecture and design show interest of the research community in the new goal-oriented paradigm. Researchers who extended KAOS either for architecture or design provided a qualitative argument for their choice of the goal-oriented method. Their argument has been qualitatively based on the prominent features of goal-orientation that enables the enhancement of the current requirements engineering practice.

Van Lamsweerde argued for the strengths of the goal-oriented KAOS framework that makes it suitable to requirements engineering since it overcomes the limitations of traditional semi-formal and formal approaches as mentioned above. There remain still some limitations to the approach in engineering requirements. The following paragraphs summarize the strengths and limitations of the KAOS framework.

The following key points about goal orientation justify the choice of the goal-oriented KAOS framework for requirements analysis in FADES [93].

- Goal-oriented modeling and specification takes a wider system engineering perspective; goals are prescriptive assertions that should hold in the system made of the software-to-be and its environment; domain properties and expectations about the environment are explicitly captured during the requirements elaboration process, in addition to the usual software requirements specifications.
- Operational requirements are derived incrementally from the higher-level system goals they “implement”.
- Goals provide the rationale for the requirements that operationalize them and, in addition, a correctness criterion for requirements completeness and pertinence [102].
- Obstacle analysis helps producing much more robust systems by systematically generating (a) potential ways in which the system might fail to meet its goals and (b) alternative ways of resolving such problems early enough during the requirements elaboration and negotiation phase.
- Alternative system proposals are explored through alternative goal refinements, responsibility assignments, obstacle resolutions and conflict resolutions.
- The goal refinement structure provides a rich way of structuring and documenting the entire requirements document.
- Different levels of formality could be offered by the framework allowing one to combine different levels of expression and reasoning: semi-formal for modeling and structuring, qualitative for selection among alternatives, and formal, when needed, for more accurate reasoning.
- Goal formalization allows requirements engineering-specific types of analysis to be carried out, like:
    - Guiding the goal refinement process and the systematic identification of objects and agents [38, 103];

- Checking the correctness of goal refinements and detecting missing goals and implicit assumptions [41];
- Guiding the identification of obstacles and their resolutions [103];
- Guiding the identification of conflicts and their resolutions[3];
- Guiding the identification and specification of operational requirements that satisfy the goals [37, 104].

Van Lamsweerde argument of the goal-orientation characteristics that offer better handling of requirements analysis is supported by Mylopoulos, et. al. argument in [98]. Mylopoulos, et. al. argued that the adoption of the goal-oriented mindset is very important during requirements analysis because it deals with non-functional requirements and relates them to functional ones [98]. Further, goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives behind a software development project. Many of the requirements engineering research community have argued that capturing these interdependencies between organizational objectives and the detailed software requirements can facilitate the tracing of the origins of requirements and can help make the requirements process more thorough, complete, and consistent [98].

Mylopoulos, et. al. have strengthened their argument in favor of the goal-oriented paradigm by preliminary empirical studies showing that goal-oriented analysis can indeed lead to a more complete requirements definition than OOA techniques. Further, the authors' own experiences in analyzing the requirements and architectural design for a large telecommunications software system confirm that goal-oriented analysis can greatly facilitate and rationalize early phases of the software design process [98].

KAOS provides a graphical notation and semi-formal interface the hides the underlying formal infrastructure in order to increase the usability and applicability of the approach and decrease its cost of employment in industrial projects [3]. Van Lamsweerde stated that one of the frequently asked questions about KAOS when considered for use in industrial projects is about the minimal project size for which a KAOS approach is cost-effective. The answer of this question is that if the project is estimated to take 20 man days, the probability of a positive return on investment is quite weak as building a requirements model is time-consuming compared to the project size in this case [51]. However, a quantitative analysis on the consulting projects in which Van Lamsweerde and his team have applied KAOS shows that a typical requirements analysis of 4 to 8 man months has been needed. The typical duration of the requirements analysis phase is 3 months and the budget needed for it represents about 10% of the total project cost [3, 51, 106]. Figure 11 extrapolates the return on investment according to project size from the Van Lamsweerde's team experience and from the following hypotheses [51]:

- The cost of one developer is 0,6 k€ per day
- The cost of one analyst is 1 k€ per day
- About one development project over 2 experiments in which the cost overruns with about 189%.
- One of the two projects that failed is due to a requirements related problem.

- The cost of an ideal requirements analysis phase is estimated at 10% of the project cost with a minimum bound fixed to 30 k€.

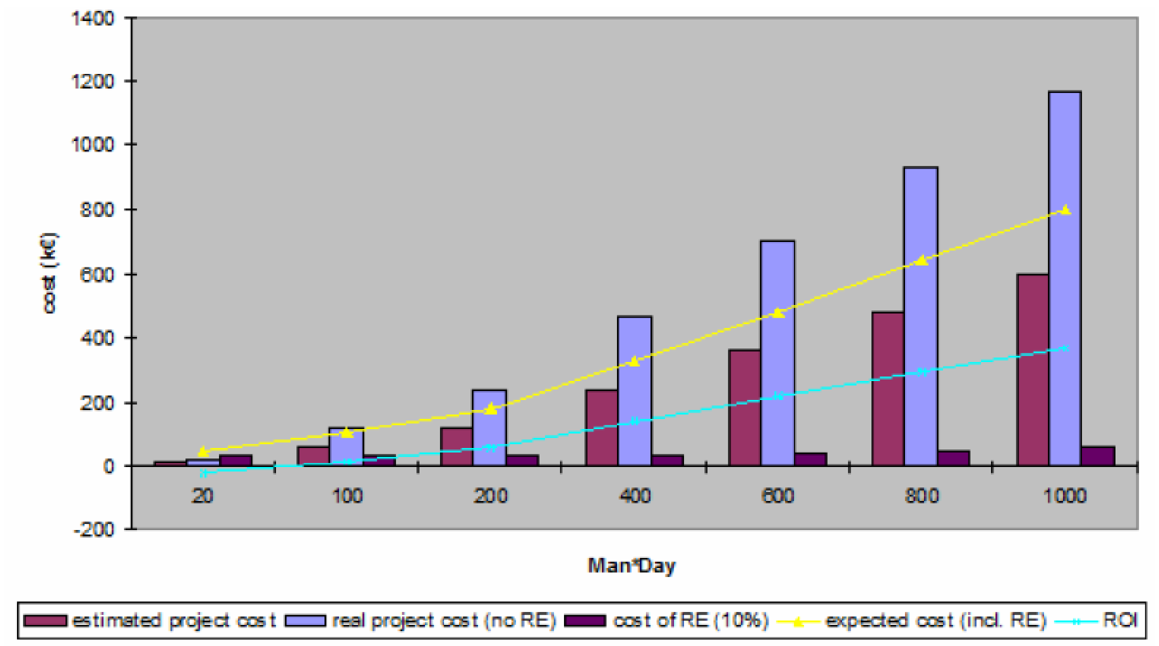


Figure 11: Return on Investment of KAOS Application According to Project Size [22]

Figure 11 shows that employing KAOS in a project for requirements analysis is cost-effective as soon as the project man power is more than 100 man days. *For medium-size and larger projects, the cost reduction is expected to be 30%* [51].

Figure 11 indicates that one of the limitations of the KAOS framework is that it is not cost-effective for projects whose size is less than 100 man days. In order to overcome this limitation, it is recommended that the company business develops a generic KAOS model once and customize it during the gap analyses made to compare the user requirements with what the package provides [97].

KAOS allows requirements engineers to use variable level of formality based on the criticality of the different parts of the software requirements. The variance in the formality level ranges from using the visual notation of KAOS to model goals to fully use first-order temporal logic to formally specify the goals, object invariants and operations pre/post conditions. Critical system aspects like security requirements might be a good candidate to the employment of high level of formality for requirements analysis. Formality variance gives more flexibility to project managers to balance their tradeoff between effort and cost of formality.

The KAOS framework is capable of constructing near-complete and near-consistent requirements models. The word "near" has been cautiously used to describe the completeness and consistency of the KAOS requirements models since one of the KAOS limitations is that it is not capable of providing formal evidence of requirements completeness and consistency. Requirements models are generally characterized by being incomplete and inconsistent by nature [97] even with formal specifications. For example, in the electronic smart card case study used to demonstrate FADES, the integrity requirement on the messages communicated in the system has been missed though the case study was specified in the Z formal language. However, the KAOS framework provides a constructive procedure justifying its ability to reach reasonable completeness and consistency levels. First, KAOS specifies 5 completeness criteria that could be used by the requirements engineer to check and ensure the model completeness as follows:

1. A goal model is complete with respect to the refinement relationship if and only if every leaf goal is an expectation, a domain property, or a requirement.
2. A goal model is complete with respect to the responsibility relationship if and only if every requirement is placed under the responsibility of one and only one agent.
3. To be complete, a process diagram must specify
    a. The agents who perform the operations.
    b. The input and output data for each operation.
4. To be complete, a process diagram must specify when operations are executed using trigger conditions.
5. All operations are to be justified by the existence of some requirements (through the use of operationalization links).

Second, KAOS provides a procedure for identifying all the possible obstacles (things that hinder goals' achievement) and conflicts (contradictions between requirements) during the obstacle analysis and resolution phase of the framework. In the security context, the KAOS security extension considers obstacles as possible security threats. The KAOS security extension provides a threat analysis mechanism to both formally or informally analyze possible threats and perform threat mitigation while building the security requirements model. Threat analysis results in the detection and resolution of security vulnerabilities very early in development [47]. Further, threat analysis allows for the anticipation of application-specific attack scenarios such as attacks on a web-based banking application that might result in disclosure of sensitive information about bank accounts or in credulous money transfer. My experience with using the obstacle analysis feature of the KAOS security extension has emphasized its effectiveness both in the demonstration of FADES with the case studies and in the empirical study hold to validate FADES. For example, when applying FADES to the electronic smart card system as described in section 4.2 and comparing it to the Z specification and implementation of the same system, the obstacle analysis feature was effective in detecting threats to messages integrity. The Z specification, on the other hand, was not able to detect the same threats. This demonstrates that introducing KAOS for requirements analysis prior to formal design in FADES is not only enhancing the cost-effectiveness of applying formal methods, but also provides means to early detect security breaches and enhance the overall security of the software.

Third, Van Lamsweerde indicated that spending reasonable effort on the construction of the requirements model enhances its completeness and consistency [3]. The more feedback sessions the analyst holds with the stakeholders to get their answers on open questions or to verify the completeness and consistency of the requirements model, the better the results obtained at the design and implementation phases. The KAOS goal graph is a structure that could be communicated with the stakeholders who often have no technical background. Providing a structure that could be understood and communicated to stakeholders allows requirements engineers to get more thorough feedback from stakeholders to enhance the completeness and consistency of the requirements model.

The choice of B as a formal development platform for elaborating security design specifications assists in preserving security properties of requirements when design specifications are being derived. B has the notion of model refinement that allows for building a detailed model of design from an abstract model of requirements while preserving the security properties of the requirements model. The refinement mechanism in B provides a means for documenting design decisions and building forward traceability links from requirements to design. Hence, the links between artifacts are clear enough to provide traceability information that serves software maintenance activities, which might be performed after the software is fully developed. The use of a software model that stores design decisions and traceability links significantly improves the accuracy and completeness of impact analysis that is concerned with identifying the impact of a given change on the software product [50]. Moreover, B is based on set mathematics with the ability to use standard first-order predicate logic facilitating the integration with the KAOS security requirements model that is based on first-order temporal logic. Further, B is a mature formal method that has been successfully employed in industrial projects for long time. The availability of good tool support for the B development platform strengths the practicality and applicability of FADES.

Employing formal methods in FADES provides a reasonable approach to the challenge of developing secure software products with formal evidence of correctness [49]. Recognizing that formal methods reduce security risks but entails more cost, we justify this cost by applying FADES only to security, which is a critical aspect of the system. Further, software systems that are evaluated for security at the Common Criteria (CC) EAL (Evaluation Assurance Level) 5, 6 and 7 need formal evidence assuring the security of the software. This makes software products developed using FADES securely compliant with CC higher levels.

## 3.2 FADES

FADES is a step towards the development of highly secure software. In a nutshell, FADES is a requirements-driven software engineering approach that derives design specifications from a set of security requirements modeled using KAOS security extension framework (described in section 2.1). The approach provides a secure software engineering methodology that effectively integrates KAOS security extension, which is characterized by the ability to formally build a complete, consistent and clear requirements model with the B method, which provides formal guarantees for the correctness of the system development. Our research showed that KAOS is promising in that it could be extended with an extra step of formality in order to fully implement security requirements while preserving the security properties specified in the requirements model. Moreover, extending KAOS with more formality in a development framework like B allows for tracing requirements at the various steps of development; that is, during both design and implementation.

FADES starts with a set of security requirements that are being elaborated with the KAOS security extension to build a goal graph for the security requirements and derive the operations that achieve the goals. FADES makes use of the analytical capabilities provided with the goal graph to achieve two objectives. The first objective is to transform the KAOS operations derived to achieve the goals to B using means of the transformation scheme described below in order to construct an abstract B model that describes the initial system state and its expected security behavior. The initial B machine is further refined using the B refinement mechanism to add more details while building the security design specifications. The second objective is to derive acceptance test cases from the goal graph outlining the different scenarios of security behavior that should be met by the derived B design and implementation specifications. This means that the goal graph is used to derive the initial abstract B model that will be further refined for design and implementation and to derive means to verify that the derived design and implementation meet the security requirements objectives through the acceptance test cases. The completeness and consistency of the derived implementation specifications are a function of the successful verification of the implementation against the acceptance test cases.

The extra verification step that FADES provides through the derivation of test cases allows for detecting inconsistencies that might have been in the requirements model or in the process of design and implementation derivation. The test cases guarantee the same level of completeness and consistency of the requirements model since they are derived using a depth first search algorithm. The algorithm traverses the goal graph to generate sequence of calls to operations in the correct order that matches the semantics of the high level goals in the goal graph. The test cases provide means to detect possible security hazards that might result from inconsistencies either in the requirements model or in design. FADES is illustrated in Figure 12.

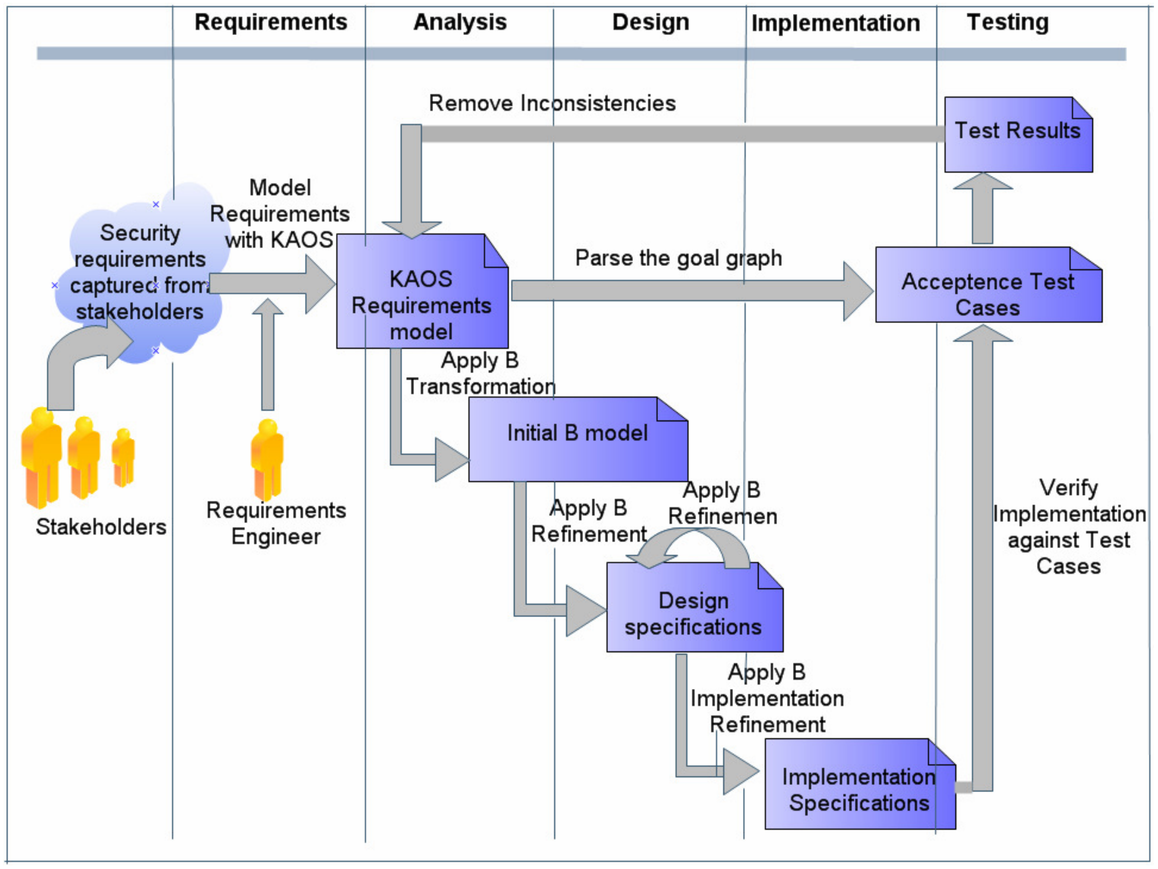


**Figure 12: FADES**

FADES provides means for transforming the security requirements model built with KAOS to an equivalent one in B using some transformation rules described in section 3.2.1. The B model that has been transformed from KAOS representing security requirements is then refined using non-trivial B refinements that generate design specifications conforming to the security requirements. Each B refinement step prior to the implementation refinement reflects some design decision(s) added by the refining B machine to the refined B machine until implementation is obtained. The B formal method exhibits one of its prominent features of model refinement in allowing us to make our security design decisions with a proof of correctness that these decisions do not violate the constraints specified in the KAOS requirements model (operations pre/post conditions and entities invariants). This means that the development platform itself provides constructs to reduce risks of introducing errors in development. After applying a number of refinement steps to the initial B model, we apply an implementation refinement step, which is a special refinement step in B as indicated in section 2.3.

FADES allows for deriving one artifact, which is design from another artifact, which is a requirements model using formal representation in B. The refinement mechanism in B provides means for documenting design decisions and building forward traceability links from requirements to design. Hence, the links between artifacts are clear enough to provide traceability information that serves the purposes of software maintenance activities that might be performed after the software is fully developed. The use of a software model that stores design decisions and traceability links significantly improves the accuracy and completeness of impact analysis that is concerned with identifying the impact of a given change on the software product [50].

The derived implementation specifications are then verified against the test cases that have been drawn from the KAOS requirements model. Our results have shown that the major two sources for problems encountered in implementation are inconsistencies either in the requirements model or in the design decisions made during the B refinement steps. The derived test cases are capable of detecting some inconsistencies in both the requirements model and design. Further, the ratio between the numbers of successful test cases and failed test cases can be used as exit criteria for security assurance and this is evidenced in the Common Criteria Evaluation Assessment Levels (EAL) 5, 6 and 7. The feedback loop established from the test cases results to the requirements model as indicated in the Figure 12 allows for detecting possible security hazards a priori to deploying the software system into production at which time "real" security threats might be encountered.

FADES is characterized with some features that make it more attractive to apply over other similar formal approaches. These features are either inherent from the underlying approaches employed in FADES, namely KAOS and B or new in FADES. FADES addresses security-specific elements of software to bridge the gap between security requirements and their realization in design and implementation. Up to our knowledge, FADES is the first approach to address the gap between requirements and design for security requirements specifically. FADES is promising in being ready for wide applicability thanks to the strong tool support provided by the underlying technologies of KAOS and B. KAOS has a commercial tool called Objectiver [51] that has been commercially used in a number of successful industrial projects. B has been in the industrial market for a while with its most two famous commercial products namely AtelierB and the B-Toolkit [52, 53]. The availability of strong tool support strengths the practicality and applicability of FADES.

FADES takes KAOS with a further step of formality to derive design and implementation through transforming the KAOS requirements model to B. The initial B model obtained from the automatic transformation enforce the same security constraints specified in the KAOS requirements model modeled in the form of pre conditions of the B operations that model the KAOS operations and the B machine invariants that model the KAOS entities invariants. The definition of these constraints in the initial B model forces the preservation of these constraints at the later B refinements steps that add more details to the initial B model to commit design decisions. The proof obligation facility provided with B method and that could be automatically generated using one of the commercial B tools allows software security developers to discharge the generated proof obligations to ensure correctness of development. Discharging the proof obligations formally proves that a refinement step in B does not violate the constraints of the more abstract B model being refined. FADES provides an extra verification step to show the maintenance of security properties specified in the requirements model in the derived implementation specifications. This is achieved when deriving a set of test cases that are generated from traversing the KAOS goal graph. The test cases provide security developers with means to assure a reasonable level of completeness and consistency of their implementation with respect to the requirements model. Finally, one of the key merits of employing formal methods in FADES is the availability of sufficient traceability information that links requirements to design decisions giving better opportunities for more accurate and less vulnerable handling of changes to security specifications as illustrated in section 3.2.3.

### 3.2.1 Transforming KAOS to B

The KAOS requirements model is represented in the form of a directed-acyclic graph rooted at the very high level goal of the system and structured in multiple levels. Goals at each level refine the goals at the higher level. An example of a goal graph for the security requirements of the spy network system presented in section 4.1 is illustrated as an example in Figure 13. The diamond shapes represent security goals while the rectangles with circular corners represent agents responsible to achieve leaf goal requirements.

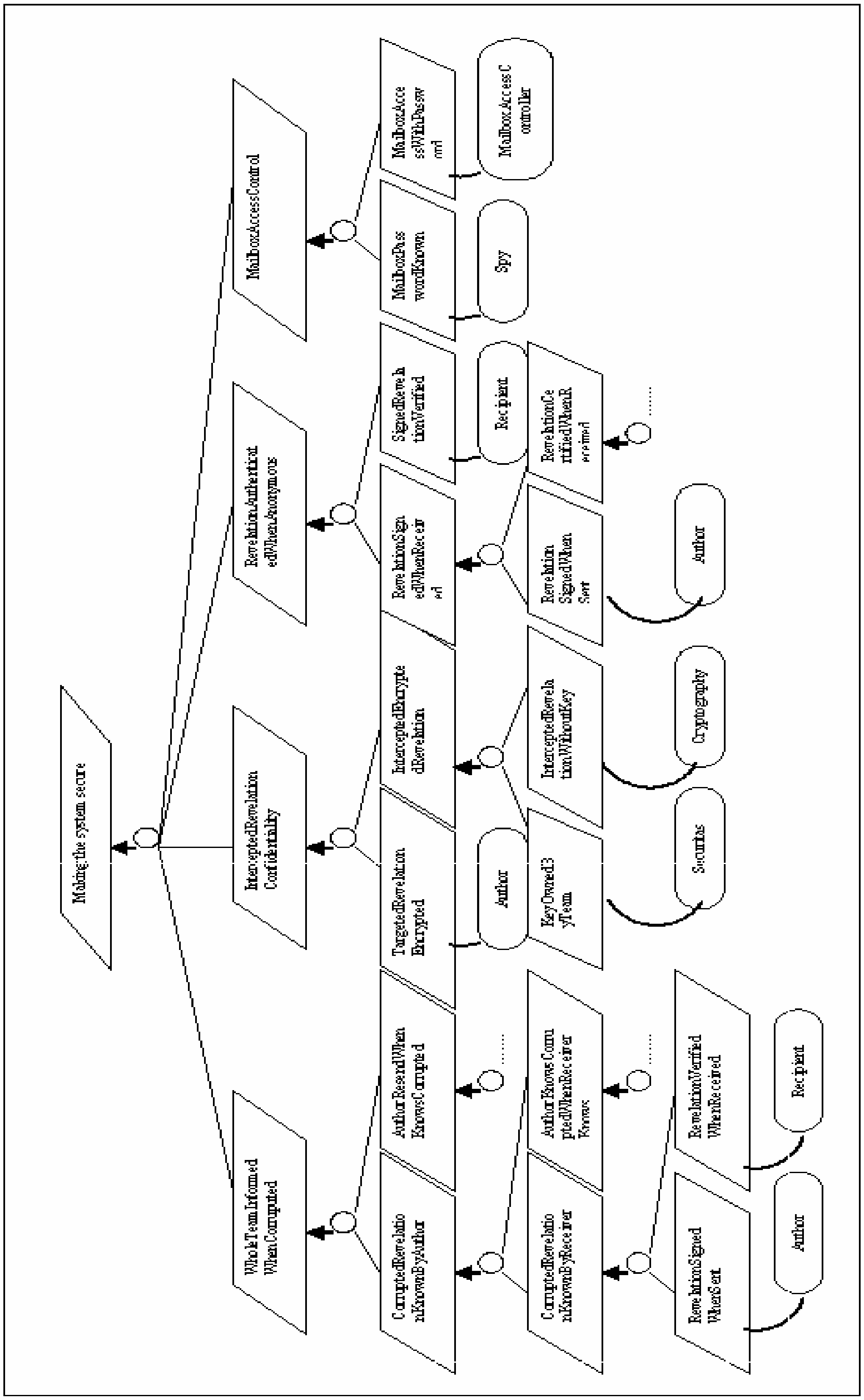

Making the system secure
WholeTeamInformedWhenCorrupted
InterceptedRevelationConfidentiality
RevelationAuthenticatedWhenAnonymous
MailboxAccessControl
CorruptedRevelationKnownByAuthor
AuthorResendWhenKnowsCorrupted
TargetedRevelationEncrypted
InterceptedEncryptedRevelation
RevelationSignedWhenReceived
SignedRevelationVerified
MailboxPasswordKnown
MailboxAccessWithPassword
Author
Recipient
Spy
MailboxAccessController
CorruptedRevelationKnownByReceiver
AuthorKnowsCorruptedWhenReceiverKnows
KeyOwnedByTeam
InterceptedRevelationWithoutKey
RevelationSignedWhenSent
RevelationCertifiedWhenReceived
Securitas
Cryptography
Author
RevelationSignedWhenSent
RevelationVerifiedWhenReceived
Author
Recipient
......

**Figure 13: Spy Network KAOS Goal Graph**

The whole system is represented either as a single abstract B machine or multiple abstract B machines related to each other based on the size of the system. Each KAOS entity (passive object) is represented as a B machine included or seen by the system machine(s) and encapsulating its KAOS attributes and operations manipulating the attributes as follows:

- The entity attributes are variables in the equivalent B machine representing the state of the object.
- The B machine invariant is composed of the types of the attributes' variables (might be primitive types or types from other KAOS entities) and the domain invariant of the KAOS object presented in first-order predicate logic.
- Each B machine representing an entity includes a set representing all its instances because B is not an object oriented language, rather it is instance-based. Entity attributes are represented as relations between the set of instances and the attribute types. This representation of the instances and their attributes allows for the use of the set arithmetic capabilities of B.

Each KAOS operation is represented as a B operation in the system machine and uses the KAOS entities either as parameters or return values. KAOS operations pre/post and trigger conditions are treated in the transformation as follows:

- Pre-condition are directly mapped to a B precondition for the B operation since both KAOS and B preconditions are written in first order-predicate logic.
- Post-condition has no equivalent construct in B. The operation specification and refinement should be responsible for enforcing the KAOS operation post-condition. This means that the post condition of the KAOS operation should be used in the operation specification, which is not part of the transformation. The transformation is only limited to building the requirements model that represent the KAOS model. It is the responsibility of the designer to fill in the body of each operation while taking into consideration the KAOS operation post-condition.
- Trigger-condition has no direct mapping to the trigger condition in B. The trigger condition in KAOS is used to model the sequence of operation calls and the conditions that would lead to calling the operation. The trigger condition could be forced by the agents who are controlling the system runtime execution. Further, it is the responsibility of the agent calling the operations to prepare all the operations trigger conditions in a true state for the operation to be called.

Agents (active objects) are not directly transformed to KAOS since they are responsible for the runtime behavior of the system. Agents' behavior is modeled through the transformed KAOS operations. The acceptance test cases derived from the KAOS goal graph as indicated in section 3.2.2 simulate the agent behavior in executing the KAOS operations that realized the requirements goals.

KAOS goals are not transformed to B since their semantics is encapsulated in the KAOS operations that realize the leaf goals. The achievement of leaf goals through KAOS operations implies the achievement of the higher-level goals through the KAOS AND/OR refinement process. The KAOS goals and their sequence of refinement are used to derive the acceptance test cases that are used to increase our confidence in the derived implementation specifications to be consistent and complete with respect to the requirements model as illustrated in section 3.2.2. Further, KAOS goals in the goal graph are used to provide links for requirement traceability. Design decisions can be easily linked to higher-level goals using the goal graph structure that indicates which goals are realized by which operation (s). Traceability information plays a crucial role in accurately specifying the impact of applying a change to security specifications.

KAOS domain properties are captured in B as invariants to the B machines that model the KAOS objects and as pre/post conditions to the operations that model KAOS operations. Objects invariants as well as operations pre conditions are preserved during the B refinement steps meaning that the domain properties are maintained throughout the software development lifecycle using the proposed approach.

KAOS scenarios are not considered in the transformation scheme since scenarios are used as a vehicle security requirements engineer uses to assure the customer that all the security requirements are well understood and captured. The primary objective of the transformation from KAOS to B is to develop security design specifications that need to detail the implementation plan of the security requirements rather than focusing, as in the requirements analysis phase, on providing evidence that the customer can understand such as scenarios.

The rules of the transformation scheme from KAOS to B are summarized in Table 2.

| KAOS Constructs | B Constructs |
|---|---|
| KAOS object | B abstract machine |
| KAOS object attribute | B machine variable |
| KAOS object operation | B machine operation |
| KAOS object invariant | B machine invariant |
| KAOS operation | B operation |
| KAOS operation pre-condition | B operation pre-condition |
| KAOS agent | Agent behavior is modeled through the transformed KAOS operations |
| KAOS domain properties | Invariants of the transformed KAOS objects and pre-conditions of B operations |

**Table 2: Transformation Rules from KAOS to B**

The transformation step of FADES from KAOS to B has been automated using the Goal Graph Analyzer tool as indicated in section 3.3. The Goal Graph Analyzer tool is a Java tool that parses the KAOS goal graph represented in an XML format to generate the initial B model using the transformation rules of Table 2 and the acceptance test suite. The generated B model needs human in the loop to complement the B operations equivalent to KAOS operations and KAOS objects' operations with body specifications describing the abstract behavior of the operation to realize security requirements. The behavior need to be complemented since it is not specified in the KAOS model that only specifies what operations are needed to achieve security requirements rather than how these operations will achieve the requirements.

To prove equivalency between the KAOS model and the initial B model resulting from the transformation, we have investigated two options; a formal mathematical proof and the derivation of acceptance test cases. We have consulted Nakagawa and his team who developed the formal specification generator for KAOS [4] that generates a VDM++ model from KAOS specifications about the feasibility of a formal mathematical proof of equivalency between KAOS and B. Nakagawa and his team have reported their experience in trying to develop the mathematical proof as infeasible since the KAOS model is a requirements model involving undefined parameters making the proof very hard to develop within a reasonable timeframe. Moreover, the automation of the transformation reduces its error-proneness since automation rigidly applies the transformation rules. Further, the fact that both KAOS and B employ first-order predicate logic to express system constraints and conditions facilitates the transformation and reduces the probability of an equivalency gap between the KAOS model and the transformed B model.

The acceptance test case option has been more feasible to realize any errors that might result from the transformation scheme. The KAOS goal graph is used to derive the initial abstract B model that will be further refined for design and implementation and to derive means to verify that the derived design and implementation meet the security requirements objectives through the acceptance test cases. Our results have shown that the major two sources of problems encountered in implementation are inconsistencies either in the requirements model or in the design decisions made during the B refinement steps, but not in the transformation rules.

### 3.2.2 Derivation of Acceptance Test Cases

FADES provides an extra verification step to show the maintenance of security properties specified in the requirements model in the derived implementation specifications. FADES derives a suite of acceptance test cases through traversing the KAOS goal graph. The test cases provide security developers with means to assure a reasonable level of completeness and consistency of their implementation with respect to the requirements model.

The KAOS goal graph is rooted at the very high level goal of the system while leaf goals at the very bottom of the goal graph represent requirements. Each requirement is realized with an operation performed by an agent in the software-to-be. This means that the sequence of operations that need to be performed in order to execute a goal at a middle level of the goal graph can be identified using a depth first search (DFS) algorithm for the subgraph rooted at this goal. We have chosen a DFS algorithm with backtracking capabilities in order to eliminate the unnecessary paths from the search process [54, 55]. Consider part of the goal graph concerned with the money exchange of electronic transactions for an electronic purse system illustrated in Figure 14. To test the achievement of the goal ExchangeMoney, the sequence of operation calls need to be generated using the DFS algorithm for the subgraph rooted at the ExchangeMoney node.

Following the DFS algorithm illustrated in Figure 15 and its Java implementation in Figure 16, we obtain the ExchangeMoney test case illustrated in Figure 17. The DFS algorithm searches the subgraph rooted at ExchangeMoney node to visit the operation nodes in the following sequence: DecryptMsg, SendMoneyValue, EncryptMsg, ReceiveMoneyValue

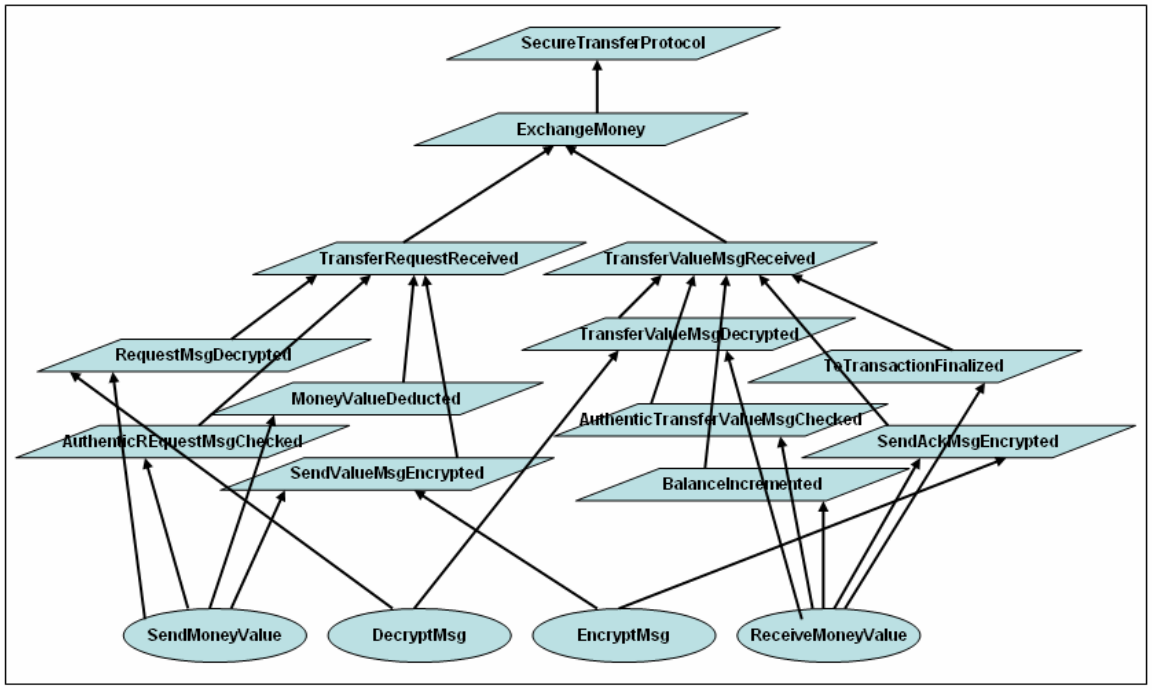
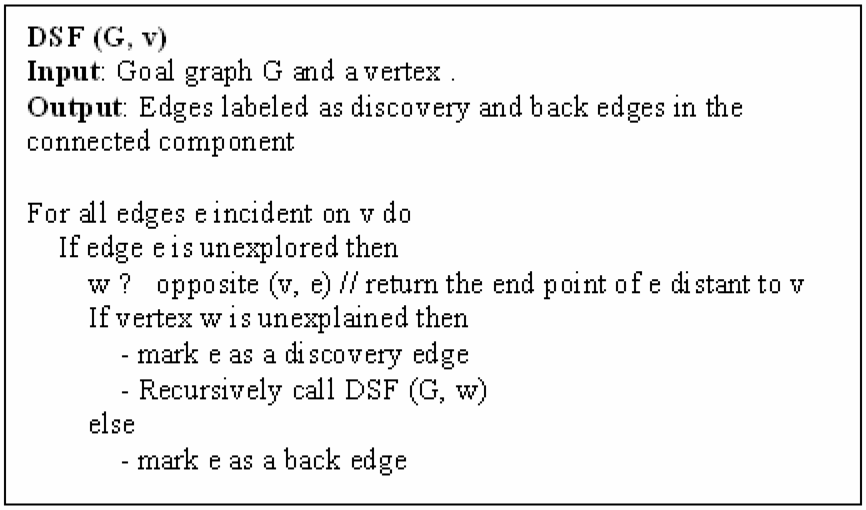


**Figure 14: KAOS Goal Graph for Money Exchange of the Electronic Purse System**

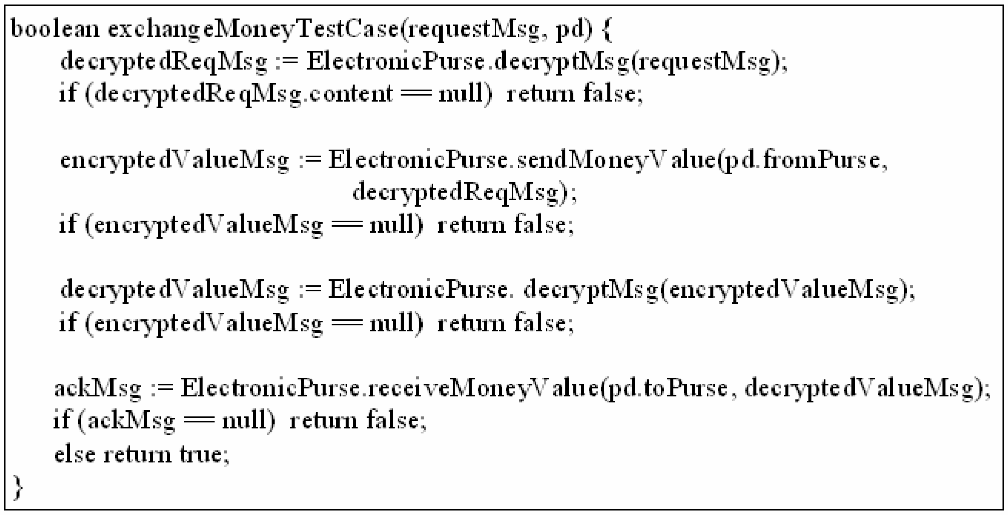


```
DSF (G, v)
Input: Goal graph G and a vertex .
Output: Edges labeled as discovery and back edges in the
connected component

For all edges e incident on v do
   If edge e is unexplored then
      w ?   opposite (v, e) // return the end point of e distant to v
      If vertex w is unexplained then
         - mark e as a discovery edge
         - Recursively call DSF (G, w)
      else
         - mark e as a back edge
```

**Figure 15: The Depth First Search Algorithm Used to Generate Acceptance Test Cases**

```
//the arrays PreOrder and PostOrder have to be preallocated before calling the DFS algorithm.
import java.io.*;
public class search {
   static public int DFS(Graph G, int v, int[] PreOrder, int[] PostOrder) {

         int n = G.order();
         for (int i=0; i<n; i++) PreOrder[i] = PostOrder[i] = 0;
         // returns number of nodes reached
         countPair cnt = new countPair();
         doDFS(G, v, PreOrder, PostOrder, cnt);
         return PostOrder[v];
   }
   public static void main(String argv[]) {

     try {

         BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
         while(true) {
            Graph G = new Graph(input);
            int n=G.order();
                 if ( n == 0 ) break;
            System.out.print(G.toStringAdjLists());
            int preOrder[] = new int[n];
            int postOrder[] = new int[n];
            GraphAlgs.BFS(G,0,preOrder);
            System.out.print(&#;`¨BFS (levelorder): &#,`¨);
            for (int i=0; i<n; i++) System.out.print(preOrder[i] + &#;`¨ &#;`¨);

            System.out.print(&#;`¨&#,\n&#;`¨);
            GraphAlgs.DFS(G,0,preOrder,postOrder);
            System.out.print(&#;`¨DFS (preorder): &#;`¨);
            for (int i=0; i<n; i++) System.out.print(preOrder[i] + &#;`¨ &#;`¨);

            System.out.print(&#;`¨&#,\n&#;`¨);
            System.out.print(&#;`¨DFS (postorder): &#;`¨);
            for (int i=0; i<n; i++) System.out.print(postOrder[i] + &#;`¨ &#;`¨);
            System.out.print(&#;`¨&#,\n&#;`¨);
        }

   } catch ( Exception e ) { System.err.println(&#;`¨Exception: &#;`¨+e); }

  } // main

} // class search
```

**Figure 16: Java Code for the DFS Algorithm in Figure 15**



```
boolean exchangeMoneyTestCase(requestMsg, pd) {
    decryptedReqMsg := ElectronicPurse.decryptMsg(requestMsg);
    if (decryptedReqMsg.content == null)  return false;

    encryptedValueMsg := ElectronicPurse.sendMoneyValue(pd.fromPurse,
                                   decryptedReqMsg);
    if (encryptedValueMsg == null)  return false;

    decryptedValueMsg := ElectronicPurse. decryptMsg(encryptedValueMsg);
    if (encryptedValueMsg == null)  return false;

    ackMsg := ElectronicPurse.receiveMoneyValue(pd.toPurse, decryptedValueMsg);
    if (ackMsg == null)  return false;
    else return true;
}
```

**Figure 17: Generated Test Case for MoneyExchange Goal**

The generated test cases might need to be augmented by the software developers or testers to add more business domain-specific assertions. More meaningful messages might be displayed with the assertions to assist developers reasoning about failures of test cases and identifying sources of errors. Our results have shown that errors in the requirements model and/or design errors are the major sources of problems identified using the acceptance test cases.

Blackburn proposed test coverage percentage and test results (number of successful and failed test cases) as measures for checking the quality of the software product [28]. The same measures could be used with FADES to evaluate the quality of the derived implementation, which is a function of the completeness and consistency of the requirements model. The acceptance test cases guarantee the same level of completeness and consistency of the requirements model since they are directly derived from that model using the DFS algorithm. Therefore, the acceptance test cases are capable of identifying inconsistencies in the derived implementation with respect to the requirements model. The capability of the acceptance test cases to identify problems in the derived implementation is demonstrated in the chapter outlining case studies.

The derived test cases are automatically generated from the KAOS requirements model using the Goal Graph Analyzer tool that traverses the model to generate both the initial B model and the acceptance test cases. The acceptance test cases could be used as exit criteria for stakeholders to verify compliance between requirements and implementation. Further, the employment of formal methods in the derivation of both the acceptance test cases and the implementation allows for the assurance of target of evaluation constructed using FADES at Common Criteria EAL 5, 6, and 7 that mandate formal evidence of development.

Clarke et. al. categorized testing into three categories namely unit testing, integration testing and acceptance testing [56]. Unit testing is concerned with assuring functionality on the module level while integration testing is concerned with assuring functionality when the various system modules are integrated. Clarke outlined selection criteria for paths that might reveal faults in software programs both for unit and integration testing [56]. However, up to our knowledge, there is no research work done in the area of identifying selection criteria for acceptance test cases that might have given more weight to some test cases over others in being able to reveal faults and inconsistencies.

## 3.3 FADES Tool Support

FADES offers a wide range of applicability thanks to the integrated tool support provided by the underlying technologies of KAOS and B. KAOS has a commercial tool called Objectiver [51] that has been commercially used in a number of successful industrial projects. Van Lamsweerde has reported success stories of applying KAOS using the Objectiver toolbox to more than twenty enterprise software projects [3]. B has been in industry for a while with its most two famous commercial products namely AtelierB [52] and the B-Toolkit [53]. The availability of strong tool support strengths the practicality and applicability of FADES and allows for its demonstration on large case studies.

The KAOS requirements model created with Objectiver is exported in XML format. We have built a Java tool called the Goal Graph Analyzer to parse the XML of the KAOS model and extract the necessary information required to build the initial B model and the acceptance test cases. The Analyzer produces two artifacts; 1) the initial B model representing the requirements model, and 2) the acceptance test cases. The software engineer needs to augment the B model with abstract specification of the B operations' body while the tester augments the generated test cases with some assertions and messages to enhance the usability of the test cases to produce more meaningful results. This inserts a human-in-the-loop to produce more useful output. The B-Toolkit is then used refine the abstract B machines to make design decisions and finally derive implementation specifications. Proof obligations are generated with each refinement step and the required proofs are discharged by the software developer in order to prove correctness of development. Figure 18 illustrates the tool support of FADES.

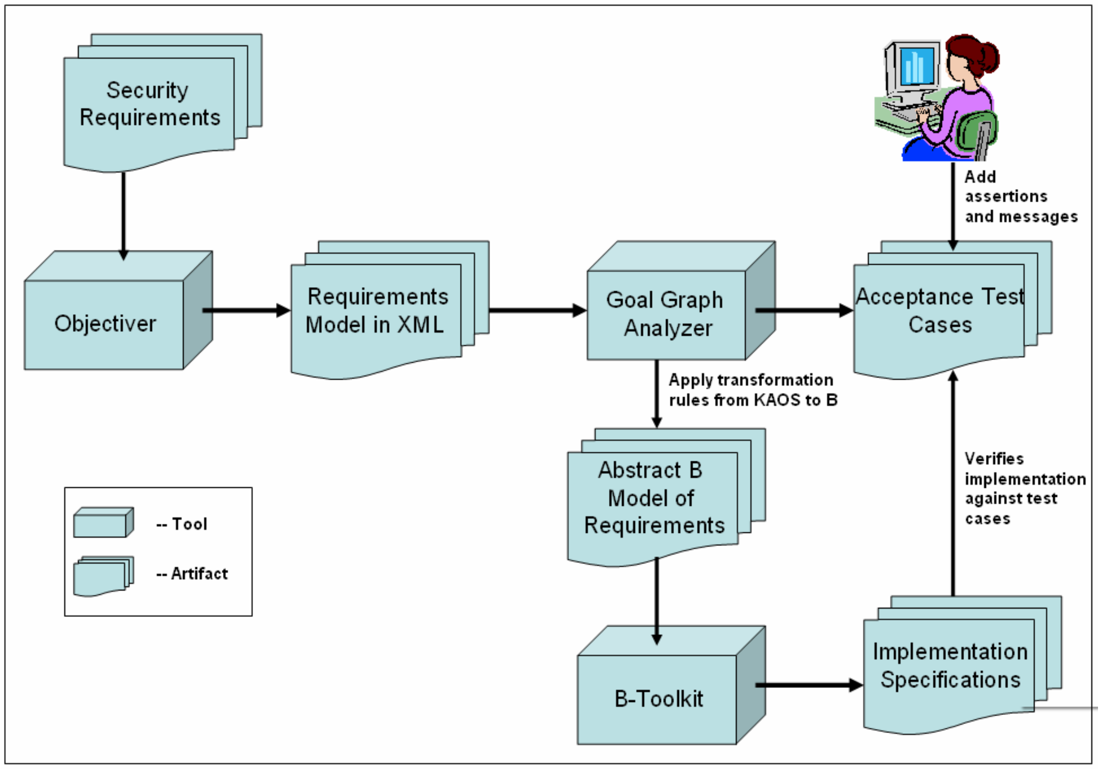

Security Requirements
Objectiver
Requirements Model in XML
Goal Graph Analyzer
Acceptance Test Cases
Add assertions and messages
Apply transformation rules from KAOS to B
Abstract B Model of Requirements
Verifies implementation against test cases
-- Tool
-- Artifact
B-Toolkit
Implementation Specifications

**Figure 18: FADES Tool Support**

# Chapter 4: FADES Demonstration with Case Studies

FADES from a logical perspective shows promise – employing KAOS, a semi-formal requirements approach, to capture, organize, and elaborate on security requirements provides an advantage over a fully formal approach as its goal-directed nature allows enough flexibility while managing to characterize and preserve key security properties that can then be transformed into a proven B representations for further elaboration and refinement at the design and implementation levels. However, demonstrating its effectiveness in light of existing approaches is necessary to understand the implications of this research. While the limitations of a Ph.D. research timeline and finances prevent an exhaustive treatment with a statistically meaningful set of experiments, this research does develop some valuable empirical evidence for FADES through two case studies derived from real projects showing the applicability and potential strengths of the approach. The first case study is a spy network system originally designed by Fontaine [40] to exercise KAOS on secure systems. The second case study is an electronic smart card presented by Stepney [6] to investigate the Z formal method applied to this class of trusted software.

This chapter outlines the security requirements of each of the two case studies and demonstrates how these requirements are modeled with KAOS and transformed to B for further refinement to derive implementation specifications in B. Section 4.1 describes the application of FADES to the spy network system with the details provided in Appendix A. Section 4.2 describes the application of FADES to the electronic smart card system. Section 4.3 summarizes the results obtained by Stepney [6] when applying the Z formal language to the electronic smart card system. Further, it qualitatively compares Stepney's results to our results of applying FADES to the same system and discusses the strengths and limitations of FADES.

## 4.1 The Spy Network Case Study

While there are a number of technical papers on security patterns [88, 89], there are few canonical examples from which to formulate a reasonable comparator. According to Fontaine, the security literature does not provide security requirements benchmarks [40]. Rather it has some small examples, which are associated to security models. Unlike many case studies in the security literature, the spy network case study is of a reasonable size; small enough to be manageable and large enough to be convincing.  The spy network case study has been derived from two real case studies:

- The British National Health Service (NHS). The main goal of the system is to protect medical records from illegitimate access to a centralized database.
- The eBay on-line auction web site. It is an example of a typical e-business application with a range of constrains about distributed user behaviors.

The spy network system represents a sample of the category of communication systems that share a common set of security requirements. This assists in verifying the applicability of FADES to communication systems that exhibit high security demands.

### 4.1.1 Case Study Preliminary Problem Statement

The spy network application is aimed at broadcasting secret revelations into a network of spies around the world. Spies are collaborating in teams that achieve a mission each. Each team has a boss. The big boss supervises all missions, allocates spies to missions, and appoints bosses to teams. Spies collect revelations about the enemy and target them to other members of the team working on the mission. The spy who collects a revelation is its author. A spy can be reallocated on another mission, meaning that he goes to another team.

Team members should be provided with an uncorrupted copy of the revelation within a certain timeframe. Only spies who are currently allocated to a mission are allowed to know revelations about that mission. Some spies may be malicious spies. Therefore, we need to be sure of the author of a revelation. Workload balancing should be achieved between different spies in the system; therefore we will avoid centralized operations as much as possible.

Each spy subscribes to his local service provider for a mailbox in which all his incoming email arrives. Therefore, each spy has a different mail server. Mailboxes are identified by their address. Spy mailbox addresses are very confidential and should not be written down in an insecure location. For security reasons, spies change their mailbox every month. Some spies are old friends and often write to each other outside of the missions duties. Therefore, they memorize others spies' mailbox addresses without having to write them down. Only the owner of a mailbox should be able to access it.

Each spy has at least a few spy friends. Friendship is assumed to last forever. Friends can be in different teams. The author of a revelation is not necessarily the team boss.

Revelations are contained in messages that are sent through the email transfer system (asynchronous messages). The email transfer system cannot attack actively, although it could fail to deliver messages or be subject to passive eavesdropping.

A revelation is written by an author and read by one or several recipients. A message is sent by a sender and received by a receiver. A message containing a revelation is not necessarily sent directly to the recipients. It could be sent to an intermediate receiver who will send another message containing the same revelation to the recipient, or to another intermediate spy. Revelations are persistent objects, whereas messages are temporary objects that cease to exist upon reception, after their content has been processed. We could think of messages as envelops and of revelations as their content.

When a spy collects a new revelation, he sends a message to the team relay. The relay then sends messages to all other members of the team. This assumes that the team relay does know every team member's identity. The team boss appoints the team relay. There is one single relay in each team at a time and every spy knows that fact.

## 4.1.2 Elaborating Security Requirements with KAOS

This subsection outlines the elaboration of security requirements for the spy network system with KAOS. The security goals resulting from the elaboration of security requirements are global in the sense that a particular agent cannot enforce them; instead, they apply to the whole system. We will elaborate generic security requirements that are typical in the security domain. This means that they are applicable by analogy to other security domain case studies.

**Goal** *Maintain* [Security]
**InformalDef** The system is secure
**InstanceOf** SecurityGoal

This high level security goal is refined using the traditional classification [57, 58].

**Goal** *Maintain* [Integrity]
**InformalDef** Information is guarded against unauthorised update or tampering
**InstanceOf** SecurityGoal
**Refines** Security

**Goal** *Maintain* [Confidentiality]
**InformalDef** Information is guarded against unauthorised disclosure
**InstanceOf** SecurityGoal
**Refines** Security

**Goal** *Maintain* [Authentication]
**InformalDef** Agents are really who they claim to be
**InstanceOf** SecurityGoal
**Refines** Security

**Goal** *Achieve* [Availability]
**InformalDef** Information is guarded against interruption of service
**InstanceOf** SecurityGoal
**Refines** Security

**Goal** *Maintain* [AccessControl]
**InformalDef** Access control prevents unauthorised access to protected information
**InstanceOf** SecurityGoal
**Refines** Security

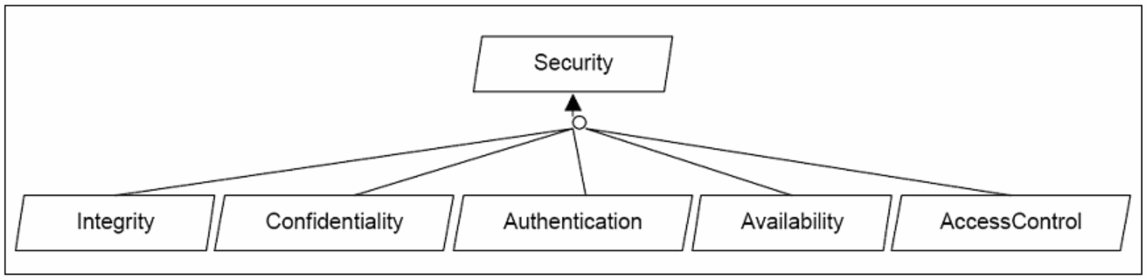


**Figure 19: Refinement of security goals**

This refinement is not complete in that all offspring goals do not necessarily imply the father goal because security properties of a system are of multiple natures. We have chosen these five subgoals because they are known to be the most frequent aspects of a secure system.

In order to formally express these security goals, we need to specify a security model for the system, which can be either generic or specific. The literature on security defines generic security models that have to be instantiated to particular systems such as the Bell-LaPadula or Biba models [40]. In order to apply a generic security model to a system, the security requirements of this system need to fit into the model. The Bell-LaPadula or Biba models are appropriate for hierarchical systems like military systems. In these systems, uses at the same level of the hierarchy have the same rights, which means that privileges are granted to a user class rather than specific users. Many distributed systems do not necessarily fit into this model.

In the context of the spy network case study, we will not use a generic security model, but instead we will instantiate security goals with domain specific patterns related to the domain.

All security goals are expressed in terms of the stakeholder's language. For instance, the concept of a Message, which is specific to a particular design is not used. This reflects the fact that these are high level goals and are applicable to any alternative design chosen for the system. In other words, by expressing these security goals, the system is not constrained to fit into a particular security model.

### 4.1.2.1 Integrity Goals

The generic pattern for integrity goals can be formalized:

**Goal** *Maintain* [ObjectCopyAccuracy]
    **InformalDef** Each copy of an object is accurate
    **InstanceOf** IntegrityGoal, AccuracyGoal
    **FormalDef** ∀ ob1, ob2 : Object
        CopyOf(ob1, ob2) ⇒ ob1.Content = ob2.Content

The following heuristic is proposed to instantiate this goal:
For every object copy in the system, the ObjectCopyAccuracy goal should be instantiated. This requires finding out which agent owns the master copy of the object. All other instances of the object will be considered as copies from this master object.

In the context of the spy network case study, integrity means that every copy of a revelation should be identical to the original revelation written by the author.

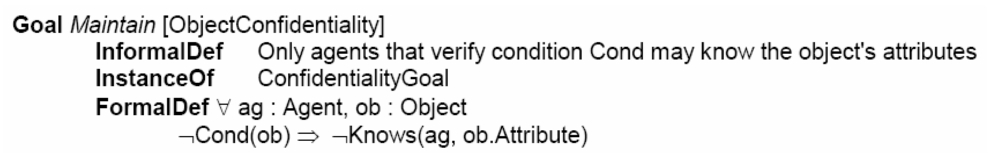
**Goal** *Maintain* [RevelationIntegrity]
    **InformalDef** A copy of a revelation is identical to the original
    **InstanceOf** IntegrityGoal, AccuracyGoal
    **FormalDef** ∀ sp1, sp2 : Spy, re1, re2 : Revelation
        Collecting(sp1, re1) ∧ Owning(sp2, re2) ∧ CopyOf(re1, re2)
            ⇒ re1.Content = re2.Content

The goal RevelationIntegrity is too ideal and cannot be assigned to any agent because the author does not know which revelation content is actually received by the recipient, and conversely the recipient does not know which revelation content the author has sent. Fortunately, this goal is a non-functional one, which means that it is not supposed to be assigned to a particular agent, and it is global in the system. Maintaining this goal is achieved through the introduction of new functional goals that enforce such non-functional goal.

If every agent assigned to a goal successfully achieves his goal, the integrity of revelations is guaranteed since integrity is implicitly stated in the functional goals. However, stating an explicit goal like InformationIntegrity allows for covering a wider range of agent behaviors in case an agent fails to achieve his goal. We are then able to elaborate strategies that will be refined into strengthened design. This leads to the achievement of system goals even in case of agent failures, which means more robust design. For instance, we could use digital signatures, in which case the recipient is able to verify whether the revelation is intact and is from the purported author. In this operationalization scheme, additional goals are needed to notify the author in case a recipient has received a corrupted copy of a revelation.

### 4.1.2.2 Confidentiality Goals

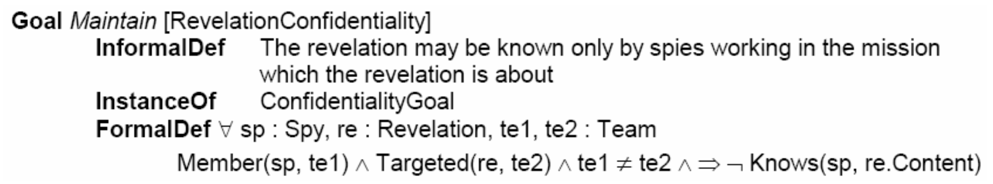
The generic pattern for confidentiality goal can be formalized as follows:



**Goal** *Maintain* [ObjectConfidentiality]
**InformalDef** Only agents that verify condition Cond may know the object's attributes
**InstanceOf** ConfidentialityGoal
**FormalDef** $\forall$ ag : Agent, ob : Object
$\neg$Cond(ob) $\Rightarrow$ $\neg$Knows(ag, ob.Attribute)

The following heuristic is proposed to instantiate this goal:
For every attribute of an object, express what necessary condition for an agent to know such attribute is. For attributes that do not have such condition, no confidentiality goal is necessary. These conditions depend on domain knowledge.
In the context of the spy network case study, confidentiality is refined into two subgoals:

1. Confidentiality of revelations means that a revelation may be known only by spies working in the mission. Therefore, the generic goal can be instantiated as follows:



**Goal** *Maintain* [RevelationConfidentiality]
**InformalDef** The revelation may be known only by spies working in the mission which the revelation is about
**InstanceOf** ConfidentialityGoal
**FormalDef** $\forall$ sp : Spy, re : Revelation, te1, te2 : Team
Member(sp, te1) $\wedge$ Targeted(re, te2) $\wedge$ te1 $\neq$ te2 $\wedge$ $\Rightarrow$ $\neg$ Knows(sp, re.Content)

This goal is non-functional goal that can neither be assigned to the recipient nor the sender. The recipient might want to know the revelation although he should refrain himself from doing so and the sender is not able to control that a wrong target intercepts the message. Therefore, this goal needs to be enforces by further functional goals. As we will see later, secret keys allows for expressing who is authorized to know a revelation.

2. Confidentiality of mailboxes means that a spy mailbox address should known only by a friend who can memorize it.

   **Goal** *Maintain* [MailboxConfidentiality]
   **InformalDef** Only friends should know each other's mailbox address
   **InstanceOf** ConfidentialityGoal
   **FormalDef** $\forall$ sp1, sp2 : Spy, ma2 : Mailbox
   $\neg$ Friend(sp1, sp2) $\Rightarrow \neg$ Knows(sp1, Subscribed[sp2, ma2])

   This goal is too ideal because friendship cannot be controlled by any single agent. It can only be controlled by both parties involved. So, we are likely to weaken this goal in different designs.

#### 4.1.2.3 Authentication Goals

The generic pattern for authentication goals can be formalized as follows:

**Goal** Maintain[ObjectAuthentication]
**InformalDef** Agents can verify who has authorship on an object
**InstanceOf** AuthenticationGoal
**FormalDef** $\forall$ ag1, ag2 : Agent, ob : Object
AuthorOf(ag1, ob) $\Rightarrow$ Knows(ag2, AuthorOf[ag1, ob])

The following heuristic is proposed to instantiate this goal:
For every object, agents should be able to verify its author.

In the context of the spy network case study, authentication means that every revelation is attributable to an author and that the purported author of the revelation is correct. The generic goal can be instantiated as follows:

**Goal** *Maintain* [RevelationAuthentication]
**InformalDef** Agents can verify who has authorship on a revelation
**InstanceOf** AuthenticationGoal
**FormalDef** $\forall$ sp1, sp2 : Spy, re: Revelation
AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\Rightarrow$ Knows(sp2, AuthorOf[sp1, re])

The relationship AuthorOf is equivalent to the relationship Collecting. The spy who collects the revelation is the author. The author mentioned on a revelation can be a forged name, so we will need to verify that the purported author is the real author.

#### 4.1.2.4 Availability Goals

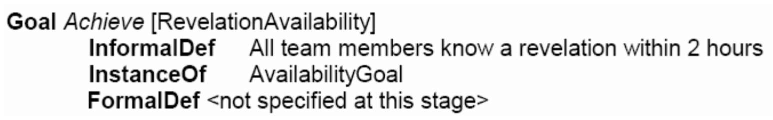
The generic pattern for availability goals can be formalized as follows:

**Goal** Achieve [ResourceAvailableForRequest]
**InformalDef** All agents requests get the resource(s) needed to satisfy the request
**InstanceOf** AvailabilityGoal
**FormalDef** $\forall$ ag : Agent, re: Request
RequestIssued(ag, re) $\Rightarrow \Diamond_{<TF}$ ($\exists$ res : Resource) Available(res) $\wedge$ Sufficient(res, re)

The following heuristic is proposed to instantiate this goal:

For every Achieve goal, determine which resource(s) needs to be available for the goal achievement.

In the context of the spy network case study, availability means that revelations are known within a certain timeframe by all other team members. The generic goal can be instantiated as follows:

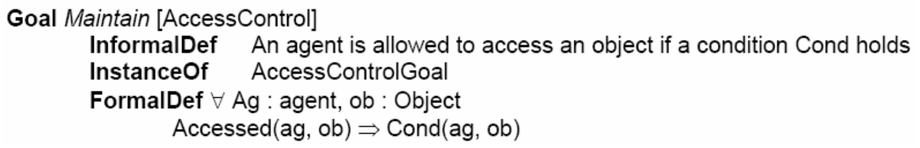


**Goal** *Achieve* [RevelationAvailability]
    **InformalDef** All team members know a revelation within 2 hours
    **InstanceOf** AvailabilityGoal
    **FormalDef** <not specified at this stage>

This goal is a quantitative one in which the critical aspect is the timeframe. In this formulation, it is assumed that revelations can be owned within 2 hours. The constant 2 is used as a parameter representing the expected mean time for a revelation transmission.

### 4.1.2.5 Access Control Goals

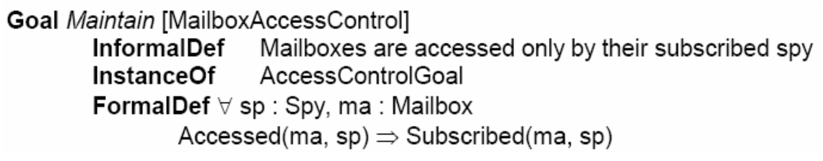
The generic pattern for access control goals can be formalized as follows:



**Goal** *Maintain* [AccessControl]
    **InformalDef** An agent is allowed to access an object if a condition Cond holds
    **InstanceOf** AccessControlGoal
    **FormalDef** $\forall$ Ag : agent, ob : Object
        Accessed(ag, ob) $\Rightarrow$ Cond(ag, ob)

The following heuristic is proposed to instantiate this goal:
For every object, express what the necessary condition for an agent to access it is. For objects that do not have such condition, no access control goal is necessary. These conditions depend on domain knowledge.

In the context of the spy network case study, access control means that a mailbox should be accessed only by its subscribed spy



**Goal** *Maintain* [MailboxAccessControl]
    **InformalDef** Mailboxes are accessed only by their subscribed spy
    **InstanceOf** AccessControlGoal
    **FormalDef** $\forall$ sp : Spy, ma : Mailbox
        Accessed(ma, sp) $\Rightarrow$ Subscribed(ma, sp)

## 4.1.3 Analysis and Resolution of Obstacles and Conflicts for Security Goals

In this subsection, the security goals elaborated in the previous section are refined and analyzed by finding conflicts and obstacles to security, that is, potential attacks. This subsection focuses on finding obstacles to security goals only taking obstacles to the RevelationIntegrity goal as an example while obstacles to the rest of security goals are detailed in appendix A. Resolution strategies to potential security attacks are proposed.

### 4.1.3.1 Generating Obstacle to the Goal RevelationIntegrity

The goal RevelationForwardedFromRelay is assigned to the relay. In case the relay agent fails to achieve this goal and modifies the revelation content (maliciously or not), the goal RevelationIntegrity becomes violated as well.

Negating the goal RevelationIntegrity gives the following:

**Obstacle** RecipientHasCorruptedRevelation
**InformalDef** Recipient owns a corrupted copy of the revelation
**FormalDef** $\lozenge\, \exists$ sp1, sp2 : Spy, re1, re2 : Revelation
$\text{Collecting(sp1, re1)} \wedge \text{Owning(sp2, re2)} \wedge \text{CopyOf(re1, re2)} \wedge \text{re1.Content} \neq \text{re2.Content}$

The following object model increment is required:

**Relationship** Corrupted
**InformalDef** A revelation copy is not intact with respect to the original revelation
**Links** Revelation {**Role** Caracterised, **Card** 0:N}
**DomInvar**
$\forall$ re1, re2 : Revelation
$\text{Corrupted(re1)} \vee \text{Corrupted(re2)} \Rightarrow \text{CopyOf(re1, re2)} \wedge \text{re1.Content} \neq \text{re2.Content}$
$\forall$ sp1, sp2, sp3 : Spy, re1, re2 : Revelation, me1, me2 : Message
$\text{Receiving(sp2, me, sp1)} \wedge \text{Sending(sp2, me, sp3)} \wedge \text{About(me1, re1)}$
$\wedge\, \text{About(me2, re2)} \wedge \text{CopyOf(re1, re2)} \wedge \text{re1.Content} \neq \text{re2.Content} \Rightarrow \text{Corrupted(re2)}$
$\forall$ sp : Spy, re : Revelation
$\text{Collecting(sp, re)} \Rightarrow \neg\text{Corrupted(re)}$

Regressing through the domain properties, we get:

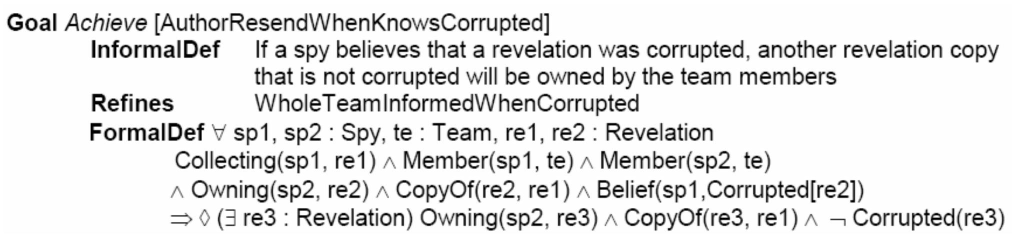
**Obstacle** RecipientHasCorruptedRevelation
**InformalDef** An agent has sent a different revelation than the one he has received
**FormalDef** $\lozenge\, \exists$ sp1, sp2 : Spy, re1, re2 : Revelation
$\text{Collecting(sp1, re1)} \wedge \text{Owning(sp2, re2)} \wedge$
$[\text{Receiving(sp2, me, sp1)} \wedge \text{Sending(sp2, me, sp3)} \wedge \text{About(me1, re1)}$
$\wedge\, \text{About(me2, re2)} \wedge \text{CopyOf(re1, re2)} \wedge \text{re1.Content} \neq \text{re2.Content}] \vee \text{Corrupted(re1)}$

Because of the third domain property, we know that the revelation has been corrupted by sp2 (the relay) rather than by sp1 (the author). If the relay changes the content of the revelation, the RevelationIntegrity goal is violated. A strong mitigation would be achieved through the following goal:

**Goal** *Achieve* [WholeTeamInformedWhenCorrupted]
**InformalDef** If a spy collects a revelation, eventually all team members will own it even if a previous copy of the revelation was corrupted
**FormalDef** $\forall$ sp1, sp2 : Spy, re1, re2 : Revelation, te : Team
$\text{Collecting(sp1, re1)} \wedge \text{Member(sp1, te)} \wedge \text{Member(sp2, te)}$
$\wedge\, \text{Owning(sp2, re2)} \wedge \text{CopyOf(re2, re1)} \wedge \text{Corrupted(re2)}$
$\Rightarrow \lozenge\, (\exists\, \text{re3 : Revelation})\ \text{Owning(sp2, re3)} \wedge \text{CopyOf(re3, re1)} \wedge \neg\, \text{Corrupted(re3)}$

This goal refined as follows:

**Goal** *Maintain* [CorruptedRevelationKnownByAuthor]
**InformalDef** The author knows when another spy owns a corrupted copy of his revelation
**Refines** WholeTeamInformedWhenCorrupted
**FormalDef** $\forall$ sp1, sp2 : Spy, re1, re2 : Revelation
$\text{Collecting(sp1, re1)} \wedge \text{Owning(sp2, re2)} \wedge \text{CopyOf(re2, re1)} \wedge \text{Corrupted(re2)}$
$\Rightarrow \text{Knows(sp1, Corrupted[re2])}$


**Goal** *Achieve* [AuthorResendWhenKnowsCorrupted]

**InformalDef** If a spy believes that a revelation was corrupted, another revelation copy that is not corrupted will be owned by the team members

**Refines** WholeTeamInformedWhenCorrupted

**FormalDef** $\forall$ sp1, sp2 : Spy, te : Team, re1, re2 : Revelation

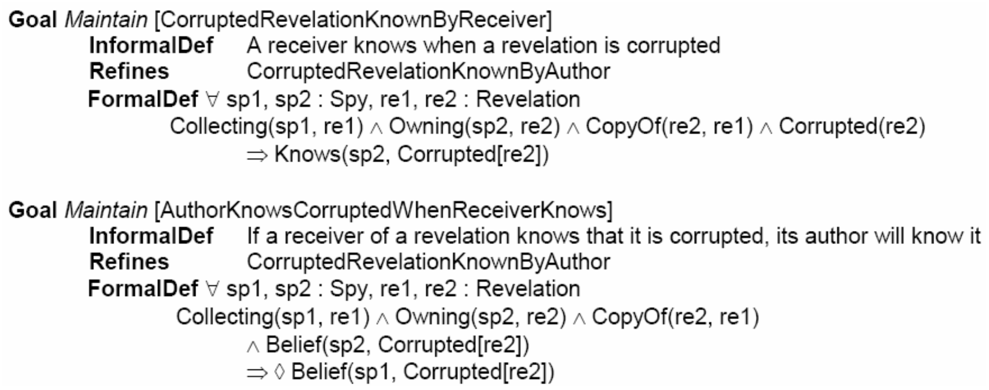
Collecting(sp1, re1) $\wedge$ Member(sp1, te) $\wedge$ Member(sp2, te)

$\wedge$ Owning(sp2, re2) $\wedge$ CopyOf(re2, re1) $\wedge$ Belief(sp1,Corrupted[re2])

$\Rightarrow \Diamond$ ($\exists$ re3 : Revelation) Owning(sp2, re3) $\wedge$ CopyOf(re3, re1) $\wedge$ $\neg$ Corrupted(re3)

The goal AuthorResendWhenKnowsCorrupted needs to be refined the same way as the goal WholeTeamInformed. In this case, the revelation author could send the revelation again hoping that it was only a temporary error and that the revelation will not get corrupted this time.

The goal CorruptedRevelationKnownByAuthor needs to be refined as follows:


**Goal** *Maintain* [CorruptedRevelationKnownByReceiver]

**InformalDef** A receiver knows when a revelation is corrupted

**Refines** CorruptedRevelationKnownByAuthor

**FormalDef** $\forall$ sp1, sp2 : Spy, re1, re2 : Revelation

Collecting(sp1, re1) $\wedge$ Owning(sp2, re2) $\wedge$ CopyOf(re2, re1) $\wedge$ Corrupted(re2)

$\Rightarrow$ Knows(sp2, Corrupted[re2])

**Goal** *Maintain* [AuthorKnowsCorruptedWhenReceiverKnows]

**InformalDef** If a receiver of a revelation knows that it is corrupted, its author will know it

**Refines** CorruptedRevelationKnownByAuthor

**FormalDef** $\forall$ sp1, sp2 : Spy, re1, re2 : Revelation

Collecting(sp1, re1) $\wedge$ Owning(sp2, re2) $\wedge$ CopyOf(re2, re1)

$\wedge$ Belief(sp2, Corrupted[re2])

$\Rightarrow \Diamond$ Belief(sp1, Corrupted[re2])

The goal AuthorKnowsCorruptedWhenReceiverKnows will be refined using notifications in a similar way as the goal WholeTeamInformed as shown below. The goal CorruptedRevelationKnownByReceiver will be refined using digital signatures as indicated in the coming subsection.

#### 4.1.3.2 Resolving Obstacles to the Goal RevelationIntegrity

Van Lamsweerde defined some patterns to resolve obstacles early in requirements and accommodate these solutions in the requirements model [3]. In the security context, obstacles represent security threats to the system that need to be resolved using security patterns [89]. For example, data integrity is normally preserved using digital signatures, so a way to resolve the obstacles to the RevelationIntegrity goal is to employ digital signatures in the goal operationalization of the RevelationIntegrity goal. The early accommodation of this solution during requirements analysis allows for reflecting the impact of this solution on the rest of the requirements that have dependency on this RevelationIntegrity goal. The early accommodation of security solutions in FADES enhances the quality of the resulting security as evidenced in empirical study hold to validate the approach as described in Chapter 5.

The RevelationIntegrity goal is operationalized such that each revelation has a signature that depends on the revelation text and the author's identity. With a public key scheme, there are different keys for signing (private key) and verifying (public key). Verification allows the receiver to check whether the message is intact. Each spy can have a list of public keys of every other spy including those not in his team.

When he receives a revelation from a spy, he can verify that the revelation is from the purported author if the key is accurately associated with the author. If a spy has several public/private keys, they need to be identified. For simplicity, let's assume in the coming illustration that each spy has only one key pair.

The signature of the message could be used to deduce that the revelation is not corrupted with respect to the public key used to verify it as defined in the goal RevelationVerifiedWhenReceived. In case the verification indicates that the revelation is corrupted, it could also be the case that it is actually the signature that is corrupted and the revelation is correct.

The goal CorruptedRevelationKnownByReceiver can be refined into the goals RevelationSignedWhenSent and RevelationVerifiedWhenReceived that mandate the signing of the revelation at the sender side and the verification of that signature at the receiver end as follows:

**Goal** *Maintain* [RevelationSignedWhenSent]
**InformalDef** Revelations are signed with the author's private key
**Refines** CorruptedRevelationKnownByReceiver
**FormalDef** $\forall$ sp1, sp2 : Spy, re1, re2 : Revelation
Collecting(sp1, re1) $\wedge$ Owning(sp2, re2) $\wedge$ CopyOf(re2, re1)
$\Rightarrow$ ($\exists$ prk : PrivateKey, puk : PublicKey, si : Signature)
Signed(re1, si, prk) $\wedge$ Owning(sp2, si)
$\wedge$ KeyPair(puk, prk) $\wedge$ Owning(sp2, puk)

**Goal** *Maintain* [RevelationVerifiedWhenReceived]
**InformalDef** The receiver knows whether the revelation is genuine with signature verification
**Refines** CorruptedRevelationKnownByReceiver
**FormalDef** $\forall$ sp1, sp2 : Spy, me1, me2 : Message, re1, re2 : Revelation, prk : PrivateKey, puk : PublicKey
Collecting(sp1, re1) $\wedge$ Owning(sp2, re2) $\wedge$ CopyOf(re2, re1)
$\wedge$ Signed(re1, si, prk) $\wedge$ Owning(sp2, si) $\wedge$ $\neg$ Verified(re2, si, puk).Genuine
$\wedge$ KeyPair(puk, prk) $\wedge$ Owning(sp2, puk)
$\Rightarrow$ Knows(sp2, Corrupted[re2])

The goals RevelationSignedWhenSent and RevelationVerifiedWhenReceived are assigned to the sender and the receiver agents respectively. The operations SignRevelation and VerifyRevelation operationalize the two goals as indicated below.

The goal graph showing the refinement of the resolution of the RevelationIntegrity obstacles is illustrated in Figure 20:

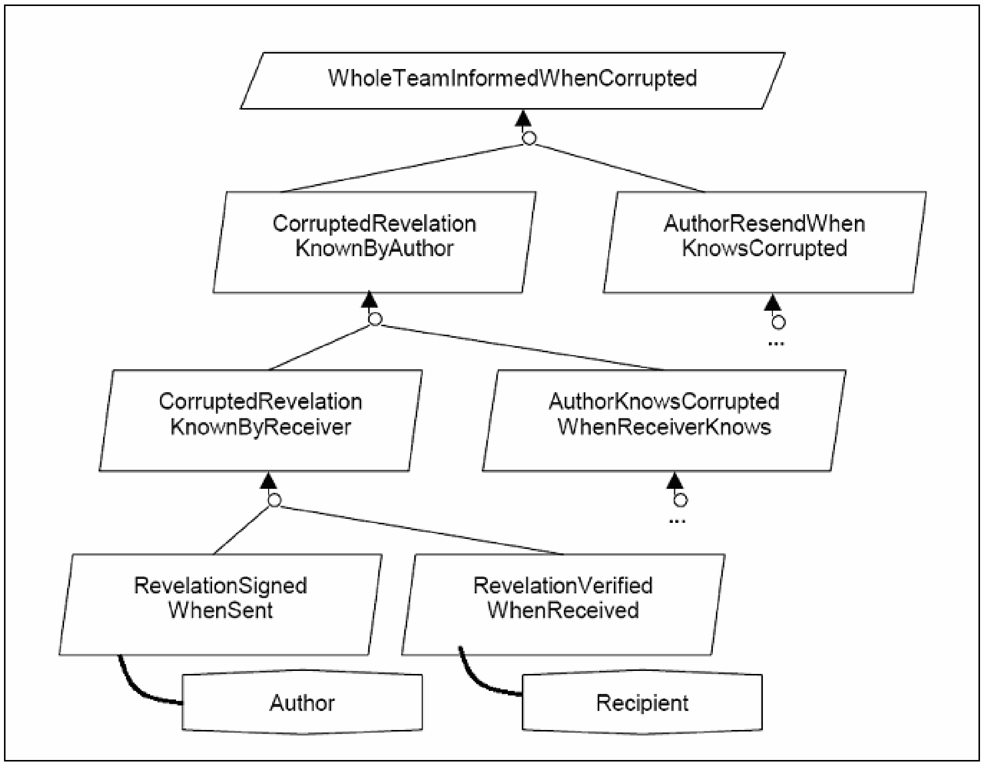
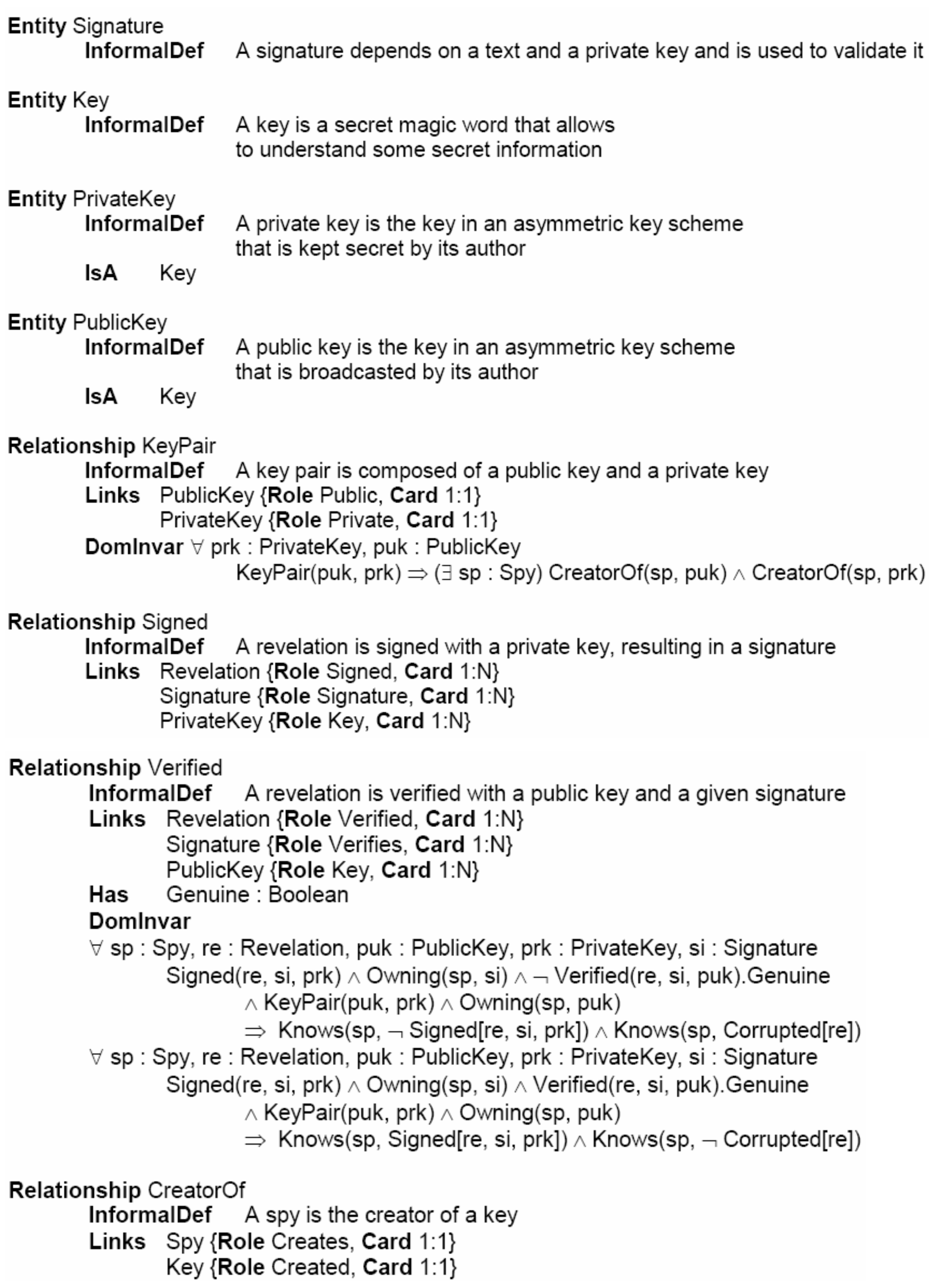

WholeTeamInformedWhenCorrupted
CorruptedRevelation KnownByAuthor
AuthorResendWhen KnowsCorrupted
...
CorruptedRevelation KnownByReceiver
AuthorKnowsCorrupted WhenReceiverKnows
...
RevelationSigned WhenSent
RevelationVerified WhenReceived
Author
Recipient

**Figure 20: Refinement of the Integrity obstacle resolution**

The above two goals introduced to resolve the obstacles of the RevelationIntegrity goal yield some increments in the object model in order to accommodate the digital signature solution. Assuming that public key infrastructure is employed to carry out digital signatures, the entities of a PrivateKey, PublicKey and KeyPair are needed as well as entities modeling the signature itself and the relationships Signed and Verified. The object model increments are defined as follows:



**Entity** Signature
  **InformalDef** A signature depends on a text and a private key and is used to validate it

**Entity** Key
  **InformalDef** A key is a secret magic word that allows
  to understand some secret information

**Entity** PrivateKey
  **InformalDef** A private key is the key in an asymmetric key scheme
  that is kept secret by its author
  **IsA** Key

**Entity** PublicKey
  **InformalDef** A public key is the key in an asymmetric key scheme
  that is broadcasted by its author
  **IsA** Key

**Relationship** KeyPair
  **InformalDef** A key pair is composed of a public key and a private key
  **Links** PublicKey {**Role** Public, **Card** 1:1}
  PrivateKey {**Role** Private, **Card** 1:1}
  **DomInvar** $\forall$ prk : PrivateKey, puk : PublicKey
  KeyPair(puk, prk) $\Rightarrow$ ($\exists$ sp : Spy) CreatorOf(sp, puk) $\wedge$ CreatorOf(sp, prk)

**Relationship** Signed
  **InformalDef** A revelation is signed with a private key, resulting in a signature
  **Links** Revelation {**Role** Signed, **Card** 1:N}
  Signature {**Role** Signature, **Card** 1:N}
  PrivateKey {**Role** Key, **Card** 1:N}

**Relationship** Verified
  **InformalDef** A revelation is verified with a public key and a given signature
  **Links** Revelation {**Role** Verified, **Card** 1:N}
  Signature {**Role** Verifies, **Card** 1:N}
  PublicKey {**Role** Key, **Card** 1:N}
  **Has** Genuine : Boolean
  **DomInvar**
  $\forall$ sp : Spy, re : Revelation, puk : PublicKey, prk : PrivateKey, si : Signature
  Signed(re, si, prk) $\wedge$ Owning(sp, si) $\wedge$ $\neg$ Verified(re, si, puk).Genuine
  $\wedge$ KeyPair(puk, prk) $\wedge$ Owning(sp, puk)
  $\Rightarrow$ Knows(sp, $\neg$ Signed[re, si, prk]) $\wedge$ Knows(sp, Corrupted[re])
  $\forall$ sp : Spy, re : Revelation, puk : PublicKey, prk : PrivateKey, si : Signature
  Signed(re, si, prk) $\wedge$ Owning(sp, si) $\wedge$ Verified(re, si, puk).Genuine
  $\wedge$ KeyPair(puk, prk) $\wedge$ Owning(sp, puk)
  $\Rightarrow$ Knows(sp, Signed[re, si, prk]) $\wedge$ Knows(sp, $\neg$ Corrupted[re])

**Relationship** CreatorOf
  **InformalDef** A spy is the creator of a key
  **Links** Spy {**Role** Creates, **Card** 1:1}
  Key {**Role** Created, **Card** 1:1}

In the goal CorruptedRevelationKnownByReceiver, it is stated that the author's public key is owned by the receivers of the revelation. It means that keys have to be distributed like revelations with the difference that public keys last for the entire life of a spy. The key distribution will also yield a goal for their broadcasting similar to the goal WholeTeamInformed:

**Goal** *Maintain* [BroadcastPublicKey]
  **InformalDef** Spies in the same teams should own each other's key
  **FormalDef** $\forall$ sp1, sp2 : Spy, puk : PublicKey, te : Team
    CreatorOf(sp1, puk) $\wedge$ Member(sp1, te) $\wedge$ Member(sp2, te) $\Rightarrow$ Owning(sp2, puk)

A new type of message is introduced, PublicKeyNotif to be sent by a spy to inform other people of his public key.

**Entity** PublicKeyNotif
  **InformalDef** Announcement from a spy giving out his public key
  **IsA** Message
  **Has** Spy : Spy
    PubKey : PublicKey

Because a private key identifies a spy, it needs to be owned only by its creator:

**Goal** *Maintain* [SafePrivateKey]
  **InformalDef** A spy owns a private key only if he is the owner of this key
  **FormalDef** $\forall$ sp : Spy, prk : PrivateKey
    Owning(sp, prk) $\Rightarrow$ CreatorOf(sp, prk)

The above object model increment has propagated the impact of introducing digital signature as a solution to the obstacles of the RevelationIntegrity goal. The rest of the security goals that relate to RevelationIntegrity feel the impact of the digital signature solution through the update of the object model that mediates the interaction among goals.

A graphical summary of all security goals after applying obstacle analysis is illustrated in Figure 21.

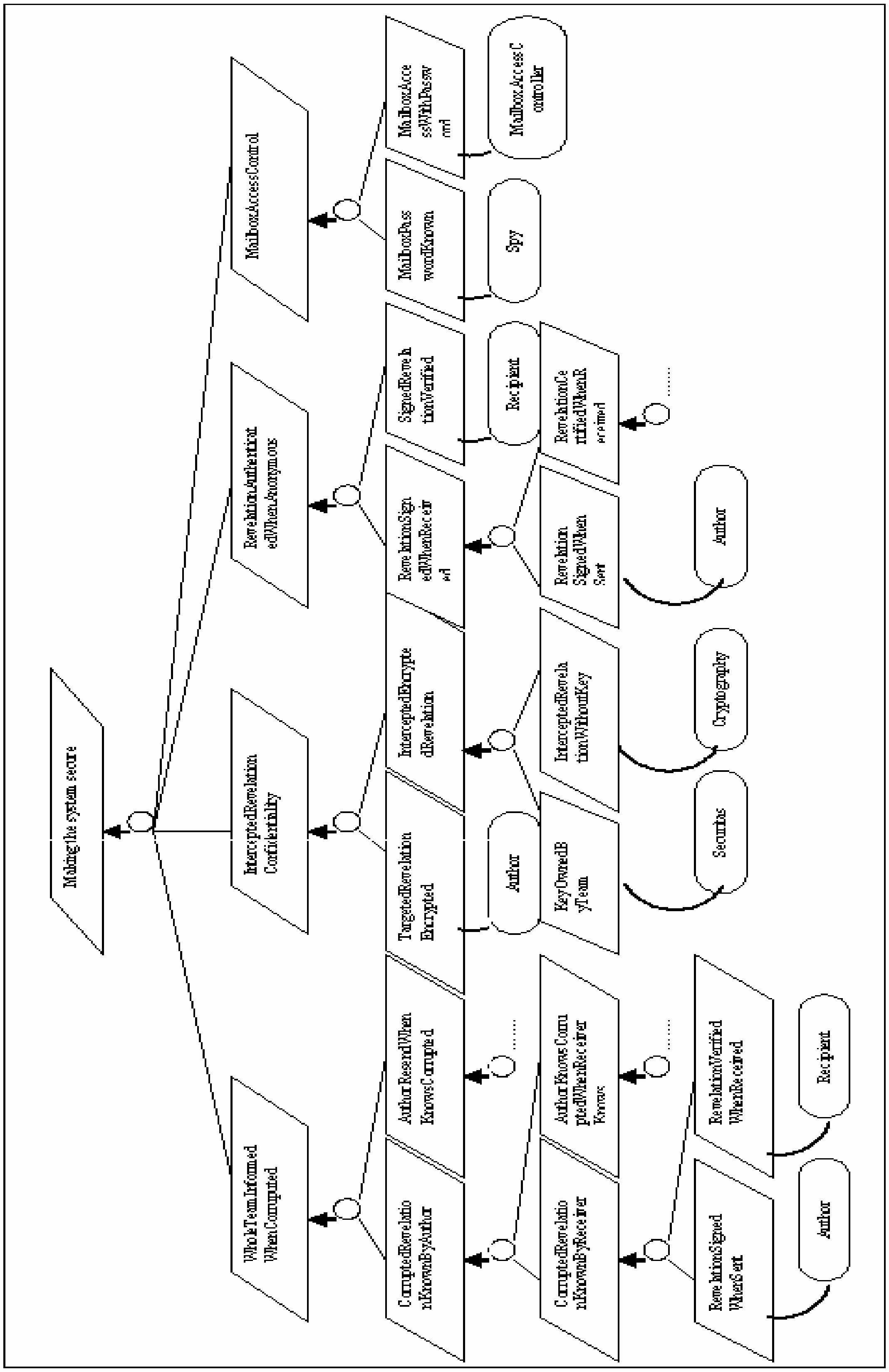

Making the system secure
WholeTeamInformedWhenCorrupted
InterceptedRevelationConfidentiality
RevelationAuthenticatedWhenAnonymous
MailboxAccessControl
CorruptedRevelationKnownByAuthor
AuthorResendWhenKnowsCorrupted
......
TargetedRevelationEncrypted
Author
InterceptedEncryptedRevelation
RevelationSignedWhenReceived
SignedRevelationVerified
Recipient
MailboxPasswordKnown
Spy
MailboxAccessWithPassword
MailboxAccessController
CorruptedRevelationKnownByReceiver
AuthorKnowsCorruptedWhenReceiverKnows
......
KeyOwnedByTeam
Securitas
InterceptedRevelationWithoutKey
Cryptography
RevelationSignedWhenSent
Author
RevelationCertifiedWhenReceived
......
RevelationSignedWhenSent
Author
RevelationVerifiedWhenReceived
Recipient

**Figure 21: Security goal graph for the spy network system**

Figure 22 shows the part of the object model involved in security goals.

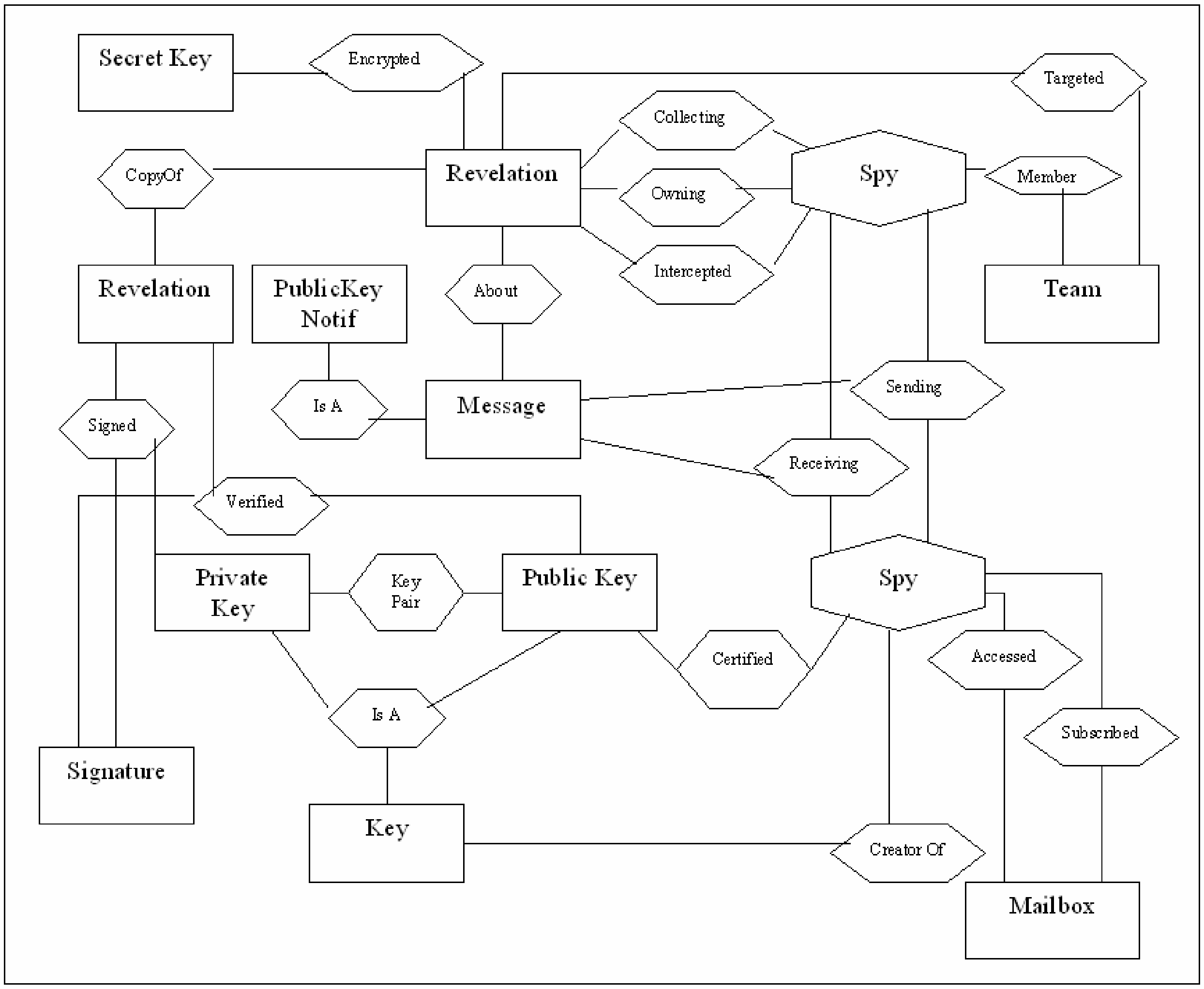


**Figure 22: Partial object model involved in security goals**

Requirements at the very bottom of the goal graph need to be operationalized in order to complete the KAOS model. KAOS operations are means for agents in the software-to-be to achieve their assigned requirements. The goal graph of the security requirements for the spy network system in Figure 21 is big and complicated, so we have summarized the security operations along with the fundamental security goals in Figure 23.

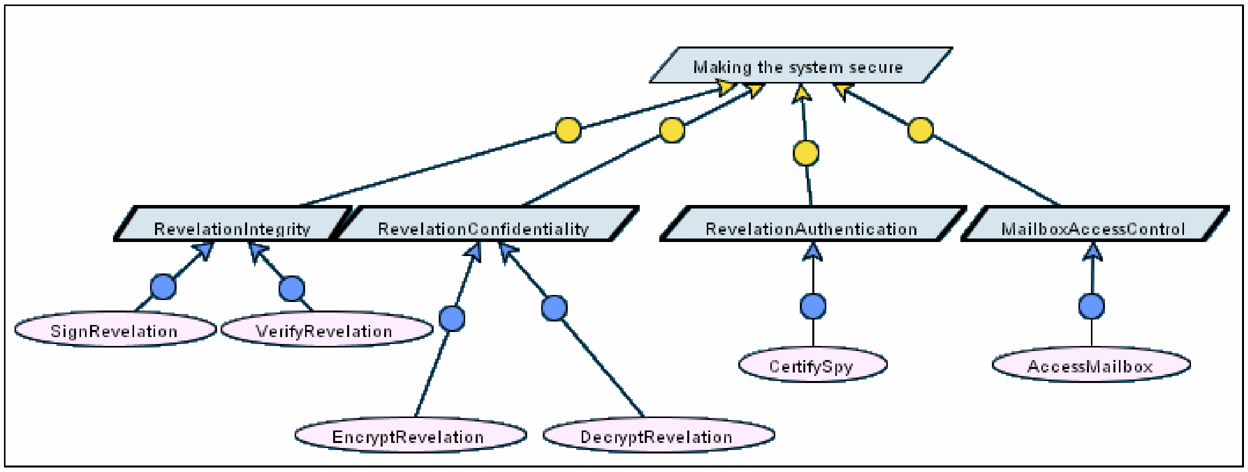


**Figure 23: Summary of the KAOS Operations for the Spy Network Security Goal Graph**

Figure 23 shows six security operations. The SignRevelation and VerifyRevelation operations realize the revelation integrity goals. The EncryptRevelation and DecryptRevelation goals realize the revelation confidentiality goals. The CertifySpy operation realizes the revelation authentication goals and the AccessMailbox realizes the mailbox access control goals. The following formal definition of each operation shows the operation name (highlighted), the input parameters (Input), the output parameters (Output), the precondition (DomPre) that must be true prior to the operation execution, the post condition (DomPost) that must be true after the operation finishes execution, and the goals that this operation are pre-requisite to their achievement (ReqPreFor).

The SignRevelation operation is responsible for achieving the goal RevelationIntegrity and its subgoals. This operation is called when a spy sends a revelation to another spy in order to protect the integrity of the revelation against malicious acts. The operation takes the spy signing the revelation, the revelation to be signed and the private key with which the revelation is signed as input parameters and returns the digital signature as an output.

**Operation** SignRevelation
**Input** Spy{arg sp}, Revelation{arg rev}, Privatekey {arg pk}
**Output** Signature {res sig}
**DomPre** $\neg(\exists$ rev1:Revelation) Signed(rev1, sig, pk)
**DomPost** $(\exists$ rev2:Revelation) Signed(rev2, sig, pk)
**ReqPreFor** RevelationSignedWhenSent
CreatorOf(sp, pk)

The VerifyRevelation operation is responsible for achieving the goal RevelationIntegrity and its subgoals. This operation is called when a spy receives a revelation from another spy in order to verify that the received revelation is not tampered with while in transit. The operation takes the spy verifying the revelation, the revelation to be verified and the public key with which the revelation is verified as input parameters and returns a boolean as an output to indicate whether the revelation is verified correct or not.

**Operation** VerifyRevelation
**Input** Spy{arg sp}, Revelation{arg rev}, Publickey {arg pk}, Signature {arg sig}
**Output** Boolean {res verified}
**DomPre** ¬(∃ rev1:Revelation) Verified(rev1, sig,
Pk)
**DomPost** (∃ rev2:Revelation) Verified(rev2, sig, pk)
**ReqPreFor** RevelationVerifiedWhenReceived
Knows(sp, pk)
**ReqPreFor** RevelationDecryptedWhenReceived
(∃ msg:Message, prk: PrivateKey)
Decrypted(msg, prk) ^ CreatorOf(sp, prk)

The EncryptRevelation operation is responsible for achieving the goal RevelationConfidentiality and its subgoals. This operation is called when a spy sends a revelation to another spy in order to protect the confidentiality of the revelation against malicious acts. The operation takes the spy encrypting the revelation, the revelation to be encrypted and the public key with which the revelation is encrypted as input parameters and returns the encrypted revelation as an output.

**Operation** EncryptRevelation
**Input** Spy{arg sp}, Revelation{arg rev}, Publickey {arg pk}
**Output** Message {res msg}
**DomPre** ¬(∃ rev1:Revelation) Encrypted(rev1, pk)
**DomPost** (∃ rev2:Revelation) Encrypted(rev2, pk)
**ReqPreFor** RevelationEncryptedWhenSent
Knows(sp, pk)
**ReqPreFor** RevelationSignedWhenSent
(∃ prk: PrivateKey, sig:Signature)
Signed(rev, sig, prk) ^ CreatorOf(sp, prk)
**ReqPostFor** RevelationEncyptedWhenSent
About(msg, rev)

The DecryptRevelation operation is responsible for achieving the goal RevelationConfidentiality and its subgoals. This operation is called when a spy receives a revelation to another spy in order to get the content of the encrypted revelation. The operation takes the spy receiving the revelation, the encrypted revelation to be decrypted and the private key with which the revelation is decrypted as input parameters and returns the revelation content as an output.

**Operation** DecryptRevelation
**Input** Spy{arg sp}, Message{arg msg}, Privatekey {arg pk}
**Output** Revelation {res rev}
**DomPre** ¬(∃ msg1:Message) Decrypted(msg1, pk)
**DomPost** (∃ msg2:Message) Decrypted(msg2, pk)
**ReqPreFor** RevelationDecryptedWhenReceived
CreaterOf(sp, pk)
**ReqPostFor** RevelationDecryptedWhenReceived
Knows(sp, rev.Content)

The CertifySpy operation is responsible for achieving the goal RevelationAuthentication and its subgoals. This operation is called when a spy receives a revelation from another spy in order to authenticate the sender of the revelation. The operation takes the spy sending the revelation, and the public key of the sender and returns a boolean as an output indicating whether the revelation comes from a certified spy or not.

**Operation** CertifySpy
**Input** Spy{arg sp}, Publickey{arg pk}
**Output** Boolean {res Authenticated}
**DomPre** ¬Certified(sp, pk)
**DomPost** Certified(sp, pk)
**ReqPreFor** RevelationVerifiedWhenReceived
(∃ rev:Revelation,, sig:Signature)
Verified(rev, sig, pk) ^ CreatorOf(sp, pk)

The AccessMailbox operation is responsible for achieving the goal MailboxAccessControl and its subgoals. This operation is called when a spy tries to access the mailbox in which he/she is subscribed. The operation takes the spy trying to access his/her mailbox, the mailbox being accessed, and the password of the mailbox and returns a boolean as an output indicating whether access to the mailbox is allowed or denied based on the provided password.

**Operation** AccessMailbox

**Input** Spy{arg sp}, Mailbox{arg ma}, Password {arg pa}

**Output** Boolean {res accessed}

**DomPre** $\neg$ Accessed(ma, sp)

**DomPost** Accessed(ma, sp)

**ReqPreFor** MailboxAccessControl

Subscribed(ma, sp)

**ReqPreFor** MailboxAccessedWithPassword

pa = ma.Password

The above operations are transformed to B in the coming subsection to construct the initial B machine that is further refined inside B using the B refinement mechanism to derive design specifications and generate implementation. The rational for transforming KAOS operations to B while not transforming the rest of the goal graph is that operations sums up all the behaviors that agents need to have to fulfill their requirements, which are the leaf goals in the goal graph. The mechanism for constructing the goal graph shows that high level goals are refined using AND/OR refinement steps until leaf goals are derived meaning that the fulfillment of leaf goals implies the fulfillment of the higher level goals in the goal graph. Therefore, it is safe to only transform KAOS operations used to express behaviors of agents that perform them to fulfill the leaf goals in the goal graph without compromising the completeness and consistency properties of the requirements model.

### 4.1.4 Transforming the Spy Network Security Goal Graph to B

Modeling the security requirements of the spy network system with KAOS has defined the goals, the agents responsible for achieving these goals and the means to achieve these goals in the form of the KAOS operations. In order to go further with the security-specific elements from the requirements phase to the design phase, the KAOS requirements model need to be transformed to a design elaboration language, which is the B language in FADES, The transformation focuses on both the KAOS security operations and the entities of the partial object model involved in security goals. The transformation scheme described in section 3.2.1 is a key contribution of FADES since it provides a means to bridge the gap between security requirements and their realization in formal design. The value of the transformation scheme is in stepping further from the relaxed formality of the KAOS requirements model in which requirements are well-organized and reasoned about to more rigid formality in the initial B model that is further refined for design. This means that without the transformation scheme, the variance of formality between requirements and design would not have been possible. This is evidenced in the formal security engineering literature in which rigid formality applies to all the phases of development starting for specifying requirements to deriving implementation like employing the Z or the VDM formal languages for manipulating security concerns. Applying formal languages to requirements specifications has the disadvantages of increasing the cost and complexity of development, using formal languages that are very specific to model requirements that usually have lots of unknowns that cannot be specified formally, and lacking built-in constructs for threat analysis and mitigation.

The Goal Graph Analyzer tool that automates the transformation from KAOS to B parses the XML model produced by the Objectiver tool and representing the KAOS security requirements model for the spy in order to construct the initial B abstract model equivalent to the KAOS model. The initial B model is manipulated using the B-Toolkit, which is one of the two most famous commercial tools for B development as a tool to develop our B model and refine it to derive design specifications and implementation. The abstract B machine for the spy network system is illustrated in the below code.

The spy network system is represented as an abstract machine called SpyNetwork parameterized with the maximum number of spies allowed in the system. The machine represents the spies as a set in the Variables section that model the system state as highlighted in the below code. The Spies' attributes are also modeled as part of the Variables section with the Invariant stating the types of each attribute. The machine invariant states the constraints on the machine state variables. The SpyNetwork machine represents each of the six KAOS operations illustrated in Figure 23 as a B operation as highlighted below. KAOS operations preconditions are directly mapped to preconditions of their corresponding B operations since both are defined in first-order predicate logic. The KAOS definition of operations provides the interface of the operation with which its clients (callers) will call it and that is the task of the requirement analysis phase. The abstract definition of each operation behavior is the responsibility of the early design phase. This means that the generated B abstract machine from the Goal Graph Analyzer needs to be augmented with the abstract specifications for each operation as shown below. The abstraction specification of each operation is further refined using the B refinement mechanism to make the definition more concrete through adding more details and removing the non-determinism in the abstract definition.

```
MACHINE SpyNetwork (maxSpies)
CONSTRAINTS maxSpies : 1..10000
SEES StrTokenType, Bool_Type
DEFINITIONS SPY== 1.. maxSpies

VARIABLES
spies, spyId,spyName, spyMailboxPassword, spyPublicKey, spyPrivateKey,
key, signature –-used as a placeholder for use in local variables

INVARIANT
spies <:SPY &
spyId : spies >-> NATURAL1 &
```

```
spyName : spies --> STRTOKEN &
spyMailboxPassword : spies >-> STRTOKEN &
spyPublicKey : spies >-> STRTOKEN &
spyPrivateKey : spies >-> (STRTOKEN - spyPublicKey) &
spyId >< spyName >< spyMailboxPassword >< spyPublicKey >< spyPrivateKey :
spies >-> (NATURAL1 >< STRTOKEN >< STRTOKEN >< STRTOKEN >< STRTOKEN) &
key : STRTOKEN & signature : STRTOKEN
```

**OPERATIONS**

```
NewSpy(identity, name, mailboxPassword, publicKey, privateKey) =
  PRE
          identity: NATURAL1 & name: STRTOKEN & mailboxPassword: STRTOKEN &
          publicKey: STRTOKEN & privateKey: STRTOKEN &
          (identity, name, mailboxPassword, publicKey, privateKey)/:
          ran(spyId >< spyName >< spyMailboxPassword >< spyPublicKey >< spyPrivateKey) &
          spies /= SPY
   THEN
          ANY newSpy WHERE newSpy : SPY – spies THEN
      spies := spies \/ {newSpy} || spyId(newSpy) := identity ||
      spyName(newSpy) := name ||
          spyMailboxPassword(newSpy) := mailboxPassword ||
          spyPublicKey(newSpy) := publicKey ||
          spyPrivateKey(newSpy) := privateKey
    END
END;

found, spyDetails <-- getSpy(identity) =
    PRE identity : NATURAL1  THEN
    IF (identity) : ran(spyId) THEN
       spyDetails := (spyId >< spyName >< spyMailboxPassword ><
                                    spyPublicKey >< spyPrivateKey)~ (identity) ||
           found := TRUE
    ELSE
           spyDetails :  SPY ||
           found := FALSE
    END
```

```
END;

full <-- spyNetworkFull =
BEGIN
    full := bool(spies = SPY)
END;

encyptedRevelation <-- encryptRevelation(revelation, identity) =
    PRE  revelation : STRTOKEN & identity : NATURAL1  THEN
    IF (identity) : ran(spyId) THEN
            key := spyPublicKey(identity) ||
            signature := signRevelation(revelation, identity);
            encyptedRevelation : (signature; key) >-> STRTOKEN
    ELSE
            encyptedRevelation ::  STRTOKEN
    END
END;

revelation <-- decryptRevelation(ecryptedRevelation, identity) =
    PRE  ecryptedRevelation : STRTOKEN & identity : NATURAL1 THEN
    IF (identity) : ran(spyId) THEN
            key := spyPrivateKey(identity) ||
            revelation : ecryptedRevelation >-> STRTOKEN
    ELSE
            revelation ::  STRTOKEN
    END
END;

signature <-- signRevelation(revelation, identity) =
   PRE revelation : STRTOKEN & identity : NATURAL1 THEN
    IF (identity : ran(spyId)) THEN
            key := spyPrivateKey(identity) ||
            signature : revelation >-> STRTOKEN
    ELSE
            signature :: STRTOKEN
    END
END;
```

```
verified <-- verifyRevelation(revelation, identity, signature) =
   PRE
           revelation : STRTOKEN & identity : NATURAL1 & signature : STRTOKEN  THEN
   IF (identity) : ran(spyId) THEN
           key := spyPublicKey(identity) ||
           IF (signature == revelation(key))  THEN -- how to indicate the application of key to revelation
                verified := TRUE
           ELSE
                verified := FALSE
           END
    ELSE
      verified := FALSE
    END
END;

authenticated <-- certifySpy(identity, publicKey) =
    PRE  identity : NATURAL1 & publicKey : STRTOKEN  THEN
    authenticated := bool(spyPublicKey(identity) = publicKey)
END;

accessAllowed <-- accessMailbox(identity, password) =
   PRE identity : NATURAL1 & password : STRTOKEN THEN
    IF (identity : ran(spyId)) THEN
           IF (password == spyMailboxPassword(identity)) THEN
              accessAllowed := TRUE
           ELSE
              accessAllowed := FALSE
           END
    ELSE
           accessAllowed := FALSE
    END
END;
```

To illustrate the idea of the abstract definition of operation behavior that augments the generated B machine from the Goal Graph Analyzer Tool, let’s describe the definition of the encryptRevelation operation as an example. The definition first checks on whether the spy whose identity is used to encrypt the message belongs to the set of spies in the network. If the spy identity is verified, the public key of the spy is retrieved from the set of public keys stored in the variables section using the spy identity. The revelation is signed before encrypted by calling the signRevelation operation and finally the encrypted revelation is abstractly defined as another string value calculated from the original revelation. The abstract definition of the encrypted revelation allows for multiple concrete definitions of how to encrypt the original revelation depending on the design decision made in the coming refinement steps on which encryption algorithm is used.

### 4.1.5 Derivation of Design and Implementation

The initial B machine is refined using the B refinement mechanism to enforce design decisions. The first refinement step is classified as data refinement concerned with refining the representation of the machine variables that reflect the system state. The first refinement step represents the pool of spies as an array of spies and we will show how the security operations are refined accordingly. From the traceability perspective, the data refinement step does not directly address the realization of a specific security requirement in the system. It rather concentrates on building a concrete data structure representing the internal system state in a form realizable by programming languages while implementation is generated. The first refinement is by convention named with the same name of the machine it refines with an R appended to the machine name. The invariant of the refining machine should be linked to the variables of the refined machine by means of linking invariant that describes the relationship between the state spaces of the two machines [43]. The linking invariant is used to generate proof obligations that specify which proofs need to be discharged in order to prove that the refining machine does not violate any of the constraints of the refined machine; therefore, proves correctness of development and preservation of security properties. The linking invariant in the first refinement step of the spy network system is dom(spiesr) = spies meaning that the set spies is precisely the domain of the function spiesr since this gives the index of the set of elements that appear in the array. The elements in the spiesr array are the Ids of all the spies in the system. The rest of the spy's attributes are linked to each spy through his/her Id. The operations are defined on the variable spiesr. Operations are required to work only within their preconditions given in the abstract machine, so those preconditions are assumed to hold for the refined operations. Therefore, the type information of the input variables and the other requirements on them do not need to be repeated in the refinement machine. The highlighted parts of the first refinement machine reflects the implication of the data refinement on the abstract definition of the operations.

**REFINEMENT** SpyNetworkR
**REFINES** SpyNetwork
**SEES** StrTokenType, Bool_TYPE

**VARIABLES**

spiesr, spyNamer, spyMailboxPasswordr, spyPrivatekeyr, spyPublickeyr

**INVARIANT**

spiesr : 1.. maxSpies >-> spyId & dom(spiesr) = spies &

spyNamer : spiesr >-> STRTOKEN & ran(spyNamer) = spyName &

spyMailboxPasswordr : spiesr >-> STRTOKEN &

ran(spyMailboxPasswordr) = spyMailboxPassword &

spyPrivatekeyr : spiesr >-> (STRTOKEN – spyPublickey) &

ran(spyPrivatekeyr) = spyPrivatekey &

spyPublickeyr : spiesr >-> (STRTOKEN- spyPrivatekey) &

ran(spyPublickeyr) = spyPublickey

**INITIALIZATION** Spiesr := (1..maxSpies) >< {0} -- all spies ids are initialized to 0

**OPERATIONS**

```
encyptedRevelation <-- encryptRevelation(revelation, identity) =
   VAR key IN  THEN
   IF (spiesr~(identity): dom(spiesr)) THEN
         key := spyPublickeyr(identity) ||
         signature := signRevelation(revelation, identity);
         encyptedRevelation : (signature; key) >-> STRTOKEN
   ELSE
         encyptedRevelation ::  STRTOKEN
   END
END;

revelation <-- decryptRevelation(ecryptedRevelation, identity) =
   VAR key IN
   IF (spiesr~(identity): dom(spiesr)) THEN
         key := spyPrivatekeyr(identity) ||
         revelation : ecryptedRevelation >-> STRTOKEN
   ELSE
         revelation ::  STRTOKEN
   END
END;
```

```
signature <-- signRevelation(revelation, identity) =
  VAR pk IN
  IF (spiesr~(identity): dom(spiesr)) THEN
     pk := spyPrivatekeyr(identity);
     signature : revelation >-> STRTOKEN
  ELSE
     signature:= EmptyStringToken;
  END
END;

verified <-- verifyRevelation(revelation, identity, signature) =
    VAR key IN
    IF (spiesr~(identity) : dom(spiesr))  THEN
              key := spyPublickeyr(identity) ||
              IF (signRevelation(revelation, identity) == signature) THEN
                    verified := TRUE
              ELSE
                    verified := FALSE
              END
     ELSE
        verified := FALSE
     END
END;

authenticated <-- certifySpy(identity, publicKey) =
      VAR key IN
     authenticated := bool(spyPublickeyr(identity) = publicKey)
END;

accessAllowed <-- accessMailbox(identity, password) =
  IF (spiesr~(identity) : dom(spiesr)) THEN
     IF (password == spyMailboxPasswordr(identity)) THEN
              accessAllowed := TRUE
     ELSE
              accessAllowed:= FALSE
     END
  ELSE
```

```
    accessAllowed:= FALSE
  END
END;
```

The above B code that represents the first refinement step maintains the abstract definition of the machine operations and reflects the new data representation of machine variables through using the arrays spiesr, spyNamer, spyMailboxPasswordr, spyPrivatekeyr, spyPublickeyr instead of the abstract sets.

The second refinement step is a procedural refinement that makes design decisions about the security algorithms used for preserving the revelation integrity and confidentiality. The refining machine does not modify the state representation (machine variables) of the refined machine to remove the non-determinism of the precise procedures to protect the revelation integrity and confidentiality.

The design decision to employ the DSA (Digital Signature Algorithm) as the signature algorithm has been made to achieve the revelation integrity requirements since the DSA is a standard algorithm proposed by the National Institute of Standards and Technology (NIST) in 1991. For the revelation confidentiality, it was decided to employ DES (Data Encryption Standard) since the DES algorithm is has been developed and endorsed by the U.S. government as an official encryption standard in 1977. For the mailbox access control, each spy is assigned a password for the mailbox in which he/she is subscripted.

The refinement mechanism in B allows for documenting design decisions at each refinement step and this links these design decisions to the relevant requirements. For example, the employment of the DSA algorithm for digital signature links to the requirements of revelation integrity through the operations signRevelation and verifyRevelation that carry out the algorithm. The KAOS goal graph provides traceability links between operations carried out in design and the requirements they achieve. As stated earlier in the document, the availability of sufficient traceability information allows for more accurate impact analysis in FADES when changes to security specifications are introduced as demonstrated in section 4.1.7.

The design decisions made in the second refinement step are highlighted in the following B code:

```
REFINEMENT SpyNetworkRR
REFINES SpyNetworkR
SEES StrTokenType, Bool_TYPE, SpyNetworkUtilities

OPERATIONS

encyptedRevelation <-- encryptRevelation(revelation, identity) =
   VAR publicKey IN
   IF (spiesr~(identity): dom(spiesr)) THEN
     publicKey := spyPublicKeyr(identity);
```

```
            encyptedRevelation := SpyNetworkUtilities.DESencrypt (revelation, publicKey);
    ELSE
            encyptedRevelation :=  EmptyStringToken;
    END
END;

revelation <-- decryptRevelation(ecryptedRevelation, identity) =
    VAR privateKey IN
    IF (spiesr~(identity): dom(spiesr)) THEN
      privateKey := spyPrivateKeyr(identity);
            revelation := SpyNetworkUtilities.DESdecrypt (revelation, privateKey);
    ELSE
            revelation :=  EmptyStringToken;
    END
END;

signature <-- signRevelation(revelation, identity) =
   VAR privateKey, publicKey IN
   IF (spiesr~(identity) : dom(spiesr))  THEN
     privateKey := spyPrivateKeyr(identity);
     publicKey := spyPublicKeyr(identity);
     signature := SpyNetworkUtilities.DSAsign
                     (revelation, privateKey, publicKey) ;
   ELSE
     signature:= EmptyStringToken;
   END
END;

verified <-- verifyRevelation(revelation, identity, signature) =
    VAR privateKey, publicKey IN
    IF (spiesr~(identity) : dom(spiesr))  THEN
            privateKey := spyPrivateKeyr(identity);
      publicKey := spyPublicKeyr(identity);
            verified := SpyNetworkUtilities.DSAverify
                     (revelation, privateKey, publicKey, signature) ;
    ELSE
      verified := FALSE;
    END
END;

authenticated <-- certifySpy(identity, publicKey) =
     VAR key IN
    authenticated := bool(spyPublicKey(identity) = publicKey)
END;

accessAllowed <-- accessMailbox(identity, password) =
   IF (spiesr~(identity) : dom(spiesr)) THEN
    IF (password == spyMailboxPasswordr(identity)) THEN
            accessAllowed := TRUE
    ELSE accessAllowed:= FALSE
    END
   ELSE accessAllowed:= FALSE
   END
END;
```

The security algorithms used in the second refinement step are encapsulated in another machine SpyNetworkUtilities that encapsulates generic security utilities. This makes the design more modular by dividing the tasks among multiple machines, each of which provides interfaces for its operations to be used by other machines. For example, the second refinement uses the DESencrypt, DESdecrypt, DSAsign , and DSAverify from the SpyNetworkUtilities machine. Separating these operations in separate machine allows for changing the security algorithms used for encryption/decryption and digital signatures seamlessly as far as the spy network machine itself is concerned since the change will be localized only in the SpyNetworkUtilities machine. The spy network machine will not be affected by the change since it remains using the same operations with the same interface while the SpyNetworkUtilities machine changes the implementation.

The last refinement step derives implementation specifications for the security requirements of the spy network system. The implementation step can be done only once for each development with some constraints such as that the implementation machine has no state and cannot use abstract substitutions like non-determinism choice and parallel composition. The implementation machine imports the SpyNetworkUtilities, privateKeyArray, publicKeyArray, and spyMailboxArray machines that are then considered under the control of the implementation machine. Further, the implementation machine has to instantiate the parameters of the imported machines. The implementation machine has no variables section meaning that it has no state itself; rather it maintains the state of the imported machines. The invariant section of the implementation machine contains liking invariants between the imported state variables namely spiesArray, privateKeyArray, publicKeyArray, spyMailboxPasswordArray, and spyMailboxArray and the variables of the SpyNetworkR1 that this implementation refines. Operations in the implementation machine need to convert mathematical specifications to a format that could be translated to a programming language. For example, all the operations in the implementation machine use a loop to locate the spy whose identity is specified in the operation parameters in the array of spies. This search in the array using a loop could be directly translated to equivalent constructs in the generated C code. The B code for the implementation specifications is as follows:

```
IMPLEMENTATION SpyNetworkI
REFINES SpyNetworkR1
USES StrTokenType, Bool_TYPE
IMPORTS
SpyNetworkUtilities, privateKeyArray(maxSpies), publicKeyArray(maxSpies),
spyMailboxArray(maxSpies)
INVARIANT
spiesArray = spiesr & privateKeyArray = spyPrivateKeyr &
publicKeyArray = spyPublicKeyr & spyMailboxPasswordArray = spyMailboxPasswordr &
spyMailboxArray = spyMailboxPasswordr

OPERATIONS

encyptedRevelation <-- encryptRevelation(revelation, identity) =
  VAR ii, publicKey IN
  ii := 0;
  WHILE ii <= maxSpies
  DO ii := ii + 1;
  IF (spiesArray(ii) == identity) THEN
          publicKey := publicKeyArray(ii);
          encyptedRevelation := SpyNetworkUtilities.DESencrypt (revelation, publicKey);
  ELSE
     encyptedRevelation:= EmptyStringToken;
  END
  INVARIANT
    ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr & publicKeyArray = spyPublicKeyr
  VARIANT
    maxSpies - ii
  END
```

```
END;

revelation <-- decryptRevelation(ecryptedRevelation, identity) =
   VAR ii, privateKey IN
   ii := 0;
   WHILE ii <= maxSpies
   DO ii := ii + 1;
   IF (spiesArray(ii) == identity) THEN
            privateKey := privateKeyArray(ii);
            revelation := SpyNetworkUtilities.DESdecrypt (revelation, privateKey);
   ELSE
       revelation :=  EmptyStringToken;
   END
   INVARIANT
     ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr &
     privateKeyArray = spyPrivateKeyr
   VARIANT
     maxSpies - ii
   END
END;

signature <-- signRevelation(revelation, identity) =
   VAR ii, privateKey, publicKey IN
   ii := 0;
   WHILE ii <= maxSpies
   DO ii := ii + 1;
   IF (spiesArray(ii) == identity) THEN
            privateKey := privateKeyArray(ii);
            publicKey := publicKeyArray(ii);
            signature := SpyNetworkUtilities.DSAsign
                        (revelation, privateKey, publicKey);
   ELSE
       signature:= EmptyStringToken;
   END
   INVARIANT
     ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr &
     privateKeyArray = spyPrivateKeyr & publicKeyArray = spyPublicKeyr
```

```
  VARIANT
    maxSpies - ii
  END
END;

verified <-- verifyRevelation(revelation, identity, signature) =
  VAR ii, privateKey, publicKey IN
  ii := 0;
  WHILE ii <= maxSpies
  DO ii := ii + 1;
  IF (spiesArray(ii) == identity) THEN
          privateKey := privateKeyArray(ii);
          publicKey := publicKeyArray(ii);
          verified := SpyNetworkUtilities.DSAverify
                    (revelation, privateKey, publicKey, signature);
  ELSE
      verified := FALSE;
  END
  INVARIANT
    ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr &
    privateKeyArray = spyPrivateKeyr & publicKeyArray = spyPublicKeyr
  VARIANT
    maxSpies - ii
  END
END;

authenticated <-- certifySpy(identity, publicKey) =
  VAR ii, pk IN
  ii := 0;
  WHILE ii <= maxSpies
  DO ii := ii + 1;
  IF (spiesArray(ii) == identity) THEN
          pk := publicKeyArray(ii);
          IF (pk == publicKey) THEN
             authenticated := TRUE;
          ELSE
             authenticated := FALSE;
```

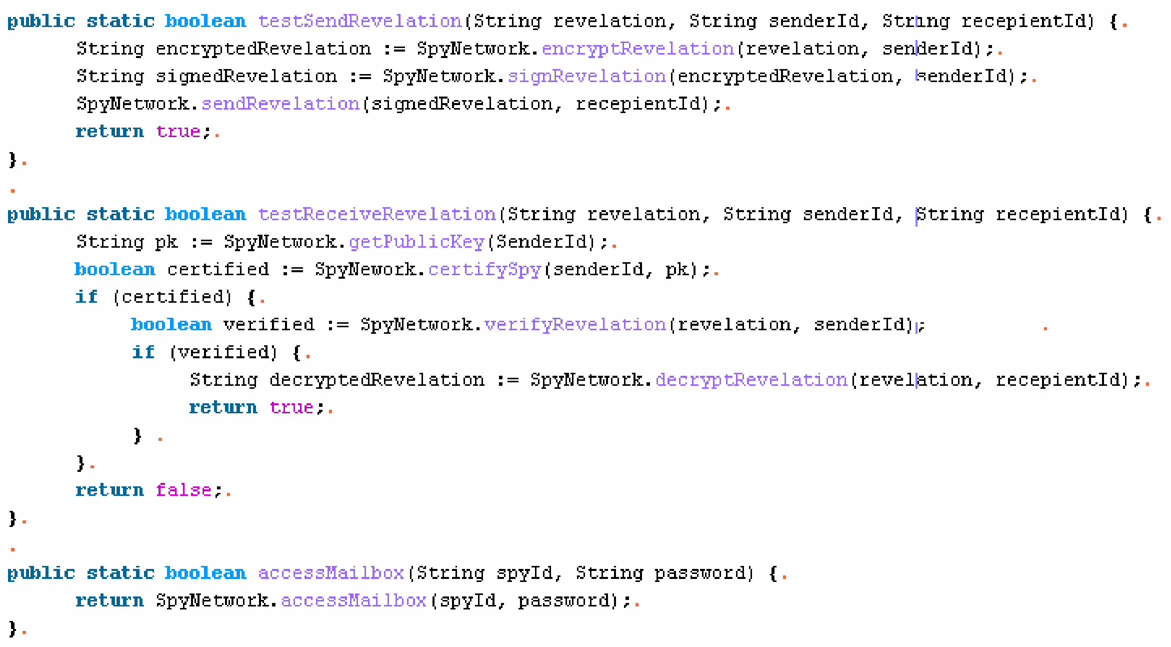
```
        END
    ELSE authenticated:= FALSE
    END
    INVARIANT
      ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr  &
      publicKeyArray = spyPublicKeyr
    VARIANT
      maxSpies - ii
    END

accessAllowed <-- accessMailbox(identity, password) =
    VAR ii IN
    ii := 0;
    WHILE ii <= maxSpies
    DO ii := ii + 1;
    IF (spiesArray(ii) == identity) THEN
        IF (password == spyMailboxArray(ii)) THEN
          accessAllowed := TRUE
      ELSE accessAllowed:= FALSE
      END
    ELSE accessAllowed:= FALSE
    END
    INVARIANT
      ii : NATURAL1 & ii <= maxSpies & spiesArray = spiesr &
      spyMailboxArray = spyMailboxPasswordr
    VARIANT
      maxSpies - ii
    END
END;
```

The final stage is to generate C code from the implementation machine. The programming language choice is based on the programming languages available in the B tool being used. Almost all the commercial B tools generate code in C and very few of them generate ADA. The security properties should be maintained by the design decisions and the semantics of the B machines rather than by specific security construct in the programming language to which the implementation machine is translated.

### 4.1.6 Acceptance Testing

The Goal Graph Analyzer tool generates a suite of acceptance test cases derived directly from the KAOS goal graph. The tool parses the graph using a DFS algorithm with backtracking facility in order to generate scenarios with a sequence of operation calls from the goal graph. The derived B implementation specifications are then verified against the acceptance test cases to check for compliance between the requirements model and the derived implementation and to ensure the preservation of the security properties specified in the requirement model. Test results can be used to identify areas of inconsistencies and errors either in the requirements model itself or in the B refinement steps. The acceptance test cases are considered as substantial contribution of FADES since it strengthens the approach with extra verification of development correctness and compliance between the software and its requirements from the security standpoint. The implication of this contribution is that it increases confidence of the software security developed with FADES

The generated test cases might be augmented with some messages and assertions for better usability of the test results. The Goal Graph Analyzer has generated the following test cases for the spy network security requirements:



```
public static boolean testSendRevelation(String revelation, String senderId, String recepientId) {
      String encryptedRevelation := SpyNetwork.encryptRevelation(revelation, senderId);
      String signedRevelation := SpyNetwork.signRevelation(encryptedRevelation, senderId);
      SpyNetwork.sendRevelation(signedRevelation, recepientId);
      return true;
}

public static boolean testReceiveRevelation(String revelation, String senderId, String recepientId) {
      String pk := SpyNetwork.getPublicKey(SenderId);
      boolean certified := SpyNework.certifySpy(senderId, pk);
      if (certified) {
            boolean verified := SpyNetwork.verifyRevelation(revelation, senderId);
            if (verified) {
                  String decryptedRevelation := SpyNetwork.decryptRevelation(revelation, recepientId);
                  return true;
            }
      }
      return false;
}

public static boolean accessMailbox(String spyId, String password) {
      return SpyNetwork.accessMailbox(spyId, password);
}
```

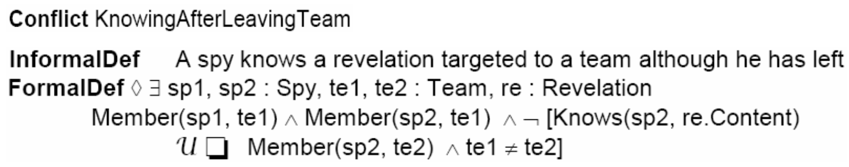
In the above generated test cases, each test case describes the sequence of calls to the security operations in order to secure the communication of revelations. The first test case outlines the scenario for the sending operation in which the encryptRevelation and signRevelation operations are called before sending the revelation. The second test case handles the scenario for the receiving operation in which the certifySpy is called to authenticate the sender spy followed by a verification of the revelation signature through calling verifyRevelation and at the end the revelation is decrypted by calling the decrptRevelation operation. The third test case tests the eligibility of access to a mailbox that could be used when spies login to their mailboxes and it calls the accessMailBox opreation.

It can be observed that all the security operations have been called in the generated test cases using the main scenarios in which these operations are called. This raises the probability of error-detection when the derived implementation is verified against the acceptance test cases increasing the confidence in the security properties of the final product. Further, the coverage of the acceptance test cases provides a means to the customer to verify that the final product meets his security requirements.

### 4.1.7 Security Specifications Changes

This section illustrates how changes to security specifications are effectively handled within FADES and how the availability of sufficient traceability information allows for accurate impact analysis of changes. Maintenance activities are classified into four categories according to [67]: adaptive (changes in the software environment), perfective (new user requirements), corrective (fixing errors), and preventive (prevent future problems). An example of a corrective change has been chosen since corrective changes consume 21% of change requests [67]. The other three categories of maintenance activities will be part of our future work.

A defective scenario threatening revelation confidentiality is as follows: A spy leaves his team and gets reallocated to another team after a message has been sent. This scenario is not handled by the current encryption/decryption solution used to protect the confidentiality of revelations since the leaving spy would receive a revelation that he is no longer eligible to receive. To correct this defect, the KAOS framework provides a conflict construct that allows the expression of situations that contradict with system requirements. We introduce the following conflict to the RevelationConfidentiality goal.


**Conflict** KnowingAfterLeavingTeam

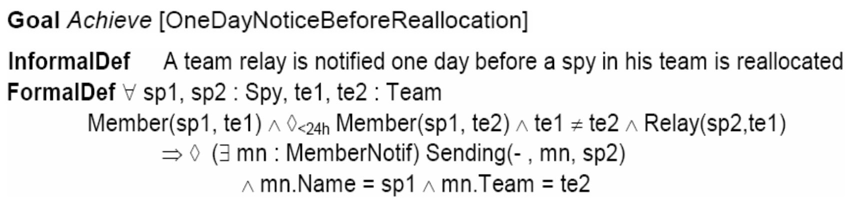
**InformalDef** A spy knows a revelation targeted to a team although he has left

**FormalDef** $\Diamond\ \exists$ sp1, sp2 : Spy, te1, te2 : Team, re : Revelation

$\quad$ Member(sp1, te1) $\wedge$ Member(sp2, te1) $\wedge \neg$ [Knows(sp2, re.Content)

$\quad\quad$ $\mathcal{U}\ \square$ Member(sp2, te2) $\wedge$ te1 $\neq$ te2]

This conflict could be resolved using one of the patterns for conflict resolution [4] by introducing a new goal to anticipate the conflict:


**Goal** *Achieve* [OneDayNoticeBeforeReallocation]

**InformalDef** A team relay is notified one day before a spy in his team is reallocated

**FormalDef** $\forall$ sp1, sp2 : Spy, te1, te2 : Team

$\quad$ Member(sp1, te1) $\wedge\ \Diamond_{<24h}$ Member(sp1, te2) $\wedge$ te1 $\neq$ te2 $\wedge$ Relay(sp2,te1)

$\quad\quad$ $\Rightarrow \Diamond$ ($\exists$ mn : MemberNotif) Sending(- , mn, sp2)

$\quad\quad\quad$ $\wedge$ mn.Name = sp1 $\wedge$ mn.Team = te2

This goal could be assigned to a reliable agent such as the big boss. Analyzing the impact of introducing the new goal shows that the RevelationConfidentiality requirement would be affected by this change. The traceability information provided by the hierarchical structure of the goal graph and the KAOS refinement mechanism direct the change impact analysis to revisit the AND refinement of the RevelationConfidentiality goal. The new goal needs to be added as a subgoal to the refinement of the RevelationConfidentiality goal. The goal graph for the RevelationConfidentiality goal would be modified as in Figure 24 to add the new goal:

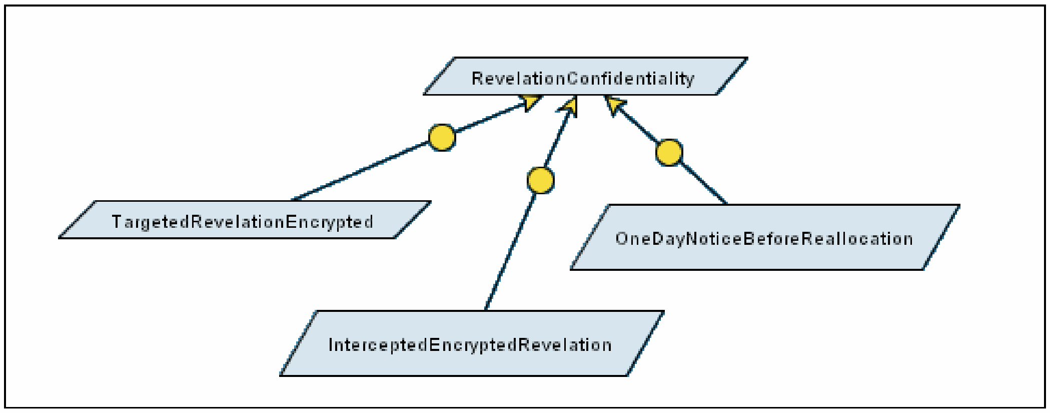

RevelationConfidentiality
TargetedRevelationEncrypted
OneDayNoticeBeforeReallocation
InterceptedEncryptedRevelation

**Figure 24: Accommodating the new goal**

Since the new goal is a leaf goal, it will be operationalized using the following operation:

**Operation** NotifyRelayWithReallocation

**Input** Spy{arg relay}, Spy{arg leavingSpy}

**DomPre** $\neg$($\exists$ relay, leavingSpy:Spy) Notified(relay, leavingSpy)

**DomPost** ($\exists$ relay, leavingSpy:Spy) Notified(relay, leavingSpy)

**ReqPreFor** ($\exists$ team1, team2:Team) Member(relay, team1) ^ Member(leavingSpy, team1) ^ $\Diamond_{<24h}$ Member(leavingSpy, team2)

This operation needs to be transformed to B in order to propagate this corrective change to the derived design and implementation. According to the change impact analysis performed with respect to the transformation of the new operation to B, we discovered that the state representation (Variables) of the SpyNetwork machine needs to be complemented with the following variables: team, relaySpies, authorizedReceiversFrom, authorizedSendersTo. These variables represent the set of assigned relays of all teams as well as the set of spies authorized to send to or receive from relays. Constraints on these variables need to be added to the invariant of the machine as follows:

```
team : spies-->STRTOKEN &
relaySpies <: spyId &
authorizedReceiversFrom : relaySpies >+> spies &
authorizedSendersTo : spies +> relaySpies
```

The definition of the operation notifyRelayWithReallocation is given below and it should be refined for design and implementation like the rest of the operations.

```
notifyRelayWithReallocation(relayId, leavingSpyId) =
  PRE relayId : NATURAL1 & leavingSpyId : NATURAL1 &
  team (relayId) = team (leavingSpyId) THEN

  IF ((relayId : ran(spyId)) & (leavingSpyId : ran(spyId))) THEN
     authorizedReceiversFrom(relayId) := authorizedReceiversFrom(relayId) - {leavingSpyId} ||
     authorizedSendersTo(relayId) := authorizedSendersTo(relayId) - {leavingSpyId}
  END;
END
```

## 4.2 The Electronic Smart Card Case Study

This section demonstrates FADES using the electronic smart card case study presented by Stepney et. al. who modeled and implemented the security requirements of the system using the Z formal language [6]. The last segment of this section provides a qualitative comparison between the application of FADES versus the application of Z to analyze, design and implement the security-specific elements of the electronic smart card system.

### 4.2.1 Electronic Smart Card Problem Statement and Security Requirements

The electronic purse case study is a smaller version of an actual project developed by the NatWest Development Team of a Smartcard product for electronic commerce [6]. The product needs high security assurance since it is essential to ensure the smart cards are free from security deficiencies for the cards to be safely usable.

The system consists of a number of electronic purses containing financial values, each hosted on a smart card. The purses interact with each other through a communication device to exchange value. The system is a decentralized distributed system; each purse is supposed to ensure the security of all its transactions without the interference of a central controller. All the security measures have to be embedded in the card itself with no real-time external audit logging or monitoring.

We have followed the n-step protocol used by Stepney to model a transaction for money exchange between purses in the electronic smart card system. The basic n-step protocol comprises of the following steps as illustrated in Figure 25:

1. The payee requests the transfer of an amount from the payer.
2. The payer sends the amount requested to the payee if the payer balance is greater than or equal the requested amount.
3. The payee sends an acknowledgement of receipt to the payer.

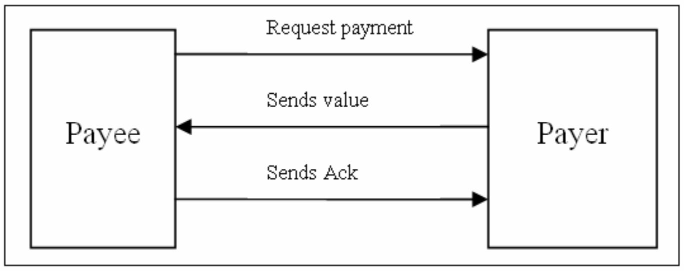


**Figure 25: n-step Protocol for Money Exchange**

There are some assumptions and domain properties in the environment in which the electronic smart card system will operate. The electronic purses communicate using an insecure and unreliable medium forcing the handling of security concerns by the application itself. The communication device is responsible for ascertaining the transactions to perform. The application should also maintain no global state as each purse has to be implemented in isolation. Further, lost messages should be tracked through transaction logging.

The security properties that should be maintained by any transaction for money transfer:

- *No value may be created in the system*: the sum of all the purses' balances after any transactions does not increase.
- *All value is accounted for in the system*; that is, no value is lost: the sum of all purses' balances and lost components after any transaction does not change.
- *Authentic purses*: a transfer can occur only between authentic purses. Authentic purses are those registered in the communication device with valid card and pin numbers for each purse.
- *Sufficient funds*: a transfer can occur only if there are sufficient funds in the payer purse.
- *Exception logging*: if a purse aborts a transfer at a point where value could be lost, the purse should log the transaction details.

While Figure 25 illustrates the abstract model of the communication protocol, Figure 26 illustrates the end to end scenario for a single transaction using a concrete protocol based on messages.

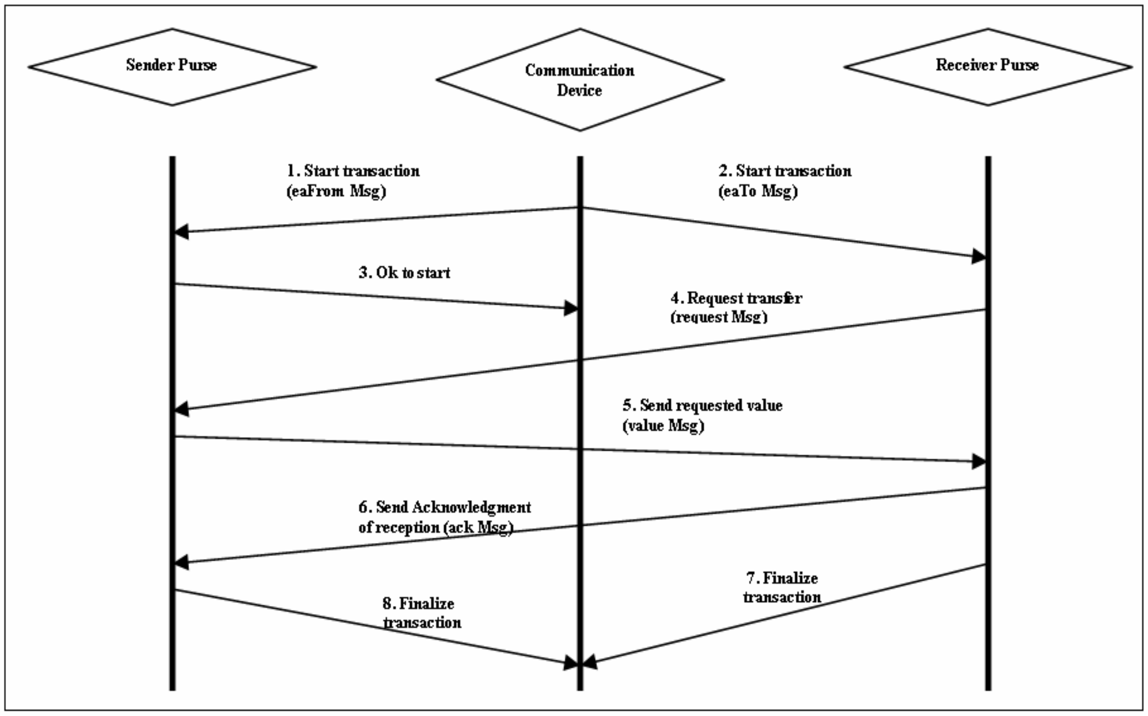

Sender Purse
Communication Device
Receiver Purse
1. Start transaction (eaFrom Msg)
2. Start transaction (eaTo Msg)
3. Ok to start
4. Request transfer (request Msg)
5. Send requested value (value Msg)
6. Send Acknowledgment of reception (ack Msg)
7. Finalize transaction
8. Finalize transaction

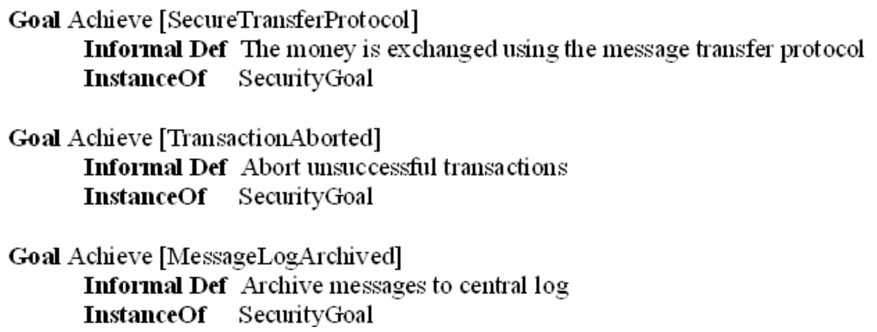
**Figure 26: Communication Protocol Specifications**

### 4.2.2 Elaborating Security Requirements with KAOS

This subsection outlines the elaboration of security requirements for the electronic smart card system with KAOS. The high level goal of this system is to transfer money securely between purses.

**Goal** Maintain [TransferMoneySecurely]
  **InformalDef** The money is exchanged between two purses securely
  **InstanceOf** SecurityGoal

The high level goal is refined into three subgoals that provide two alternatives to the secure exchange of money, either to successfully implement the steps of the communication protocol specified in Figure 24 or to abort an unsuccessful transaction so that money is lost securely. In either case of a successful money exchange or an aborted transaction, all messages involved in the transaction have to be logged to a central archive for history recording. TransferMoneySecurely is refined into the following three subgoals:

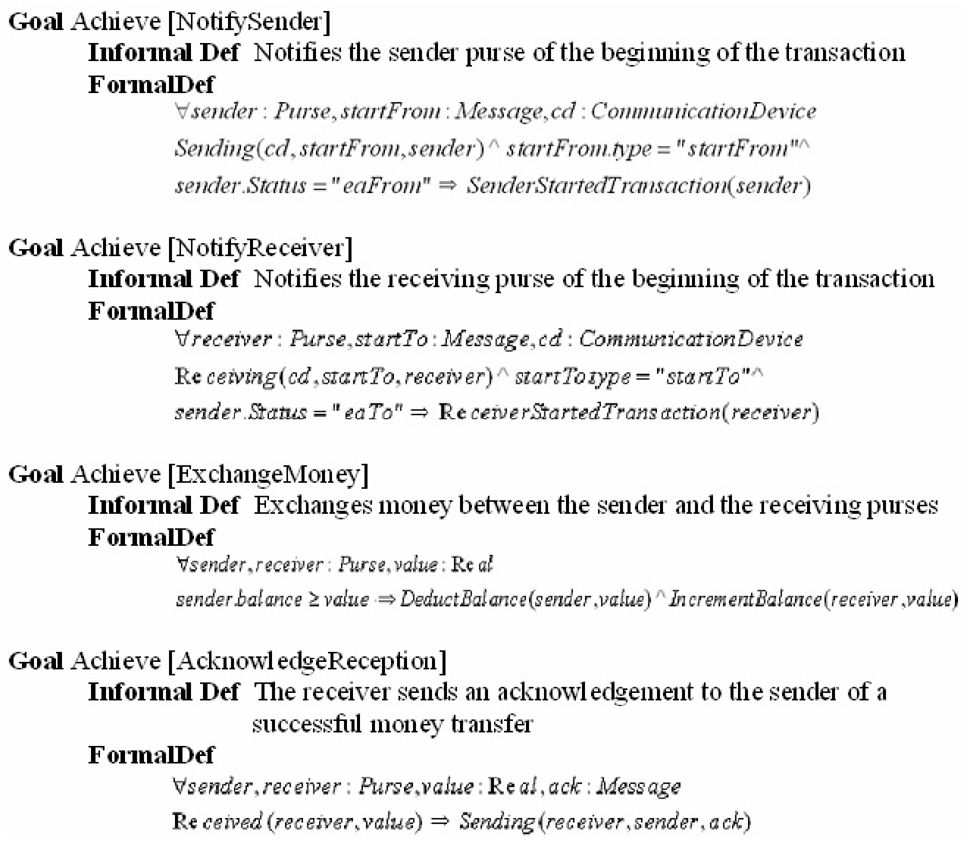


**Goal** Achieve [SecureTransferProtocol]
  **Informal Def** The money is exchanged using the message transfer protocol
  **InstanceOf** SecurityGoal

**Goal** Achieve [TransactionAborted]
  **Informal Def** Abort unsuccessful transactions
  **InstanceOf** SecurityGoal

**Goal** Achieve [MessageLogArchived]
  **Informal Def** Archive messages to central log
  **InstanceOf** SecurityGoal

The goal SecureTransferProtocol is refined into four subgoals realizing the steps of the communication protocol for money exchange, namely notify both the sender and the receiver of the transaction start, exchange money and finally acknowledge sender of successful money reception at the receiver side. The communication device sends a notification message to the sender with the transaction details to prepare the sender for the start of a new transaction. The sender purse should be ready to start a new transaction by being in the state “eaFrom”. The goal NotifyReciever prepares the receiving purse to start a new transaction and ensures that the receiving purse is in the “eaTo” status. The ExchangeMoney goal ensures that if the sending purse’s balance is sufficient to cover the requested amount of money, the required amount is deducted from the sender’s balance and added to the receiver’s balance. The final step in the communication protocol is formalized in the goal AcknowledgeReception that lets the receiving purse sends an acknowledgement to the sender once the receiver obtains the required money value. The four subgoals are defined as follows:



**Goal** Achieve [NotifySender]

**Informal Def** Notifies the sender purse of the beginning of the transaction

**FormalDef**

$\forall sender : Purse, startFrom : Message, cd : CommunicationDevice$

$Sending(cd, startFrom, sender) \wedge startFrom.type = "startFrom" \wedge$

$sender.Status = "eaFrom" \Rightarrow SenderStartedTransaction(sender)$

**Goal** Achieve [NotifyReceiver]

**Informal Def** Notifies the receiving purse of the beginning of the transaction

**FormalDef**

$\forall receiver : Purse, startTo : Message, cd : CommunicationDevice$

$\mathrm{Re}\,ceiving(cd, startTo, receiver) \wedge startTo.type = "startTo" \wedge$

$sender.Status = "eaTo" \Rightarrow \mathrm{Re}\,ceiverStartedTransaction(receiver)$

**Goal** Achieve [ExchangeMoney]

**Informal Def** Exchanges money between the sender and the receiving purses

**FormalDef**

$\forall sender, receiver : Purse, value : \mathrm{Re}\,al$

$sender.balance \geq value \Rightarrow DeductBalance(sender, value) \wedge IncrementBalance(receiver, value)$

**Goal** Achieve [AcknowledgeReception]

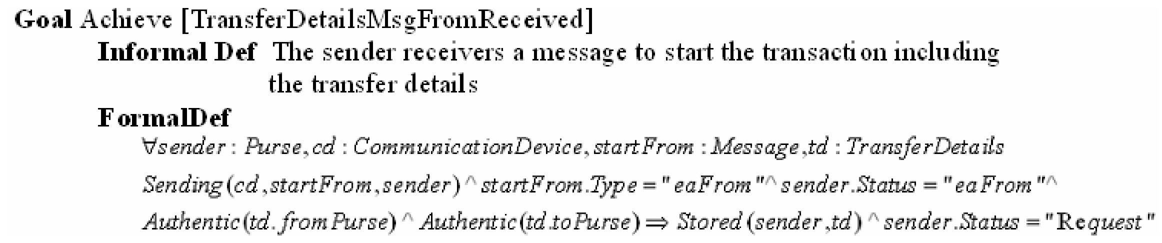
**Informal Def** The receiver sends an acknowledgement to the sender of a successful money transfer

**FormalDef**

$\forall sender, receiver : Purse, value : \mathrm{Re}\,al, ack : Message$

$\mathrm{Re}\,ceived(receiver, value) \Rightarrow Sending(receiver, sender, ack)$

The goal NotifySender is further refined into the goal TransferDetailsMsgFromReceived that describes how the sending purse should behave when it receives a startFrom message indicating the start of a new transaction in which the purse will play the role of a payer.



**Goal** Achieve [TransferDetailsMsgFromReceived]

**Informal Def** The sender receivers a message to start the transaction including the transfer details

**FormalDef**

$\forall sender : Purse, cd : CommunicationDevice, startFrom : Message, td : TransferDetails$

$Sending(cd, startFrom, sender) \wedge startFrom.Type = "eaFrom" \wedge sender.Status = "eaFrom" \wedge$

$Authentic(td.fromPurse) \wedge Authentic(td.toPurse) \Rightarrow Stored(sender, td) \wedge sender.Status = "\mathrm{Re}\,quest"$

The goal NotifyReceiver is refined into the goal TransferDetailsMsgToReceived that illustrates the behavior of the receiving purse upon reception of a startTo message recording the beginning of a new transaction in which the purse will play the role of a payee.

**Goal** Achieve [TransferDetailsMsgToReceived]

**Informal Def** The receiving purse receives a message to start the transaction including the transfer details

**FormalDef**

$\forall receiver : Purse, cd : CommunicationDevice, startTo : Message, td : TransferDetails$

$Sending(cd, startFrom, receiver) \wedge startTo.Type = "eaTo" \wedge receiver.Status = "eaTo" \wedge$

$Authentic(td.fromPurse) \wedge Authentic(td.toPurse) \Rightarrow Stored(receiver, td) \wedge$

$sender.Status = "Value" \wedge Sending(receiver, td, cd)$

The goal ExchangeMoney is refined into two subgoals, one for the sending purse to do the actual money deduction from its balance and the other for the receiving purse to obtain the money and increment its balance with the requested value. The goal TransferRequestReceived is concerned with the sending purse receiving a transfer request message from the communication device. If the request message is valid, the sending purse deducts the money value from its balance. The goal TransferValueMsgReceived is concerned with the receiving purse getting a value message from the communication device indicating that the requested money value has been transferred in which case the receiving purse adds the money to its balance. The definition of the two goals that refine the ExchangeMoney goal is:

**Goal** Achieve [TransferRequestReceived]

**Informal Def** The sending purse receives a request message to do the actual money transfer

**FormalDef**

$\forall sender : Purse, cd : CommunicationDevice, request : Message, td : TransferDetails$

$Sending(cd, request, sender) \wedge request.type = "request" \wedge sender.Status = "request" \wedge$

$td = sender.TransferDetails \Rightarrow DeductBalance(sender, td.value) \wedge Sending(sender, td, cd)$

**Goal** Achieve [TransferValueMsgReceived]

**Informal Def** The receiving purse receives a value message to increment its balance with the requested money

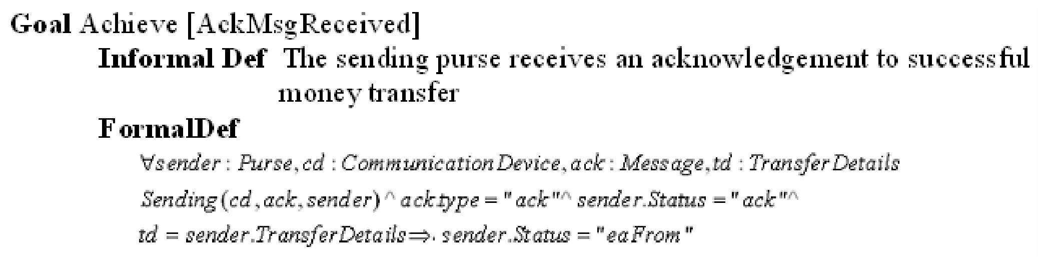
**FormalDef**

$\forall receiver : Purse, cd : CommunicationDevice, value, ack : Message, td : TransferDetails$

$Sending(cd, value, receiver) \wedge value.type = "value" \wedge receiver.Status = "value" \wedge$

$td = receiver.TransferDetails \Rightarrow IncrementBalance(receiver, td.value) \wedge Sending(receiver, ack, cd)$

The goal AcknowledgeReception is refined into the goal AckMsgReceived that handles the reception of an acknowledgement message at the sending purse side in order to finalize the transaction and put the sending purse in the ready state for a new transaction.



**Goal** Achieve [AckMsgReceived]

**Informal Def** The sending purse receives an acknowledgement to successful money transfer

**FormalDef**

$\forall sender : Purse, cd : CommunicationDevice, ack : Message, td : TransferDetails$

$Sending(cd, ack, sender) \wedge acktype = "ack" \wedge sender.Status = "ack" \wedge$

$td = sender.TransferDetails \Rightarrow \cdot\ sender.Status = "eaFrom"$

Figure 27 illustrates the goal graph for the SecureTransferProtocol goal and its subgoals

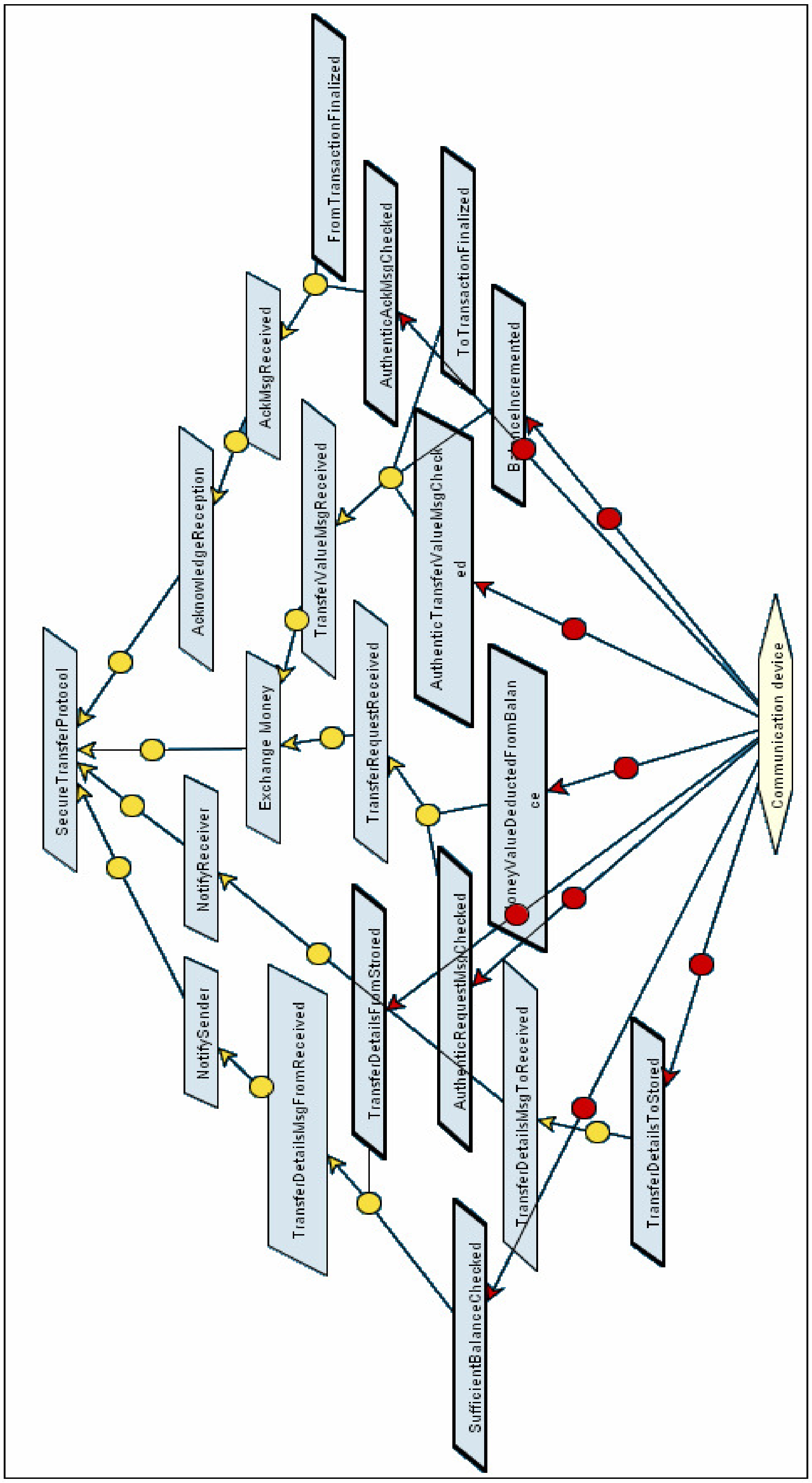


**Figure 27: SecureTransferProtocol Goal Refinement**

The TransactionAborted goal is refined into two goals to store the transaction details into the purse exception log and to put the sending and receiving purses in the ready state for new transactions. Figure 28 illustrates the refinement of the TransactionAborted goal.

**Goal** Achieve [IncompleteTransactionDetailsStored]
**Informal Def** Stores the transfer details of the transaction in the purse exception log
**Formal Def**

$$\forall sender, receiver : Purse, td : TransferDetails$$
$$TransactionAborted(sender, reveiver, td) \Rightarrow$$
$$Stores(sender.ExceptionLog, td) \wedge Stores(receiver.ExceptionLog, td)$$

**Goal** Achieve [PurseReadyForNewTransaction]
**Informal Def** Puts the sending and receiving purses in the ready state to start new transaction
**Formal Def**

$$\forall sender, receiver : Purse, td : TransferDetails$$
$$TransactionAborted(sender, reveiver, td) \wedge Stored(sender.ExceptionLog, td) \wedge$$
$$Stored(receiver.ExceptionLog, td) \Rightarrow sender.Status = "eaFrom" \wedge receiver.Status = "eaTo"$$

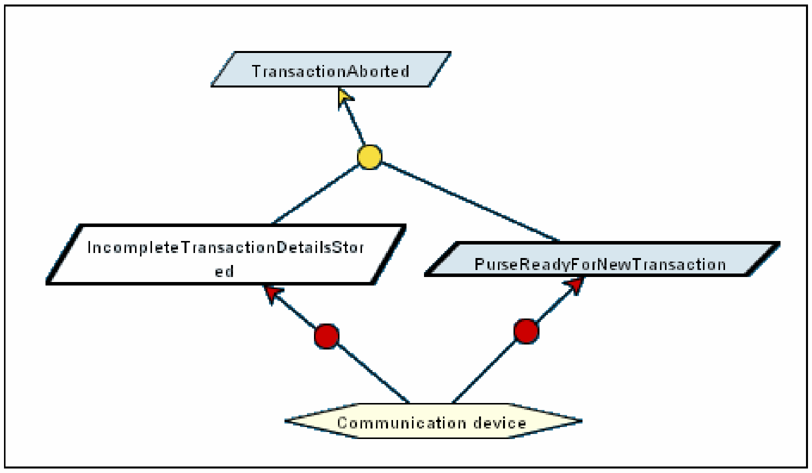

TransactionAborted
IncompleteTransactionDetailsStored
PurseReadyForNewTransaction
Communication device

**Figure 28: Refinement of the Goal TransactionAborted**

Stepney has used a two-step archiving technique to log the transaction details occurring in the system, the purse exception log and the central archive [6]. Each purse has an exception log in which it stores the transfer details of aborted transactions. The purse exception log is flushed every specific period of time by moving its content to the central archive that stores all the messages circulated in the system. The goal MessageLogsArchived is refined into three subgoals modeling the different functionalities provided by the two archiving steps (purse exception log and central archive) namely writing a message to the central archive, reading a message from the purse exception log and flushing the purse exception log to the central archive. The definition of the three subgoals that refine the goal MessageLogsArchived is as follows:

**Goal** Achieve [MessageWrittenToArchive]

**Informal Def** Writes a message to the central Archive

**Formal Def**

$$\forall msg : Message, archive : Archive, cd : CommunicationDevice$$
$$Sending(cd, msg, archive) \Rightarrow Stores(archive, msg)$$

**Goal** Achieve [MessageReadFromPurseExceptionLog]

**Informal Def** Reads a message from the purse exception log

**Formal Def**

$$\forall msg, resultMsg : Message, purse : Purse, cd : CommunicationDevice$$
$$msg.Type = ReadExceptionLogMessage \wedge Sending(cd, msg, purse) \Rightarrow$$
$$Read(purse.ExceptionLog, resultMsg) \wedge Sending(purse, resultMsg, cd)$$

**Goal** Achieve [ClearExceptionLogMsgReceived]

**Informal Def** Flushes the purse exception log to the central archive

**Formal Def**

$$\forall msg : Message, purse : Purse, cd : CommunicationDevice, archive : Archive$$
$$msg.Type = ExceptionLogClearMessage \wedge Sending(cd, msg, purse) \Rightarrow$$
$$Copy(purse.ExceptionLog, archive) \wedge Clear(purse.ExceptionLog)$$

Figure 29 illustrates the refinement of the goal MessageLogsArchived.

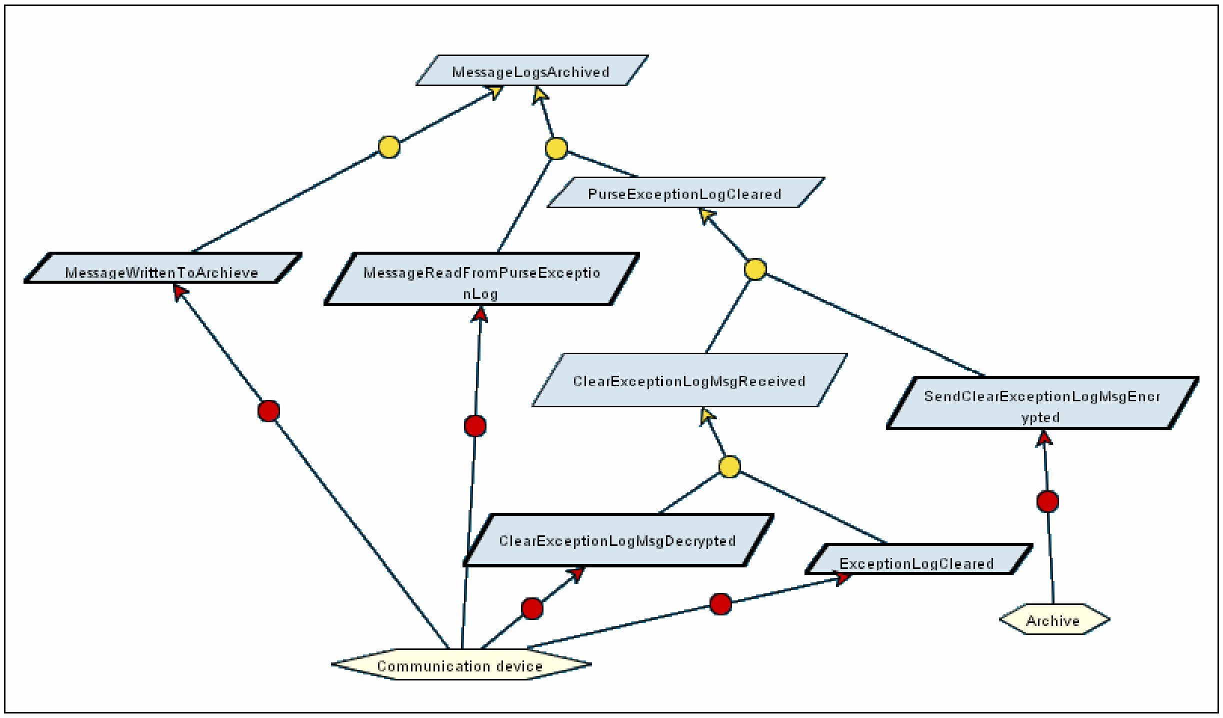

MessageLogsArchived
PurseExceptionLogCleared
MessageWrittenToArchieve
MessageReadFromPurseExceptionLog
ClearExceptionLogMsgReceived
SendClearExceptionLogMsgEncrypted
ClearExceptionLogMsgDecrypted
ExceptionLogCleared
Archive
Communication device

**Figure 29: Refinement of the Goal MessageLogsArchived**

There are two agents namely the communication device and the archive. The communication device is responsible for the mediating the communication between the payer and the payee purses since purses do not communicate directly.  The goal refinement process has resulted in a number of entities (passive objects) namely purse, message and transfer details. Stepney has categorized messages circulating in the system into protected and unprotected messages. The following message types fall in the category of protected messages:

1. *Request message*: the message sent by the communication device to the payer purse to do the actual deduction of money from its balance.
2. *Value message*: the message sent by the communication device to the payee purse to do the actual addition of money to its balance.
3. *Ack message*: the message forwarded by the communication device from the payee to the payer acknowledging successful reception of money.
4. *Exception log result message*: the message sent by the purse as including the results of reading its exception log.
5. *Exception log clear message*: the message sent by the communication device to the purse to clear its exception log and flush its content to the central archive.

The following message types fall in the category of unprotected message

1. *Start from message*: the message sent by the communication device to the payer purse indicating the start of a new transaction.
2. *Start to message*: the message sent by the communication device to the payee purse indicating the start of a new transaction.
3. *Read exception log message*: the message sent by the communication device to read a message in the purse's exception log. This message is normally sent when a transaction aborts and the purse's exception log is read for analysis of failure.

Figure 30 summarizes the object model developed during the goal elicitation stage.

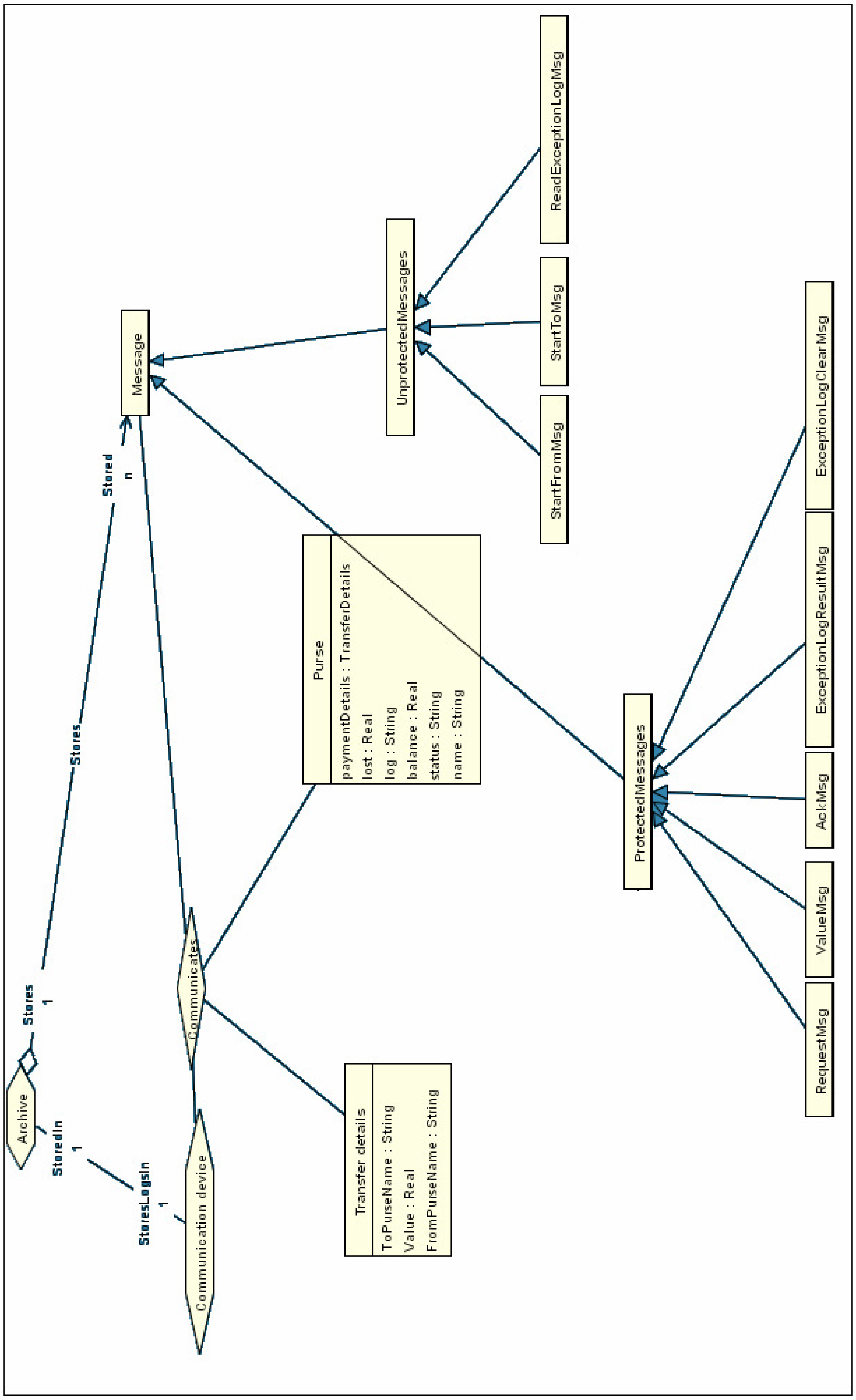


**Figure 30: Object Model for the Electronic Smart Card System**

### 4.2.3 Analysis and Resolution of Obstacles and Conflicts for Security Goals

In this subsection, the security goals elaborated in the previous section are refined and analyzed by finding conflicts and obstacles to security, that is, potential attacks. Resolution strategies to potential security attacks are proposed.

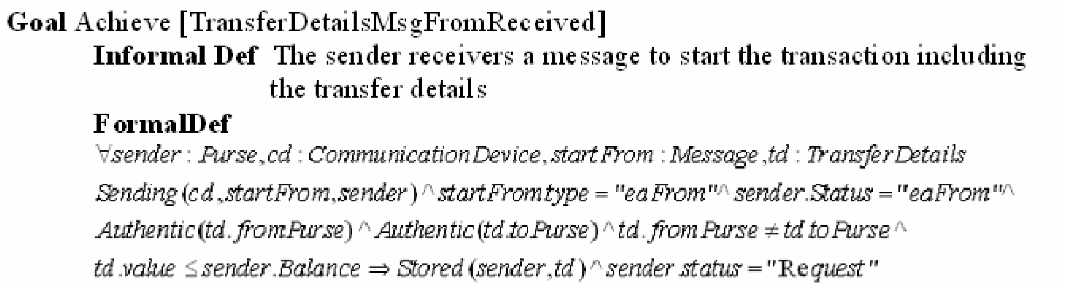
#### 4.2.3.1 Obstacle Generation and Resolution to the Goal NotifySender

The goal NotifySender is achieved by the communication device that communicates a notification start from message to start a new transaction in which the purse acts as a payer. Messages are sent in plain text over the communication line and might be tampered with while in transit, which threats the message integrity. The notification message has to be valid in order for the payer purse to start a new transaction. A start from message is valid if the payment details it carries specify a payer and payee purses that are different from each other and belong to the set of authentic purses. Further, a valid start from message has to specify an amount of money for exchange between payer and payee that is less than or equal the balance of the payer purse. The following two obstacles have been generated to the goal NotifySender.

**Obstacle** InvalidStartFromMessgeSent
    **Informal Def** An invalid start from message is received by the payer
    **Formal Def**
        $\exists msg : Message, payer : Purse$
        $\mathrm{Re}\,ceived\,(payer, msg)\ \wedge\ msg.type = "eaFrom"\ \wedge$
        $msg.TransferDetails.ToPurse = payer.Name$

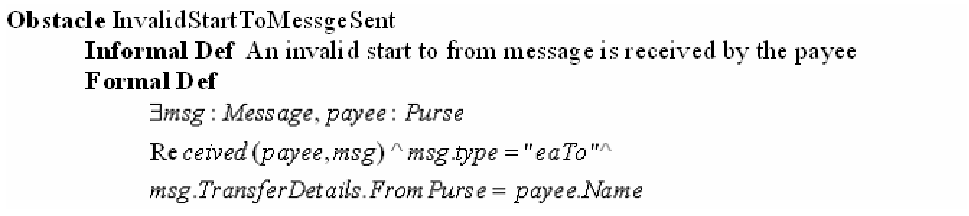
**Obstacle** InsufficientBalance
    **Informal Def** Start from message specifies an amount of money greater than the payer current balance
    **Formal Def**
        $\exists msg : Message, payer : Purse$
        $\mathrm{Re}\,ceived\,(payer, msg)\ \wedge\ msg.type = "eaFrom"\ \wedge$
        $msg.TransferDetails.Value > payer.Balance$

A solution to the above the two obstacles is to check the validity of the start from message upon reception at the payer side before storing the payment details and change the purse status to be ready for the new transaction. The definition of the TransferDetailsMsgFromReceived goal changes to resolve the two obstacles as follows:

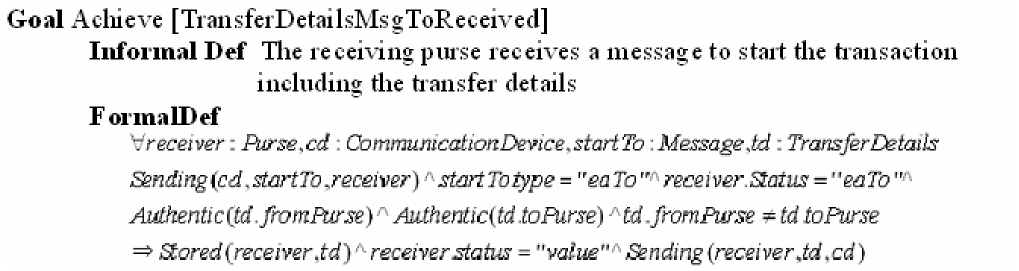

**Goal** Achieve [TransferDetailsMsgFromReceived]
**Informal Def** The sender receivers a message to start the transaction including the transfer details
**FormalDef**
$\forall sender : Purse, cd : CommunicationDevice, startFrom : Message, td : TransferDetails$
$Sending(cd, startFrom, sender) \wedge startFromtype = "eaFrom" \wedge sender.Status = "eaFrom" \wedge$
$Authentic(td.fromPurse) \wedge Authentic(td.toPurse) \wedge td.fromPurse \neq td.toPurse \wedge$
$td.value \leq sender.Balance \Rightarrow Stored(sender, td) \wedge sender.status = "Request"$

## 4.2.3.2 Obstacle Generation and Resolution to the Goal NotifyReceiver

The payee receives a start to message from the communication device in order to achieve the goal NotifyReceiver that notifies the payee of the new transaction start. A valid start to message specifies a payer purse different from the payee purse. Therefore, if the message content is altered while in transit, the start to message becomes invalid. The following obstacle is generated to the goal NotifyReceiver:


**Obstacle** InvalidStartToMessgeSent
**Informal Def** An invalid start to from message is received by the payee
**Formal Def**
$\exists msg : Message, payee : Purse$
$Received(payee, msg) \wedge msg.type = "eaTo" \wedge$
$msg.TransferDetails.FromPurse = payee.Name$

The payee needs to add an extra check for the validity of the start to message upon reception from the communication device and before storing the payment details. The goal TransferDetailsMsgToReceived is modified to accommodate the check for the message validity.


**Goal** Achieve [TransferDetailsMsgToReceived]
**Informal Def** The receiving purse receives a message to start the transaction including the transfer details
**FormalDef**
$\forall receiver : Purse, cd : CommunicationDevice, startTo : Message, td : TransferDetails$
$Sending(cd, startTo, receiver) \wedge startTotype = "eaTo" \wedge receiver.Status = "eaTo" \wedge$
$Authentic(td.fromPurse) \wedge Authentic(td.toPurse) \wedge td.fromPurse \neq td.toPurse$
$\Rightarrow Stored(receiver, td) \wedge receiver.status = "value" \wedge Sending(receiver, td, cd)$

## 4.2.3.3 Obstacle Generation and Resolution to the Goal TransferRequestReceived

The payer receives a transfer request message forwarded by the communication device on behalf of the payee to do the actual money exchange by deducting the request amount from the payer’s balance. Messages are sent in plain text over the communication line that is securely unprotected. The request message that holds the payment details (the payer purse, the payee purse and the requested amount) might be maliciously intercepted or tampered with while in transit in. Interception is only passive meaning that the attacker knows the content of the message, but does not remove it and neither the sender nor the receiver knows that someone has intercepted the message. Intercepting message while in transit threats the message confidentiality. Tampering with the message might lead a malicious eavesdropper to change the content of the message in order to change the payee information or the requested amount, which threats the message integrity. The following obstacles have been generated for the goal TransferRequestReceived.

**Obstacle** InterceptedTransferRequest

**Informal Def** An attacker intercepts the transfer request message while in transit

**Formal Def**

$\exists payer : Purse, reqMsg : Message, cd : CommunicationDevice, attacker : Person$

$Sending(cd, reqMsg, payer) \wedge Received(payer, reqMsg) \wedge Knows(attacker, reqMsg.Content)$

**Obstacle** CorruptedTransferRequest

**Informal Def** An attacker alters the transfer request message content while in transit

**Formal Def**

$\exists payer : Purse, reqMsgOriginal, reqMsgCorrupted : Message, cd : CommunicationDevice, attacker : Person$

$Sending(cd, reqMsg, payer) \wedge Received(payer, reqMsg) \wedge CopyOf(reqMsgCorrupted, reqMsgOriginal) \wedge$

$reqMsgOriginal.Content \neq reqMsgCorrupted.Content$

A way to resolve the obstacle InterceptedTransferRequest and protect the request message confidentiality is to encrypt the message with a secret key using any of the standard encryption algorithms. The payee encrypts the request message and sends it to the communication device that in turn forwards the message to the payer who decrypts the message. The following two goals are introduced in order to resolve the request message confidentiality obstacle:

**Goal** Achieve [RequestMessageEncrypted]

**Informal Def** The payee encrypts the request message before sending to the payer

**Formal Def**

$\forall payee : Purse, reqMsg : Message, cd : CommunicationDevice$

$Sending(payee, reqMsg, cd) \Rightarrow Encrypted(reqMsg)$

**Goal** Achieve [RequestMessageDecrypted]

**Informal Def** The payer decrypts the request message upon reception

**Formal Def**

$\forall payer : Purse, reqMsg : Message, cd : CommunicationDevice$

$Decrypted(reqMsg) \wedge Receiving(payer, reqMsg, cd)$

The integrity of the request message could be protected using digital signatures. Different techniques for digital signatures exist in the literature such as one way hashing and public key infrastructure. The payee needs to sign the request message before sending it to the communication device while the payer needs to verify the request message upon reception from the communication device. The payer should proceed with processing the transfer request if and only if the message verification succeeds. The following two goals are introduced in order to protect the request message integrity.

**Goal** Achieve [RequestMessageSigned]
**Informal Def** The payee signs the request message before sending it to the payer
**Formal Def**
$\forall payee : Purse(\exists key : Key), reqMsg : Message, cd : CommunicationDevice$
$Sending(payee, reqMsg, cd) \wedge Owning(payee, key) \Rightarrow Signed(reqMsg, key)$

**Goal** Achieve [RequestMessageVerified]
**Informal Def** The payer verifies the request message before opening it
**Formal Def**
$\forall payer : Purse(\exists key : Key), reqMsg : Message, cd : CommunicationDevice$
$\mathrm{Re}\,ceiving(payer, reqMsg, cd) \wedge Owning(payer, key) \Rightarrow Verify(reqMsg, key)$

Notice that the payee will encrypt the request message first before signing it while the payer will verify the message first before decrypting it.

### 4.2.3.4 Obstacle Generation and Resolution to the Goal TransferValueMsgReceived

The payer sends a transfer value message to the communication device at the end of processing the transfer request. Upon reception of a valid transfer value message, the payee adds the money value to its balance. The goal TransferValueMsgReceived might be obstructed by the same obstacles of the goal TransferRequestReceived since the value message might be intercepted or tampered with while in transit. The resolution strategy to the confidentiality and integrity obstacles is the same as the strategy for resolving the same obstacles for the goal TransferRequestReceived. Encryption/decryption techniques are needed to protect the confidentiality of the value message while digital signatures are needed to protect the message integrity. The details of the obstacles to the goal TransferValueMsgReceived are not described in details because they are similar to the ones descried in the previous section for the goal TransferRequestReceived.

### 4.2.3.5 Obstacle Generation and Resolution to the Goal AcknowledgeReception

The payee sends an acknowledgment message to the communication device to acknowledge the reception of money and finalize the transaction. The communication device forwards the ack message to the payer purse in order to end the transaction and put itself in a state ready for a new transaction. The acknowledgement message might be exposed to the same obstacles as the transfer request and transfer value messages described in the previous two sections. The confidentiality and integrity obstacles to the goal AckMessageReceived are not described in details because of their similarity their counterparts to the goals TransferRequestReceived and TransferValueMsgReceived.

The acknowledgment message is assumed to be delivered within a specific timeframe. This means that if the ack message is lost in transit or delayed beyond the timeframe of delivery, the goal AcknowledgetReception will not be achieved. The following obstacle to the goal AcknowledgetReception describes the situation.

**Obstacle** AckMsgNotReceivedWithinTimeLimit

**Informal Def** The ack message is not delivered to the payer within a specific timeframe

**Formal Def**

$$\forall payer : Purse, ackMsg : Message, cd : CommunicationDevice, timeout : \mathrm{Real}$$

$$\Box_{>timeout}\, Sending(cd, ackMsg, payer) \Rightarrow \neg\, \mathrm{Re}ceiving(payer, ackMsg)$$

This obstacle is resolved by aborting the transaction when the ack message is either lost or delayed beyond the specified timeout period. Aborting the transaction provides a safe solution to the obstacle because this allows money to be lost securely with all the messages of the transactions logged for traceability.

Figure 31 summarizes the obstacles and the resolution goals. Notice that the pink boxes in Figure 30 represent the obstacles while the blue ones represent the goals. The arrows with green heads link each obstacle to the goal(s) that resolves it.

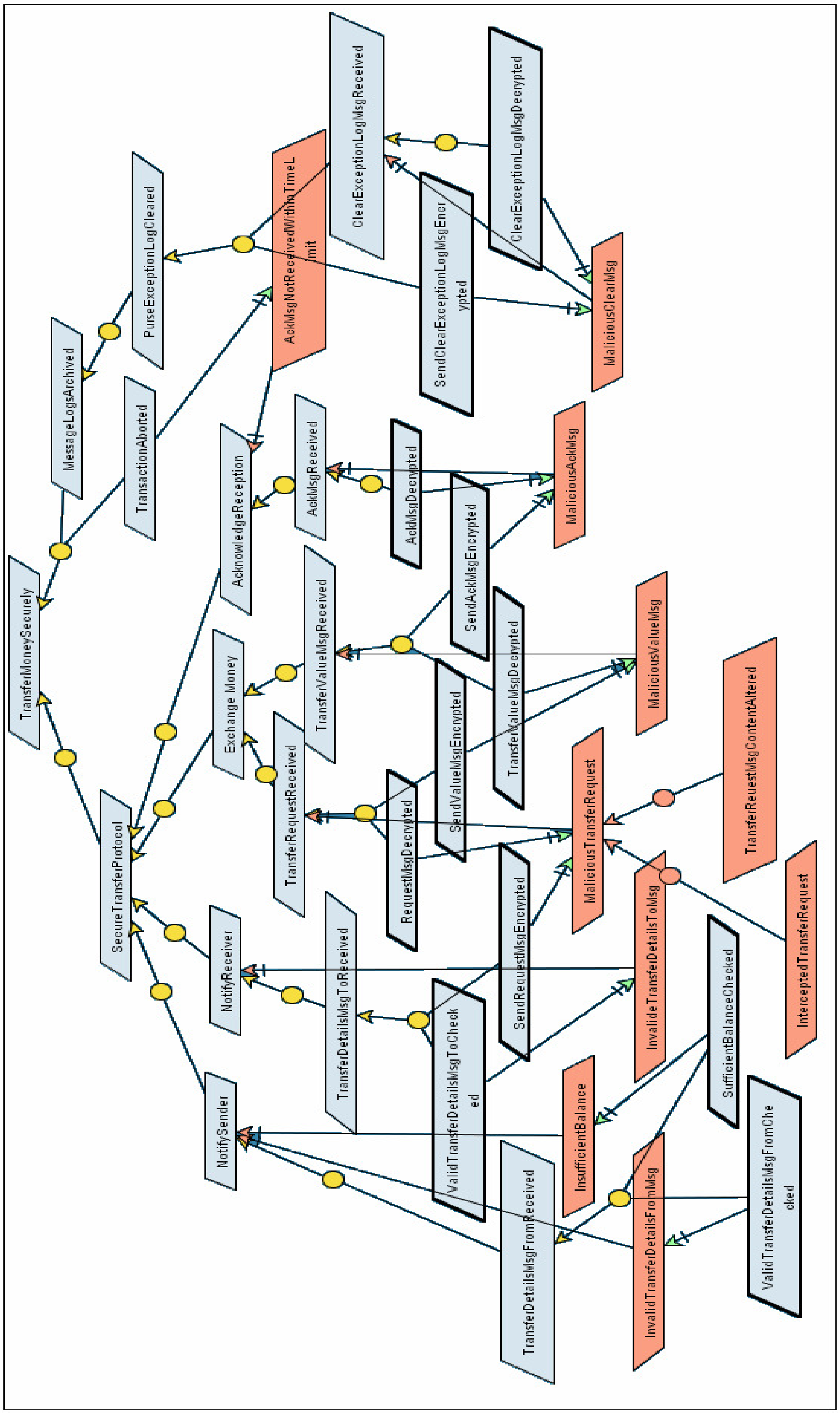

TransferMoneySecurely
MessageLogsArchived
PurseExceptionLogCleared
TransactionAborted
AckMsgNotReceivedWithinTimeL mit
ClearExceptionLogMsgReceived
ClearExceptionLogMsgDecrypted
SendClearExceptionLogMsgEncr ypted
MaliciousClearMsg
AcknowledgeReception
AckMsgReceived
AckMsgDecrypted
SendAckMsgEncrypted
MaliciousAckMsg
SecureTransferProtocol
Exchange Money
TransferValueMsgReceived
TransferValueMsgDecrypted
SendValueMsgEncrypted
MaliciousValueMsg
TransferRequestReceived
RequestMsgDecrypted
MaliciousTransferRequest
TransferReuestMsgContentAltered
InterceptedTransferRequest
NotifyReceiver
TransferDetailsMsgToReceived
SendRequestMsgEncrypted
ValidTransferDetailsMsgToCheck ed
InvalideTransferDetailsToMsg
SufficientBalanceChecked
InsufficientBalance
NotifySender
TransferDetailsMsgFromReceived
InvalidTransferDetailsFromMsg
ValidTransferDetailsMsgFromChe cked

**Figure 31: Obstacle Analysis and Resolution Goals for the Electronic Smart Card System**

### 4.2.4 Goal Operationalization

Leaf goals of the goal graph are assigned to agents to achieve them through means of operations specified during the goal operationalization stage. FADES is concerned with the KAOS operations specified for leaf goals since these are the main constructs transformed to B to build the initial B model for further refinement towards design and implementation. The operations are highlighted and formally defined in what follows. Figure 32 and 33 illustrates the goal graph and the operations. The goal graph is split into two Figures because of the space limitation. The following operations have been defined for the system. Each operation has a name, input parameters, output parameters, precondition, and goals to which the operation is a pre-requisite.

The startFromPurse operation is responsible for achieving the goal NotifySender and its subgoals. It should execute when the payer purse receives a message of type startFrom from the communication device in order to start a new transaction. The operation gets the message and payment details as input parameters and should send a message when it finishes to the communication device. The post condition of the operation indicates that the payer changes its state to the “Request” state meaning that is ready to send the money when the payee is notified with the transaction start.

**Operation** startFromPurse
**Input** Message{arg startFromMsg}, PaymentDetails{arg pd}
**Output** Message {res msg}
**DomPre**

startFromMsg.type = eaFrom ^ pd.fromPurse ∈ AuthenticPurses ^
pd.toPurse ∈ AuthenticPurses ^ pd.fromPurse ≠ pd.toPurse ^
pd.fromPurse.Status = "eaFrom" ^ pd.value ≤ pd.fromPurse.Balance

**DomPost** Stored (pd.fromPurse, pd) ^ (∃ cd: CommunicationDevice)Sending(msg, cd) ^
Pd.fromPurse.Status = “Request”
**ReqPreFor** NotifyReceiver
**ReqPreFor** ExchangeMoney

The startToPurse operation achieves the goal NotifyReceiver and its subgoals. The operation is assumed to execute when the payee purse receives a message of type startTo from the communication device to start a new transaction. The operation input parameters are the startTo message and the payment details. The operation is supposed to send a request message to the communication device that forwards it in turn to the payer purse to send the money. The operation post condition indicates that payee purse status changes to the “Value” state meaning that it is ready to receive the money.

**Operation** startToPurse

**Input** Message{arg startToMsg}, PaymentDetails{arg pd}

**Output** Message {res msg}

**DomPre**

startToMsg.type = eaTo ^ pd.fromPurse $\in$ AuthenticPurses ^
pd.toPurse $\in$ AuthenticPurses ^ pd.fromPurse $\neq$ pd.toPurse ^
pd.toPurse.Status = "eaFrom"

**DomPost** Stored (pd.toPurse, pd) ^ ($\exists$ cd: CommunicationDevice)Sending(msg, cd) ^
Pd.toPurse.Status = “Value”

**ReqPreFor** TransferRequestReceived
Sending(reqMsg, pd.fromPurse)

The SendMoneyValue operation achieves the TransferRequestReceived goal and its subgoal. The operation is executed when the payer purse receives a request message in which case, it should deduct the required money if it has sufficient funds. The output of the operation is a “Value” message sent to the communication device that forwards it in turn to the payee purse in order to add the money to its balance. The post condition of this operation indicates that the status of the payer purse changes to “ack” meaning that it waits for an acknowledgment of money reception from the payee purse.

**Operation** SendMoneyValue

**Input** Message{arg requestMsg}, Purse{arg fromPurse}

**Output** Message {res msg}

**DomPre**

verifieded(reqMsg) ^ decrypted(reqMsg) ^ reqMsg.type = Request ^
fromPurse $\in$ AuthenticPurses ^ reqMsg.Content = fromPurse.paymentDetails ^
fromPurse.Status = "Request"

**DomPost** deducted(fromPurse.paymentDetails.Balance) ^ ($\exists$ cd: CommunicationDevice)Sending(msg, cd) ^ fromPurse.Status = “Ack”

**ReqPreFor** TransferValueMsgReceived
Sending(valueMsg, fromPurse.paymentDetails.toPurse)

The ReceiveMoneyValue operation achieves the goal TransferValueMsgReceived and its subgoals. The operation is executed when the payee purse receives a value message from the communication device on behalf of the payer purse. The operation gets the value message as an input parameter. At the end of its execution, the operation should send an acknowledgement message to the communication device to forward to the payer purse acknowledging money reception. The post condition of the operation indicates that the operation changes the status of the payee purse to “eaTo” meaning that the transaction is finalized from the payee side and the purse is ready for a new transaction.

**Operation** ReceiveMoneyValue

**Input** Message{arg valueMsg}, Purse{arg toPurse}

**Output** Message {res msg}

**DomPre**

verifieded(valueMsg) ^ decrypted(valueMsg) ^ valueMsg.type = Value ^
toPurse $\in$ AuthenticPurses ^ valueMsg.Content = toPurse.paymentDetails ^
toPurse.Status = "Value"

**DomPost** Incremented(fromPurse.paymentDetails.Balance) ^

($\exists$ cd: CommunicationDevice)Sending(msg, cd) ^ toPurse.Status = “eaTo”

**ReqPreFor** AckMsgReceived

Sending(ackMsg, toPurse.paymentDetails.fromPurse)

The ackPurse operation achieves the goal AckMsgReceived and its subgoals. The operation executes when the payer purse receives an ack message from the communication device. The operation should send a message as its output to the communication device indicating the finalization of the transaction. The post condition of the operation indicates that the operation changes the status of the payer purse to “eaFrom” meaning that the purse has finalized the current transaction and is ready for a new one.

**Operation** ackPurse

**Input** Message{arg ackMsg}, Purse{arg fromPurse}

**Output** Message {res msg}

**DomPre**

verifieded(ackMsg) ^ decrypted(ackMsg) ^ ackMsg.type = Ack ^
fromPurse $\in$ AuthenticPurses ^ ackMsg.Content = fromPurse.paymentDetails ^
fromPurse.Status = "Ack"

**DomPost** ($\exists$ cd: CommunicationDevice)Sending(msg, cd) ^ fromPurse.Status = “eaFrom”

The abortTransaction operation achieves the goal TransactionAborted and its subgoals. The operation is executed when any of the messages in the secure transfer protocol is not successfully delivered to the concerned purse in which case, the purse calls the abortTransaction operation in order to lose money securely and log all the details of the aborted transaction in its exception log. The post condition of the operation indicates that purse returns to the state of “eaFrom” in order to be ready for a new transaction.

**Operation** abortTransaction

**Input** Purse{arg purse}

**DomPre**

$purse \in AuthenticPurses$

**DomPost** purse.ExceptionLog = purse.ExceptionLog U purse.PaymentDetails ^ purse.Status = "eaFrom"

The readExceptionLog operation achieves the MessageReadFromPurseExceptionLog goal and its subgoals. The operation executes when a readExceptionLogMsg is received by any purse from the communication device. The operation is supposed to retrieve the content of the purse exception log that contains details about failed transactions. The output of the operation is a message containing the content of the purse's exception log.

**Operation** readExceptionLog

**Input** Message{arg readExceptionLogMsg}, Purse{arg purse}

**Output** Message {res msg}

**DomPre**

$readExceptionLogMsg.type = readExceptionLog \wedge$
$readExceptionLogMsg.Content \neq \emptyset \wedge purse \in AuthenticPurses$

**DomPost**

Msg.Content = purse.ExceptionLog

The operation clearPurseExceptionLog achieves the goal ClearExceptionLogMsgReceived and its subgoals. The operations executes when a clear exception log message is received by any purse from the communication device. The operation is supposed to flush the content of the purse's exception log that is done periodically to archive the content of purses' exception logs in the central archive. The post condition of the operation indicates that the purse's exception log become empty after the operation.

**Operation** clearPurseExceptionLog

**Input** Message{arg clearExceptionLogMsg}, Purse{arg purse}, String {arg recordsId}

**Output** Message {res msg}

**DomPre**

$purse.ExceptionLog \neq \emptyset \wedge clearExceptionLogMsg.type = clearExceptionLog \wedge$
$clearExceptionLogMsg.Content \neq \emptyset$

**DomPost**

$purse.ExceptionLog = \emptyset$

The following set of operations is specified to protect the security aspects of the messages being communicated in the electronic smart card system. They have been derived from the security patterns defined by Van Lamsweerde [1]. The SignMessage and VerifyMessage operations are specified to protect the integrity of the messages through digital signatures. The EncryptMessage and DecyrptMessage operations are supposed to protect the message confidentiality.

**Operation** SignMessage
**Input** Message{arg msg}, Key {arg key}
**Output** Signature {res sig}
**DomPre** $\neg$($\exists$ msg1:Message) Signed(msg1, sig, key)
**DomPost** ($\exists$ msg2:Message) Signed(msg2, sig, key)
**ReqPreFor** RequestMsgSigned, ValueMsgSigned, AckMsgSigned, ClearExceptionLogMsgSigned

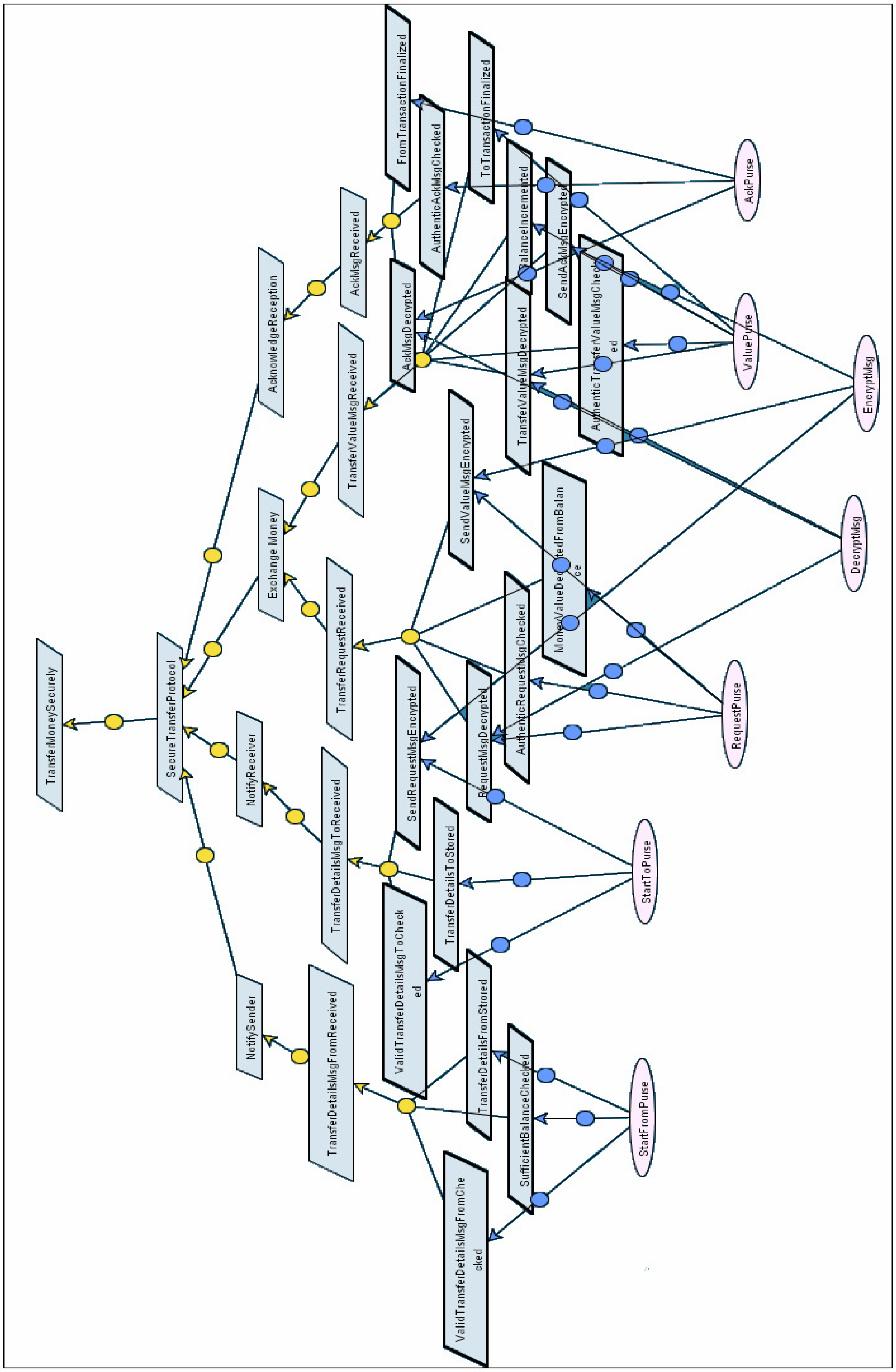
**Operation** VerifyMessage
**Input** Message{arg msg}, Key {arg key}, Signature {arg sig}
**Output** Boolean {res verified}
**DomPre** $\neg$($\exists$ msg1: Message) Verified(msg1, sig, key)
**DomPost** ($\exists$ msg2: Message) Verified(msg2, sig, key)
**ReqPreFor** RevelationVerifiedWhenReceived
Knows(sp, pk)
**ReqPreFor** RequestMsgVerified, ValueMsgVerified, AckMsgVerified, ClearExceptionLogMsgVerified

**Operation** EncryptMessage
**Input** Message{arg msg}, Key {arg key}
**Output** Message {res msg}
**DomPre** $\neg$($\exists$ msg1:Message) Encrypted(msg1, key)
**DomPost** ($\exists$ msg2:Message) Encrypted(msg2, key)
**ReqPreFor** RequestMsgEncrypted, ValueMsgEncrypted, AckMsgEncrypted,
ClearExceptionLogMsgEncrypted

**Operation** DecryptMessage
**Input** Message{arg msg}, Key {arg key}
**Output** Message {res decryptedMsg}
**DomPre** $\neg$($\exists$ msg1: Message) Decrypted(msg1, key)
**DomPost** ($\exists$ msg2: Message) Decrypted(msg2, key)
**ReqPreFor** RequestMsgDecrypted, ValueMsgDecrypted, AckMsgDecrypted,

ClearExceptionLogMsgDecrypted

KAOS operations are transformed to B in the coming subsection to construct the initial B machine that is further refined inside B using the B refinement mechanism to derive design specifications and generate implementation.



TransferMoneySecurely
SecureTransferProtocol
AcknowledgeReception
Exchange Money
NotifyReceiver
NotifySender
AckMsgReceived
TransferValueMsgReceived
TransferRequestReceived
TransferDetailsMsgToReceived
TransferDetailsMsgFromReceived
FromTransactionFinalized
AuthenticAckMsgChecked
ToTransactionFinalized
AckMsgDecrypted
BalanceIncremented
SendAckMsgEncrypted
TransferValueMsgDecrypted
AuthenticTransferValueMsgChecked
SendValueMsgEncrypted
MoneyValueDeductedFromBalance
AuthenticRequestMsgChecked
SendRequestMsgEncrypted
RequestMsgDecrypted
TransferDetailsToStored
ValidTransferDetailsMsgToChecked
TransferDetailsFromStored
SufficientBalanceChecked
ValidTransferDetailsMsgFromChecked
AckPurse
ValuePurse
EncryptMsg
DecryptMsg
RequestPurse
StartToPurse
StartFromPurse

**Figure 32: Part of the Final Goal Graph for the Electronic Smart Card System**

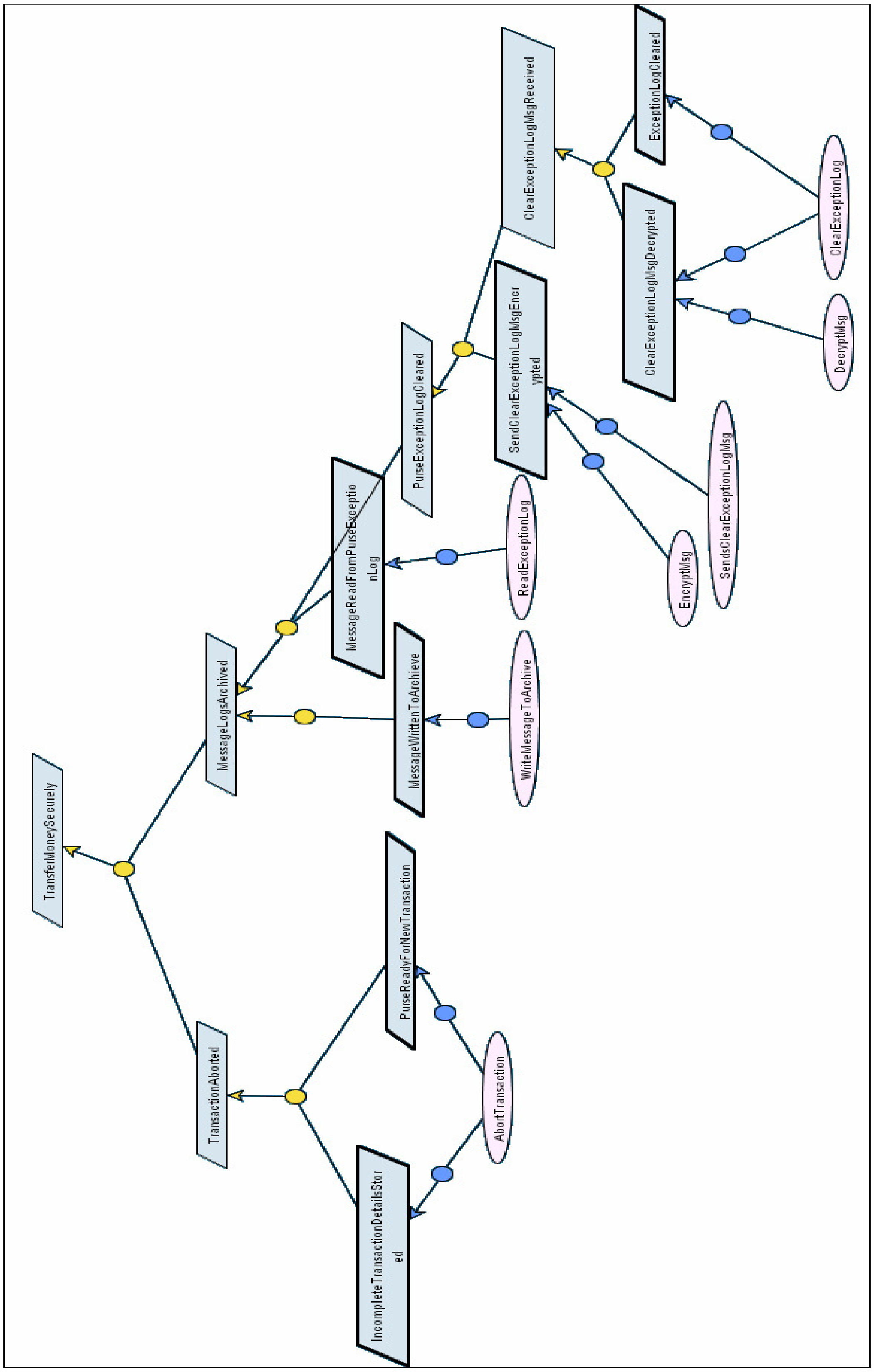


**Figure 33: Part of the Final Goal Graph for the Electronic Smart Card System**

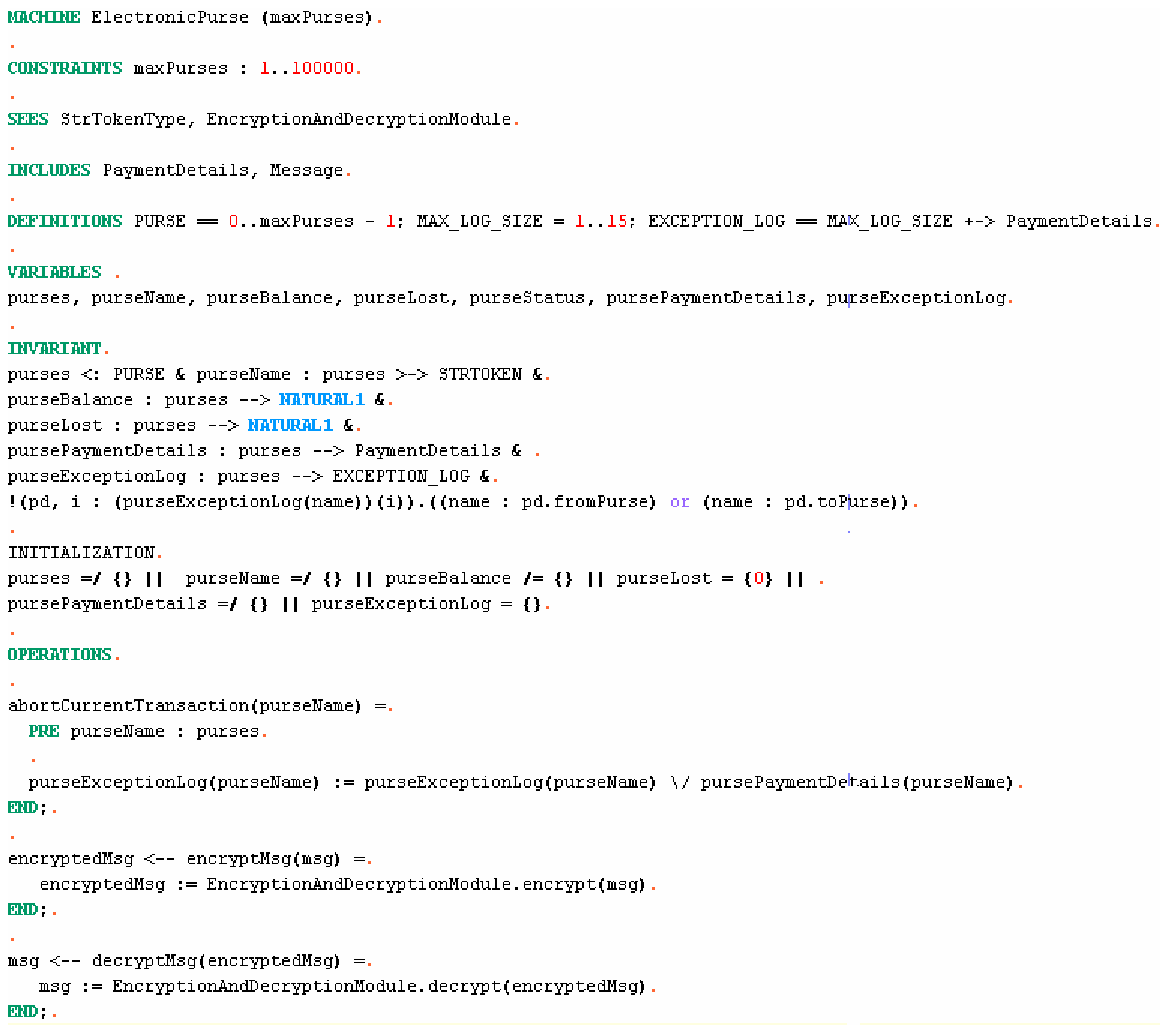
### 4.2.5 Transforming the Electronic Smart Card Security Goal Graph to B

This subsection outlines the transformation of the KAOS security goal graph built in the previous section for the electronic smart card system to the B language. The transformation scheme, as indicated earlier provides a means to extend the semi-formal KAOS requirements model with extra formality to the B language for formal design elaboration. The contribution of the transformation scheme in FADES is focused on stepping from requirements to design while allowing variable level of formality at each of the two stages.

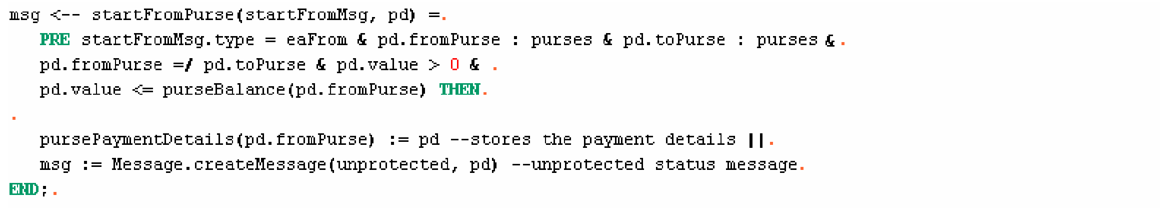
The transformation focuses on both the KAOS security operations and the entities of the partial object model involved in security goals. The ElectronicPurse machine represents the system in which each KAOS operation is represented as a B operation. KAOS operations preconditions are mapped to preconditions of their corresponding B operations. The entities PaymentDetails and Message are represented as separate machines included in the system machine. For further details on the transformation scheme, refer to section 3.2.1. The included machines are part of the including machine but structured in separate module for better design structure of the system.

The ElectronicPurse, PaymentDetails, and Message machines have been generated from the Goal Graph Analyzer through parsing the XML model of the KAOS requirements for the electronic smart card system while automatically applying the transformation rules. The generated machines are then augmented with abstract specifications for each operation since the abstract definition is part of the design process and does not exist in the KAOS model.

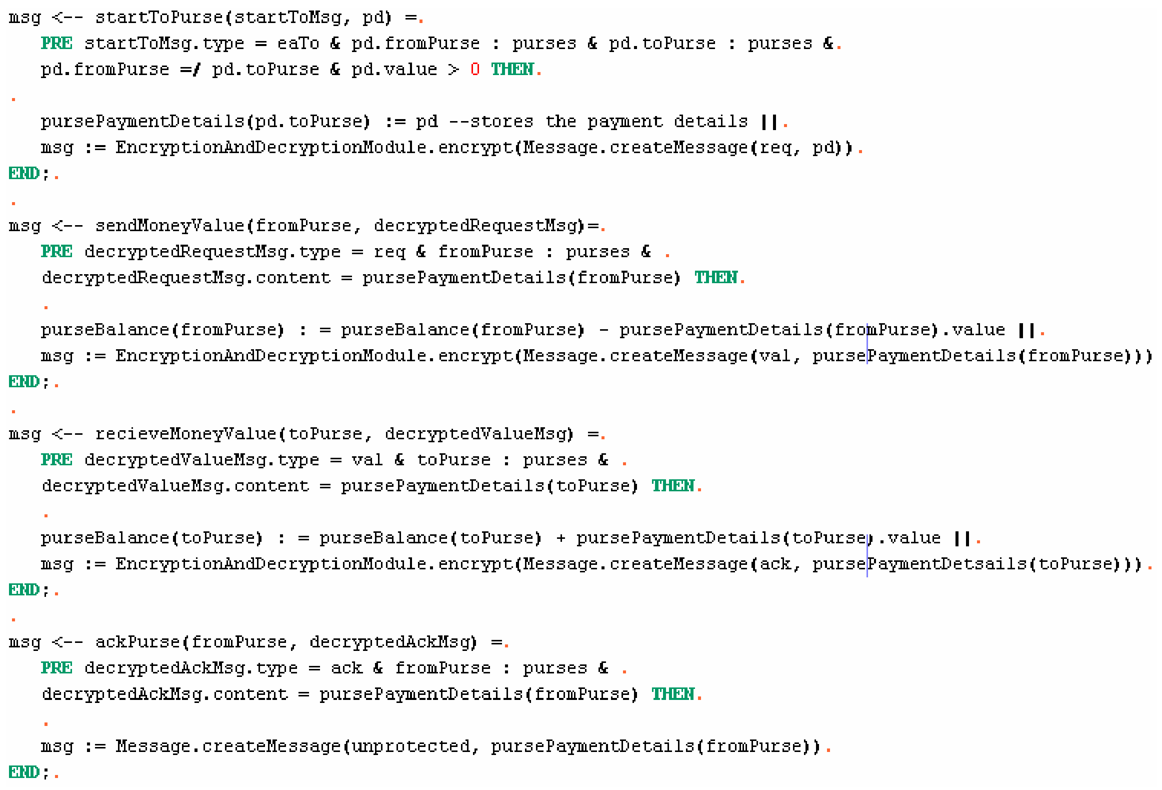
The ElectronicPurse machine is parameterized with the maximum number of purses allowed in the system. The machine includes the B machines defining the entities namely Message and PaymentDetails. The ElectronicPurse machine has a set of variables representing the set of purses in the system and their attributes (name, balance, status, exception log, and payment details). The invariant specifies the types of the machine variables and stating the constraints on these variables. The machine invariant is obtained from the invariant of the KAOS purse entity. The operations of the machine include a representation of each KAOS operation along with its precondition. Operation’s precondition also specifies the types of all the operation parameters.



```
MACHINE ElectronicPurse (maxPurses).
.
CONSTRAINTS maxPurses : 1..100000.
.
SEES StrTokenType, EncryptionAndDecryptionModule.
.
INCLUDES PaymentDetails, Message.
.
DEFINITIONS PURSE == 0..maxPurses - 1; MAX_LOG_SIZE = 1..15; EXCEPTION_LOG == MAX_LOG_SIZE +-> PaymentDetails.
.
VARIABLES .
purses, purseName, purseBalance, purseLost, purseStatus, pursePaymentDetails, purseExceptionLog.
.
INVARIANT.
purses <: PURSE & purseName : purses >-> STRTOKEN &.
purseBalance : purses --> NATURAL1 &.
purseLost : purses --> NATURAL1 &.
pursePaymentDetails : purses --> PaymentDetails & .
purseExceptionLog : purses --> EXCEPTION_LOG &.
!(pd, i : (purseExceptionLog(name))(i)).((name : pd.fromPurse) or (name : pd.toPurse)).
.
INITIALIZATION.
purses =/ {} ||  purseName =/ {} || purseBalance /= {} || purseLost = {0} || .
pursePaymentDetails =/ {} || purseExceptionLog = {}.
.
OPERATIONS.
.
abortCurrentTransaction(purseName) =.
  PRE purseName : purses.
  .
  purseExceptionLog(purseName) := purseExceptionLog(purseName) \/ pursePaymentDetails(purseName).
END;.
.
encryptedMsg <-- encryptMsg(msg) =.
   encryptedMsg := EncryptionAndDecryptionModule.encrypt(msg).
END;.
.
msg <-- decryptMsg(encryptedMsg) =.
   msg := EncryptionAndDecryptionModule.decrypt(encryptedMsg).
END;.
```



```
msg <-- startFromPurse(startFromMsg, pd) =.
   PRE startFromMsg.type = eaFrom & pd.fromPurse : purses & pd.toPurse : purses &.
   pd.fromPurse =/ pd.toPurse & pd.value > 0 & .
   pd.value <= purseBalance(pd.fromPurse) THEN.
.
   pursePaymentDetails(pd.fromPurse) := pd --stores the payment details ||.
   msg := Message.createMessage(unprotected, pd) --unprotected status message.
END;.
```


```
msg <-- startToPurse(startToMsg, pd) =.
   PRE startToMsg.type = eaTo & pd.fromPurse : purses & pd.toPurse : purses &.
   pd.fromPurse =/ pd.toPurse & pd.value > 0 THEN.
.
   pursePaymentDetails(pd.toPurse) := pd --stores the payment details ||.
   msg := EncryptionAndDecryptionModule.encrypt(Message.createMessage(req, pd)).
END;.
.
msg <-- sendMoneyValue(fromPurse, decryptedRequestMsg)=.
   PRE decryptedRequestMsg.type = req & fromPurse : purses & .
   decryptedRequestMsg.content = pursePaymentDetails(fromPurse) THEN.
   .
   purseBalance(fromPurse) : = purseBalance(fromPurse) - pursePaymentDetails(fromPurse).value ||.
   msg := EncryptionAndDecryptionModule.encrypt(Message.createMessage(val, pursePaymentDetails(fromPurse)))
END;.
.
msg <-- recieveMoneyValue(toPurse, decryptedValueMsg) =.
   PRE decryptedValueMsg.type = val & toPurse : purses & .
   decryptedValueMsg.content = pursePaymentDetails(toPurse) THEN.
   .
   purseBalance(toPurse) : = purseBalance(toPurse) + pursePaymentDetails(toPurse).value ||.
   msg := EncryptionAndDecryptionModule.encrypt(Message.createMessage(ack, pursePaymentDetsails(toPurse))).
END;.
.
msg <-- ackPurse(fromPurse, decryptedAckMsg) =.
   PRE decryptedAckMsg.type = ack & fromPurse : purses & .
   decryptedAckMsg.content = pursePaymentDetails(fromPurse) THEN.
   .
   msg := Message.createMessage(unprotected, pursePaymentDetails(fromPurse)).
END;.
```

The PaymentDetails machine included in the ElectronicPurse machine represents the PaymentDetails KAOS entity with its attributes and operations. The entity attributes, which are the payer purse, the payee purse and the amount of money requested for transfer, are modeled as the machine variables while the entity invariant is modeled as part of the machine invariant. The machine invariant specifies the types of the variables and states the constraint that the payer purse should not be the same as the payee purse meaning that a purse cannot transfer money to itself. The only operation associated with the PaymentDetails entity is to create a new PaymentDetails with some parameters.

```
MACHINE PaymentDetails.
SEES ElectronicPurse.
VARIABLES.
fromPurse,toPurse, value.
INVARIANT.
fromPurse : ElectronicPurse.purses & .
toPurse : ElectronicPurse.purses &.
value : NATURAL1 &.
fromPurse /= toPurse.
OPERATIONS.
paymentDetails <-- createPaymentDetails(sender, receiver, amount).
   PRE sender : ElectronicPurse.purses & receiver : ElectronicPurse.purses &.
        amount : NATURAL1 & sender /= receiver THEN.
   ANY newPaymentDetails THEN.
        paymentDetails.fromPurse := sender ||.
        paymentDetails.toPurse := receiver ||.
        paymentDetails.value = amount .
   END.
END;.
```

The Message machine included in the ElectronicPurse machine represents the Message KAOS entity and defines the different types of messages. Each message has type and content specified as variables for the machine. An operation to instantiate multiple messages has been added to the machine.

```
MACHINE Message.
SEES ElectronicPurse, StrTokenType.
DEFINITIONS ; MESSAGE == 0..(ElectronicPurse.maxPurses*5)-1.
SETS MESSAGE_TYPE = {eaFrom, eaTo, req, val, ack, unprotected, readExceptionLog, clearExceptionLog, exceptionLogResult, abort}
VARIABLES type, content.
INVARIANT type : MESSAGE_TYPE & content : STRTOKEN.
OPERATIONS.
message <-- createMessage(msgType, msgContent) =.
  PRE type : MESSAGE_TYPE & content : STRTOKEN THEN.
        message.type := msgType ||.
        message.content := msgContent.
END;.
```

The Message machine specifies an enumeration for the types of messages communicated in the system in the MESSAGE_TYPE set.

### 4.2.6 Derivation of Design and Implementation

The initial B machines are refined using the B refinement mechanism to enforce design decisions. In this section, only the refinement of the system machine ElectronicPurse is illustrated to keep the discussion focused and concise. The first refinement step is classified as data refinement concerned with refining the representation of the machine variables that reflect the system state. The first refinement step represents the set of electronic purses as an array of purses while the security operations are refined accordingly. From the traceability perspective, the data refinement step does not directly address the realization of a specific security requirement in the system. It rather concentrates on building a concrete data structure representing the internal system state in a form realizable by programming languages while implementation is generated. This implies that the main parts that are modified in the data refinements step are the variables section that is concerned with the representation of the system state. The machine invariant changes to reflect the types and constraints of the new machine state representation.

The invariant of the refining machine should be linked to the variables of the refined machine by means of linking invariant that describes the relationship between the state spaces of the two machines [43]. The linking invariant in the first refinement step of the electronic smart card system is dom(pursesr) = purses meaning that the set purses is precisely the domain of the function pursesr since this gives the index of the set of elements that appear in the array. The operations are defined on the variable pursesr. Operations are required to work only within their preconditions given in the abstract machine, so those preconditions are assumed to hold for the refined operations. Therefore, the type information of the input variables and the other constraints do not need to be repeated in the refinement machine. The following piece of B code illustrates the refined data representation in the variables section, and the invariant section. The send and receive money operations are illustrated below as examples of how operations are modified according to the changes in the data representation, with the parts that reflect the changes highlighted.

**REFINEMENT** ElectronicPurseR

**REFINES** ElectronicPurse

**SEES** StrTokenType, EncryptionAndDecryptionModule

**INCLUDES** PaymentDetailsR

**SETS** STATUS = {eaFrom, eaTo, Request, Value, Ack}

**VARIABLES**
pursesr, purseNextSeqNo, purseBalancer, purseLostr, purseStatusr, pursePaymentDetailsr

**INVARIANT**
pursesr : 1..maxPurses >-> purseName & dom(pursesr) = purses &
purseNextSeqNo : pursesr --> NATURAL1 &
purseBalancer : pursesr --> NATURAL1 & ran(purseBalancer) = purseBalance &
purseLostr : pursesr --> NATURAL1 & ran(purseLostr) = purseLost &
purseStatusr : pursesr --> STATUS & ran(purseStatusr) = purseStatus &
pursePaymentDetailsr : pursesr --> PaymentDetailsR & ran(pursePaymentDetailsr) = pursePaymentDetails &

```
purseStatusr(name) = Request => ((name = pursePaymentDetailsr(name).fromPurser) &
(pursePaymentDetailsr(name).valuer <= purseBalancer(name)) &
(pursePaymentDetailsr(name).fromSeqNo < purseNextSeqNo(name)))&
purseStatusr(name) = Value => (pursePaymentDetailsr(name).toSeqNo < purseNextSeqNo(name)) &
purseStatusr(name) = Ack => (pursePaymentDetailsr(name).fromSeqNo < purseNextSeqNo(name))
```

**OPERATIONS**

```
msg <-- sendMoneyValue(fromPurse, decryptedRequestMsg) =
  PRE decryptedRequestMsg.type = req & fromPurse : pursesr &
  decryptedRequestMsg.content = pursePaymentDetailsr(fromPurse) &
  purseStatusr(fromPurse) = Request THEN

  purseBalancer(fromPurse) : = purseBalancer(fromPurse) –
                                   pursePaymentDetailsr(fromPurse).valuer;
  purseStatusr(fromPurse) := Ack;
  msg := EncryptionAndDecryptionModule.encrypt(Message.createMessage(val,
                   pursePaymentDetailsr(fromPurse)));
END;

msg <-- recieveMoneyValue(toPurse, decryptedValueMsg) =
  PRE decryptedValueMsg.type = val & toPurse : pursesr &
  decryptedValueMsg.content = pursePaymentDetailsr(toPurse) &
  purseStatusr(toPurse) = Value THEN

  purseBalancer(toPurse) : = purseBalancer(toPurse) + pursePaymentDetails(toPurse).valuer;
  purseStatusr(toPurse) := eaTo;
  msg := EncryptionAndDecryptionModule.encrypt(Message.createMessage(ack,
                   pursePaymentDetailsr(toPurse)));
END;
```

The first refinement step does not change the abstract specifications of the machine operations since it is only concerned with a data refinement step. The abstract specifications of the machine operations only changes in order to accommodate the new data representation of the machine status through using pursesr instead of purses and the rest of the arrays for the purse's attributes.

The second refinement step is concerned with refining the purse exception log and message archiving. The purse exception log records hold sufficient information about failed or problematic transactions to reconstruct the value lost in the transfer. The second refinement step makes a design decision about modeling the purse exception log as a partial function whose domain is the set of purses and whose range is the exception logs. It also introduces the concept of transaction sequence number that gives a sequence number for each transaction to distinguish the different transactions running in the system at a given moment. The combination of purse name and sequence number uniquely identifies the transaction [6]. The second refinement step defines a set CLEAR that is the set of clear codes for purse exception logs. A clear code is provided by an external source in order to clear a purse's exception log. *image* is a function to calculate the clear code for a given non-empty set of exception records.

*image* : P1 *PayDetails* >->*CLEAR*

image takes a set of exception logs and produces another value used to validate a log clear command. For each set of PaymentDetails, there is a unique code. The following piece of code shows the segments of the second refinement step that reflect the design decisions concerned with the purse's exception log.

**REFINEMENT** ElectronicPurseRR

**REFINES** ElectronicPurseR

**SEES** StrTokenType, EncryptionAndDecryptionModule

**INCLUDES** MessageR

**SETS** CLEAR --the set of clear codes for purse exception logs

**CONSTANTS** IMAGE

**PROPERTIES** !(name: pursesr).(IMAGE : POW1(pursePaymentDetailsr(name) >-> CLEAR))

**VARIABLES** purseExceptionLogr

```
INVARIANT purseExceptionLogr : pursesr +-> ElectronicPurse.EXCEPTION_LOG &
        dom(purseExceptionLogr) = ElectronicPurse.MAX_LOG_SIZE &
        ran(purseExceptionLogr) = PaymentDetailsR &
        !(pd, i : (purseExceptionLogr(name))(i)).((name : pd.fromPurser) or (name : pd.toPurser))
```

The increasePurseSeqNo operation has been introduced in the second refinement to reflect the design decision of assigning a unique sequence number to distinguish each transaction.

**OPERATIONS**

```
increasePurseSeqNo(purseName) =
BEGIN
  purseNextSeqNo(purseName) := purseNextSeqNo(purseName) + 1;
END;
```

The operations send and receive money are illustrated as examples of operations reflecting the design decisions made in the second refinement step. Each of the two operations adds the messages communicated in the operation in the central archive for history recording (see the highlighted lines).

```
msg <-- sendMoneyValue(fromPurse, decryptedRequestMsg) =
   PRE decryptedRequestMsg.type = req & fromPurse : pursesr &
   decryptedRequestMsg.content = pursePaymentDetailsr(fromPurse) &
   purseStatusr(fromPurse) = Request THEN

   purseBalancer(fromPurse) : = purseBalancer(fromPurse) - pursePaymentDetailsr(fromPurse).valuer;
   purseStatusr(fromPurse) := Ack;
   msg := MessageR.createMessage(val, pursePaymentDetailsr(fromPurse));
   MessageR.addMessageToArchive(msg);
   msg := EncryptionAndDecryptionModule.encrypt(msg);
END;

msg <-- recieveMoneyValue(toPurse, decryptedValueMsg) =
   PRE decryptedValueMsg.type = val & toPurse : pursesr &
   decryptedValueMsg.content = pursePaymentDetailsr(toPurse) &
   purseStatusr(toPurse) = Value THEN

   purseBalancer(toPurse) : = purseBalancer(toPurse) + pursePaymentDetails(toPurse).valuer;
   purseStatusr(toPurse) := eaTo;
   msg := MessageR.createMessage(ack, pursePaymentDetailsr(toPurse));
   MessageR.addMessageToArchive(msg);
   msg := EncryptionAndDecryptionModule.encrypt(msg);
END;
```

The readPursexceptionLog and clearPurseExceptionLog operations are the most affected among the other operations with the design decision of introducing the purse's exception log. The readPursexceptionLog operation retrieves the content of a given purse's exception log. The clearPurseExceptionLog operation flushes the content of a given purse's exception log. All message communicated in the two operations are stored in the central archive as highlighted below.

```
msg <-- readPursexceptionLog(purseName, readExceptionLogMsg) =
  PRE readExceptionLogMsg.type = readExceptionLog & readExceptionLogMsg.content =/ {} &
  purseName : pursesr THEN

  IF(purseStatusr(purseName) /= eaFrom) THEN
     abortCurrentTransaction(purseName);
  END
  IF purseExceptionLogr(purseName) =/ {} THEN
     msg := MessageR.createMessage(exceptionLogResult, purseExceptionLogr(purseName));
     MessageR.addMessageToArchive(msg);
     EncryptionAndDecryptionModule.encrypt(msg);
  ELSE
     msg := MessageR.createMessage(unprotected, EmptyStringToken);
     MessageR.addMessageToArchive(msg);
  END
END;

msg <-- clearPurseExceptionLog(purseName, decryptedClearExceptionLogMsg, imageExpLog) =
  PRE purseExceptionLogr(purseName) /= {} &
  decryptedClearExceptionLogMsg.type = clearExceptionLog &
  decryptedClearExceptionLogMsg.content /= {} &
  purseName : pursesr & imageExpLog : IMAGE THEN

  IF(purseStatusr(purseName) /= eaFrom) THEN
     abortCurrentTransaction(purseName);
  END
  purseExceptionLogr(purseName) := {};
  msg := MessageR.createMessage(unprotected, EmptyStringToken);
  MessageR.addMessageToArchive(msg);
END;
```

The introduction of the exception log and the message archive in the second refinement step has been reflected in all the operations by archiving the messages generated in the operation. The major operations that got affected by the exception log and the message archive are the readPurseExceptionLog and clearPurseExceptionLog that refined their abstract specifications to employ the concrete exception log and message archive.

The third refinement step makes a design decision about using public key infrastructure for protecting the integrity and confidentiality of messages. The communication device receives a pair of public/private keys for each purse. These keys are used to sign/verify and encrypt/decrypt the transaction messages. The third refinement step is not illustrated in more details because its B code is quite lengthy.

The last step is to construct an implementation machine before code generation. The implementation machine is a special refinement step done only once in development after all the design decisions have been made during design meaning that it does not have any non-determinism. The implementation machine does not have a state but manipulates the state of the machine it refines and the imported machines. The following piece of code illustrates the different sections of the implementation machine except for the operations section that is not illustrated as a whole; rather the send and receive operations as examples.

**IMPLEMENTATION** ElectronicPurseI

**REFINES** ElectronicPurseRRR

**USES** StrTokenType, Bool_TYPE

**IMPORTS**
EncryptionAndDecryptionModule, PublicKey, PrivateKey, PaymentDetails, Message

**INVARIANT**
pursesArray = pursesr & purseNextSeqNoArray = purseNextSeqNo & purseBalancerArray =
purseBalancer &
purseLostrArray = purseLostr & purseStatusrArray = purseStatusr &
pursePaymentDetailsrArray = pursePaymentDetailsr & purseExceptionLogrArray = purseExceptionLogr &
pursePublicKeyArray =  pursePublicKey & pursePrivateKey = pursePrivate &
purseEaToTimeArray =  purseEaToTime & purseRequestTimeArray = purseRequestTime

Both the send and the receive operations locate the concerned purse using a while loop that could be directly translated to the C programming language during code generation.

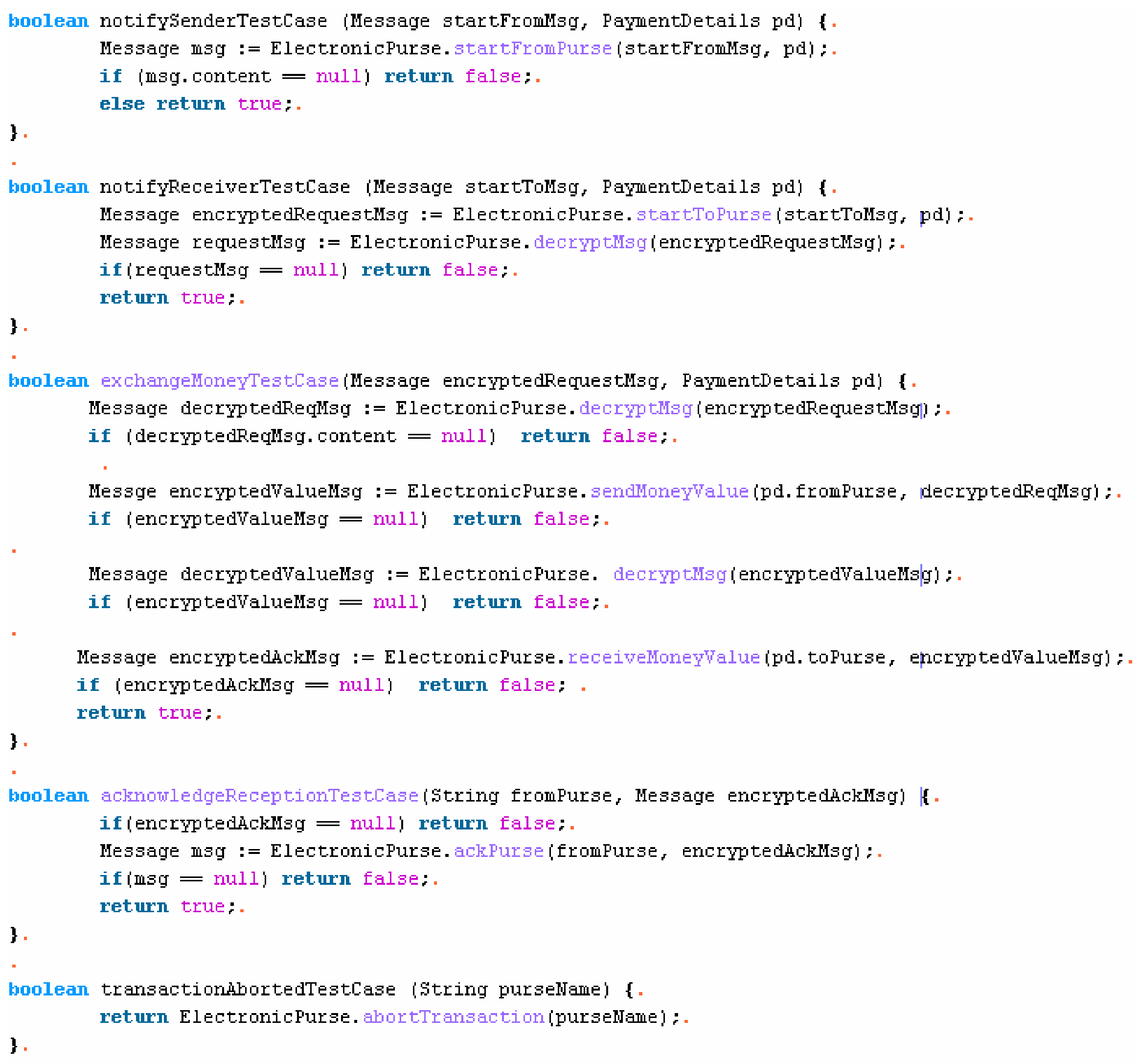
**OPERATIONS**
msg <-- sendMoneyValue(fromPurse, decryptedRequestMsg) =
  **VAR** ii **IN**

```
  ii := 0;
  WHILE ii <= maxPurses
  DO ii := ii + 1;
  IF (pursesArray(ii) == fromPurse) THEN
         purseBalancerArray(ii) := purseBalancerArray(ii) - pursePaymentDetailsrArray(ii).value;
         purseRequestTimeArray(ii) := CURRENT_TIME;
         purseStatusrArray(ii) := "Ack";
         msg := Message.createMessage("val", pursePaymentDetailsrArray(ii));
         Message.addMessageToArchive(msg);
         msg := EncryptionAndDecryptionModule.encrypt(msg,
                 pursePublicKeyArray(pursePaymentDetailsrArray(ii).toPurse));
  END
  INVARIANT
   ii : NATURAL1 & ii <= maxPurses & pursesArray = pursesr &
   pursePublicKeyArray = pursePublicKey
  VARIANT
   maxPurses - ii
  END
END;

msg <-- recieveMoneyValue(toPurse, decryptedValueMsg) =
  VAR ii IN
  ii := 0;
  WHILE ii <= maxPurses
  DO ii := ii + 1;
  IF (pursesArray(ii) == toPurse) THEN
         IF ((CURRENT_TIME - purseEaToTimeArray(ii)) < 10) THEN
           purseBalancerArray(ii) : = purseBalancerArray(ii) + pursePaymentDetailsArray(ii).value;
           purseStatusrArray(ii) := "eaTo";
           msg := Message.createMessage("ack", pursePaymentDetailsrArray(ii));
           Message.addMessageToArchive(msg);
           msg := EncryptionAndDecryptionModule.encrypt(msg,
                 pursePublicKeyArray(pursePaymentDetailsArray(ii).fromPurse));
         ELSE
            abortCurrentTransaction(toPurse);
         END
  END
  INVARIANT
    ii : NATURAL1 & ii <= maxPurses & pursesArray = pursesr &
    pursePublicKeyArray =  pursePublicKey
  VARIANT
    maxPurses - ii
  END
END;
```

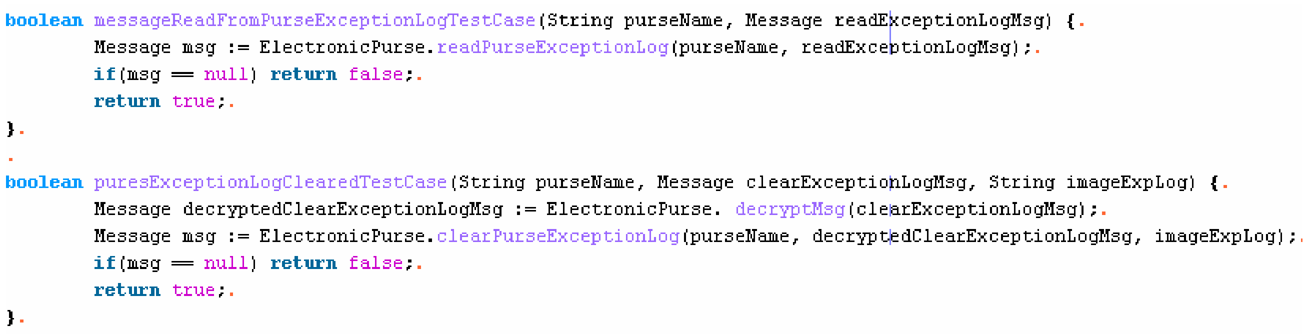
The implementation machine imports EncryptionAndDecryptionModule, PrivateKey, PublicKey, PaymentDetails and Message machines since these machines provide the functionalities needed by the operations of the implementation machine. All the operations use the loop construct in B in order to locate the purse specified in the operation parameters and manipulate its attributes as required.

### 4.2.7 Acceptance Testing

The Goal Graph Analyzer tool generates a suite of acceptance test cases derived directly from the KAOS goal graph. The tool parses the graph using a DFS algorithm with backtracking facility in order to generate scenarios with a sequence of operation calls from the goal graph. The derived B implementation specifications are then verified against the acceptance test cases to check for compliance between the requirements model and the derived implementation and to ensure the preservation of the security properties specified in the requirement model. The generated test cases might be augmented with some messages and assertions for better usability of the test results. The Goal Graph Analyzer has generated seven test cases for the electronic smart card security requirements as indicated below. The notifySenderTestCase calls the startFromPurse operation in order to test the scenario of notifying the payer purse of the start of a new transaction. The notifyReceiverTestCase calls the startToPurse operation testing the notification scenario for the payee purse to start a new transaction. The exchangeMoneyTestCase is the major among the generated test cases since it is the main scenario that realizes the actual exchange of money between the payer and the payee purses. The exchangeMoneyTestCase issues calls to both the sendMoneyValue and receiveMoneyValue operations. The other four test cases test acknowledgment reception, transaction abortion, reading of purse's exception log and clearance of purse's exception log operations. The test cases cover the scenarios that issue calls to all the operations of the system to increase the probability of error-detection and therefore increase the confidence in the resulting software security.



```
boolean notifySenderTestCase (Message startFromMsg, PaymentDetails pd) {
        Message msg := ElectronicPurse.startFromPurse(startFromMsg, pd);
        if (msg.content == null) return false;
        else return true;
}

boolean notifyReceiverTestCase (Message startToMsg, PaymentDetails pd) {
        Message encryptedRequestMsg := ElectronicPurse.startToPurse(startToMsg, pd);
        Message requestMsg := ElectronicPurse.decryptMsg(encryptedRequestMsg);
        if(requestMsg == null) return false;
        return true;
}

boolean exchangeMoneyTestCase(Message encryptedRequestMsg, PaymentDetails pd) {
       Message decryptedReqMsg := ElectronicPurse.decryptMsg(encryptedRequestMsg);
       if (decryptedReqMsg.content == null)  return false;

       Messge encryptedValueMsg := ElectronicPurse.sendMoneyValue(pd.fromPurse, decryptedReqMsg);
       if (encryptedValueMsg == null)  return false;

       Message decryptedValueMsg := ElectronicPurse. decryptMsg(encryptedValueMsg);
       if (encryptedValueMsg == null)  return false;

      Message encryptedAckMsg := ElectronicPurse.receiveMoneyValue(pd.toPurse, encryptedValueMsg);
      if (encryptedAckMsg == null)  return false; 
      return true;
}

boolean acknowledgeReceptionTestCase(String fromPurse, Message encryptedAckMsg) {
        if(encryptedAckMsg == null) return false;
        Message msg := ElectronicPurse.ackPurse(fromPurse, encryptedAckMsg);
        if(msg == null) return false;
        return true;
}

boolean transactionAbortedTestCase (String purseName) {
        return ElectronicPurse.abortTransaction(purseName);
}
```



```
boolean messageReadFromPurseExceptionLogTestCase(String purseName, Message readExceptionLogMsg) {
        Message msg := ElectronicPurse.readPurseExceptionLog(purseName, readExceptionLogMsg);
        if(msg == null) return false;
        return true;
}

boolean puresExceptionLogClearedTestCase(String purseName, Message clearExceptionLogMsg, String imageExpLog) {
        Message decryptedClearExceptionLogMsg := ElectronicPurse. decryptMsg(clearExceptionLogMsg);
        Message msg := ElectronicPurse.clearPurseExceptionLog(purseName, decryptedClearExceptionLogMsg, imageExpLog);
        if(msg == null) return false;
        return true;
}
```

Each test case returns a boolean that indicates whether the test case has succeeded or failed given a specific input vector. The master test case in the above test suite is the exchangeMoneyTestCase that does the actual exchange of money between the payer and the payee. This means that this test case has to be run carefully to test he scenario with different input vectors and it also needs to be augmented by meaningful messages to give more clear results.

## 4.3 FADES versus Z

This section qualitatively compares the application of FADES to the application of Z to specify, design and implement the security requirements of the electronic smart card system. Stepney has modeled and implemented the security requirements of the electronic smart card system using the Z formal methods [6]. Stepney has developed two key models in their specification to the security requirement. The first is an abstract model describing the world of purses and the exchange of value through atomic transactions expressing the security properties that the cards must preserve. The second is a concrete model reflecting the design of the purses which exchange value using a message protocol. Stepney then proved that the concrete model is a refinement of the abstract to ensure correctness. The notions of model refinement and development correctness are different in Z from their counterparts in B. In Z, both the model and its refinement model are developed and then we need to prove correctness of development by proving that the second model is a refinement of the first. In B, a model is refined into another model and proof obligations are generated in order to outline the proofs that need to be discharged to prove correctness of refinement. The refinement mechanism in B is more direct and the tool support for proof obligation generation facilitates the process of ensuring correctness.

The following three metrics have been defined for comparison between FADES and Z when both are used to specify and implement the electronic smart card system:

- *Requirements completeness*: this metric qualifies how much of the set of security requirements specified by the stakeholders has been covered by the system specification and implementation. This metric is measured by the number of security requirements covered in each model (FADES and Z) with respect to the total number of requirements specified by the stakeholders.
- *Requirements consistency*: this metric assesses the consistency among the requirements specified in the requirements model and the consistency between the different artifacts of the development process such as the consistency between requirements and design and the consistency between design and implementation. Van Lamsweerde has used semantic conflicts as a measure of model consistency [3]. The requirements consistency metric is therefore measured by the number of conflicts detected and resolved in the model during requirements specifications since conflicts reflect the contradictions in the requirements model. The more conflicts detected early in development during requirements specifications, the more consistent the requirements model is and the more consistent the design and implementation phases are.
- *Security quality*: this metric refers to the security quality of the resulting model, either the requirements model or the implementation. This metric is measured by the number of security vulnerabilities detected and resolved in the model. Although not all vulnerabilities are weighted

equal meaning that some categories of security vulnerability are more severe than others, there is no severity assessment to the different categories of security vulnerability in the literature and this is, in fact, an open research area that might be part of the future work of this dissertation.

The main difference between FADES and the Z formal methods when applied to the electronic smart card system comes in the requirements elaboration and analysis phase since the design and implementation phases in FADES are elaborated using the B formal method, which belongs to the same class of formal methods as Z. The following discussion focuses on the comparison between FADES and Z during requirements elaboration while assessing the requirements models using the metrics described above.

Both FADES and Z were able to consider and cover all the security requirements specified for the system meaning that as far as the requirements completeness metric is concerned, both resulting requirements models and implementation specifications were complete. However, the goal-oriented nature of the KAOS requirements modeling approach employed in FADES allows for focusing on each specific requirement, its refinement and its relation to other requirements. This tends to raise the probability of building a near-complete requirements model. The completeness of the requirements model assists in constructing more robust design and implementation in which all requirements are considered and handled properly with respect to the rest of the system. Stepney was quite capable of constructing requirements specifications in Z that are as complete as the KAOS requirements model built in FADES. However, we might expect that with larger and more complicated systems, Z as a formal method might not be as capable as FADES to construct a close to complete requirements model. We justify our expectation by the results obtained in the experiment (described in chapter 5) to compare FADES to other formal methods. The experiment results have shown that elaborating and analyzing security requirements using the traditional formal methods is not as effective as a semi-formal method with an underlying formal infrastructure like KAOS. The rationale for this result is that requirements are often vague and very hard to specify in strict format like formal methods that have well-specified structure making formal methods like Z more suitable for later development stages like design and implementation. The well-specified structure of formal methods (the B method employed in FADES and the Z method) has resulted in constructing complete design and implementation specifications with respect to the requirements models in both FADES and Z.

Comparing the requirements models resulted from FADES and Z using the consistency metric reflects more strength of FADES over Z in constructing a more consistent requirement model. The KAOS requirements framework employed in FADES dedicates a stage for analyzing possible conflicts to requirements during requirements elaboration. Allowing requirements engineers to dedicate time and effort to focus on detecting semantic conflicts among requirements increases the probability of constructing a more consistent and less conflicting requirements and therefore more consistent design and implementation. In Z, Stepney was able to build a requirements specification model that is as consistent as the one constructed with FADES since Z is a formal language that is based on a robust mathematical model in which inconsistencies could be detected. Although both models built with FADES and Z are equally consistent, it is more costly in terms of time and effort to build a consistent requirements model using formal languages have more complicated syntax.

FADES inherits the merit of KAOS in performing threat analysis to the requirements model very early in development in order to analyze possible security vulnerabilities that might threat the system. The obstacle analysis performed on the goal graph of the electronic smart card system detected some security vulnerabilities that were not considered by Stepney in her development of the system using the Z formal method. The obstacle analysis performed in FADES has revealed two security breaches that threat the confidentiality and integrity of the messages being communicated among the electronic purses to transfer money. Stepney, on the other hand, was able to detect threats to the message confidentiality and introduced an encryption/decryption facility to protect the message while in transit. FADES has enforced the protection of messages integrity through the employment of digital signatures meaning that each message has to be signed by the sending purse and verified on the receiving side so that no tampering with the message changing its content becomes possible. We expect that if the case study has been larger in size and its security requirements have been more complicated, FADES would have shown more strength in being able to detect more security vulnerabilities than the Z formal language which is not equipped with special mechanism for analyzing security vulnerabilities during requirements elaboration as in FADES.

The ability of FADES to verify the compliance of the resulting implementation against the requirements model using the derived acceptance test cases enhances the position of FADES compared to the Z development of the electronic smart card system. The derived acceptance test cases are able to detect some types of completeness and consistency problems. For example, the derived test case for money exchange would have been able to detect the missing requirements of deducting money from a purse that does not have sufficient funds had the requirements engineer missed this requirement. Another example of detecting an inconsistency in implementation through the acceptance test cases is when the software developer mistakenly forgets to change the purse state at the end of an operation that mandates a state change. These examples demonstrate the ability of the acceptance testing to provide an extra verification step that does not exist in the Z method to ensure compliance and correctness of implementation with respect to requirements.

While this section has qualitatively compared FADES to formal methods, the next chapter provides a quantitative comparison and draws some conclusions based on the analysis of the comparison results.

# Chapter 5: FADES Validation

This chapter describes the approach used to substantiate the validity of FADES. It describes the validation process and the rationale for choosing it over other alternatives. Further, this chapter presents and analyzes results collected from questionnaires, interviews, surveys and informal comments from security engineering practitioners and experts in the formal methods and security domains.

New software engineering approaches are commonly validated using a longitudinal study in which one ore more industrial enterprises employ the new approach, and the research team gather data that enables the evaluation of the approach [107]. Therefore, the longitudinal study is the ideal validation mechanism for FADES because observing and congregating data about the approach and comparing it to the current security engineering practice in the targeted enterprises produces the empirical evidence needed to validate the approach.

However, there are two problems associated with the longitudinal kind of study. First, it takes extensive periods of time. Second, it requires the targeted enterprises to buy into the use of FADES before having sufficient empirical evidence. The latter even complicates the problem because many software enterprises act hesitantly to adopt a new approach with no strong evidence of its validity. Therefore, to generate highly credible results without a longitudinal study, we designed a controlled experiment in which software engineering practitioners applied FADES versus other formal methods to elicit and analyze security requirements. Among the obstacles encountered in the practitioners experiment was to scarcity of software engineers with formal methods expertise. This led to a smaller than desired pool of participants in the study. Therefore, to strengthen the obtained results, we resorted to conducting another experiment employing academic and industry experts to evaluate the practitioners results and anticipate their scalability.

The results should highlight the positive impact and effectiveness as well as the potential limitations of FADES in handling security concerns. Moreover, validating the approach with feedback from the software engineering community whose members are the target users of the approach enables a future longitudinal study. Obtaining positive feedback from software engineers has the effect of increasing the credibility of FADES at the software enterprises side so that their willingness to participate in future longitudinal studies would increase.

## 5.1 The Validation Approach

The validation approach has the objective of evaluating FADES by the security engineering community including its two main groups; practitioners, academic and industry experts. There were some challenges met during the evaluation process. The initial idea was to let the practitioners' community only examine FADES through allowing a large sample of software engineers apply FADES and gather their feedback in a statistical study. However, the availability of the required formal methods skills that could apply FADES was limited by the complexity of learning and using formal methods. Further, practitioners have limited time in their schedules to invest in such studies. Therefore, the more feasible solution has been to select a small group of practitioners (10 software engineers) and train them in FADES in order to be able to apply it to a small case study. The process of collecting the results focused more on digging into the personal experiences of the participants rather than the statistical significance of the collected data. Then, the idea of strengthening the results of the practitioners' study by the evaluation of the approach by academic and industry experts was found reasonable. Based on their research and industrial experience in formal methods and security, academic and industry experts were able to provide objective and through evaluation of FADES and highlight some of its potential strengths and limitations. Further, the academic and industry experts were able to provide scientific expectations about the scalability of the results obtained in the practitioners' study to strengthen these results that are based on a small sample of software engineers.

The results obtained from the two groups have been collected in two empirical studies. The first study is a quantitative comparison between FADES and a relevant formal method-only approach for specifying and implementing security requirements. This study involves experienced software engineering practitioners that represent a sample of the expected applicants of FADES. The results have been collected both quantitatively through multiple-choice survey questions and qualitatively through interviews with the participants after the experiment. The second study focuses on the academic and industry experts in the area of security engineering and/or formal methods. While the scope of the research presented here does not provide a more comprehensive comparison across a range of large and small projects, it does provide a persuasive argument for FADES over a strictly formal approach supported by the evaluation of experts in the two research domains of the approach.

## 5.2 First Experiment: Practitioners

This study is a controlled experiment that selects subjects from the software engineering community with industrial experience to apply FADES and the B formal method to specify, design and implement portion of the electronic smart card case study. The chief objective of the quantitative comparison between FADES and B is to explain and measure the ability of FADES as opposed to other formal methods to build a robust, near-complete and near-consistent model of requirements and preserve the security properties throughout the rest of the development.

The coming sections illustrate the hypothesis, the experiment design, experiment variables, subject pool qualifications and subjects' selection criteria. The results of the experiments and their analysis and relation to the experiment hypothesis are also discussed. The results analysis section relates the results obtained from the experiment to the research questions of this dissertation and depicts how these questions have been validates using the empirical study.

### 5.2.1 Experiment Design and Hypothesis

The experiment focuses on the specification phase of development for the security requirements of the money exchange segment of the electronic smart card case study. That is, the subjects participating in the experiment are required to elaborate the security requirements of the money exchange and construct the initial B model that represent the requirements model. Half of the subjects construct the initial B model using FADES by constructing a KAOS requirements model and apply the transformation rules to transform the KAOS model to B using the Goal Graph Analyzer tool. The second half of the subjects constructs the initial B model by specifying and elaborating the security requirements directly in the B formal language. The rationale for focusing only on the specification phase is that the design and implementation is FADES is done already in the B language meaning that after the specification and analysis of the requirements, both approaches (FADES and B) are almost the same except for the extra verification step in FADES that derives acceptance test cases. Moreover, comparing the quality of the requirements models produced by FADES and B guides our expectations to the quality of the resulting implementations since the characteristics of the requirements model propagate to the later phases of development especially when using formal methods for development that guarantee correctness by construction. Further, the subject pool from which we choose the experiment participants are not familiar with either KAOS or B, so it would have been difficult to train them in the new approaches and require that they apply these approaches throughout the whole development lifecycle. This reason has also motivated the choice of only the money exchange segment of the electronic smart card system rather than the whole system as the case study for the experiment especially that formal methods are not easy to learn and apply.

The money exchange segment of the electronic smart card system has the following security requirements:

- *No value may be created in the system*: the sum of all the purses' balances after any transactions does not increase.
- *All value is accounted for in the system*; that is, no value is lost: the sum of all purses' balances and lost components after any transaction does not change.
- *Authentic purses*: a transfer can occur only between authentic purses. Authentic purses are those registered in the communication device with valid card and pin numbers for each purse.
- *Sufficient funds*: a transfer can occur only if there are sufficient funds in the payer purse.

The group that applies the B language needs to specify the security requirements of the money exchange part of the electronic smart card system using the B language in order to construct an abstract B model representing the requirements.

The group that applies FADES to elaborate and analyze the requirements is required to do the following:

1. Build an initial goal graph through goal refinement.
2. Negate the goals and regress them with relevant domain properties to obtain conflicts that are source of inconsistency and obstacles that represent possible security vulnerabilities.

3. Refine the obtained obstacles and apply some resolution patterns on these obstacles.
4. Integrate the results of the obstacle resolution with the requirements goal graph.
5. Operationalize the leaf goals and assign their responsibilities to agents in the system.
6. Use the Goal Graph Analyzer tool to construct the initial B model.

We accurately record the development time that each subject spends in the experiment excluding any time spent on activities other than the experiment work. The following assumptions are made about the environment of the experiment:

- Subjects do not have awareness of the business impact of security vulnerabilities. This means that subjects cannot estimate the business losses encountered by the security vulnerabilities identified during requirements elaboration.
- Relevant vulnerabilities are these that cause harm to the system and not to the people (exclude business impact of vulnerability).
- All subjects will be using the same machine that has all the necessary software tools so that the machine performance won't be variable in the experiment.

All subjects have been given the same training to both approaches FADES and B.

The hypothesis of the practitioners' experiment is that incorporating the goal-oriented KAOS framework for analyzing security requirement before stepping into formal design using FADES results in the construction of more complete, consistent, and secure software in less time than developing the same system security using formal methods like B.

The experiment has an independent variable and a set of dependent variables. The approach being applied whether it is FADES or B is the independent variable. The dependent variables are defined as follows:

- *Ability to reason about requirements*: this variable measures how the approach allows the subject to make rationale about security requirements at a high level during requirement capturing and justify analysis to these requirements at a more detailed level during requirement analysis.
- *Ability to capture environment properties*: this variable measure how provided constructs are effective and important to use to capture properties of the environment in which the envisioned system is to be deployed.
- *Ability to reason about security vulnerabilities during requirements analysis*: this variable measures how the provided constructs for vulnerability analysis are effective and important to use during requirements analysis
- *Ability to separate security concerns*: this variable measures the ability of the approach to separate security concerns from the rest of the system requirements.
- *Usability of the approach*: this variable measures the level of convenience the subject finds in applying the approach.
- *Learnability of the approach*: this variable measures the level of difficulty of learning the approach encountered by the subject.

- *Development time*: this variable measures the time the subject has spent in building the requirements specifications model.
- *Requirements completeness*: based on our assessment of the specifications models constructed by participants during the experiment, the requirements completeness level of the requirements specifications model done by each subject with respect to the model built for the case study is measured.
- *Requirements consistency*: based on our assessment of the specifications models constructed by participants during the experiment, the requirements consistency level of the requirements specifications model done by each subject with respect to the model built for the case study is measured.
- *Number of relevant security vulnerabilities discovered*: this variable measures the number of relevant security vulnerabilities that have system impact and might cause harm to the system. This variable does not take into account the vulnerabilities that might have a business impact since the subjects in this experiment are assumed to have no awareness of the business impact of vulnerabilities.

### 5.2.2 Experiment Questionnaires and Participants

Ten subjects participated in the experiment, and they are all software engineers working in IBM. The ten subjects were divided into two groups of five subjects each. The first group was given training in the FADES in order to apply it to the money exchange problem while the second group was given training in the B language so that subjects in this group are able to use the B language to specify and elaborate the security requirements of the money exchange. The results of the background survey distributed to the pool of candidates were used to choose the final ten participants based on selection criteria. The target subjects for the experiment are software engineers with an engineering degree and industrial experience of seven to ten years. The subjects should at least have a fair knowledge of security concepts and first-order predicate logic meaning that they have theoretically studied these concepts in an academic or training course, but not necessarily applied them in industrial projects. Subjects with background in formal methods and specifically KAOS and B are preferred, but it is not highly probable that software engineers working in industry might have this knowledge.

One of the main objectives while selecting participants was to have a baseline for the knowledge and experience level of all the participants so that the experiment results are not biased by these factors.

The experiment participants received a two-hour training session in the approach they apply either FADES or B prior to working on the experiment. The experiment was estimated to take three to four hours and subjects were required to fill in a questionnaire at the end of the experiment reflecting their feedback to the applied approach. An interview was held with each participant after answering the questionnaire to discuss the participant's answers. Both the questionnaires given to subjects to select the participants before the experiment and to assess the approaches at the end of the experiment are detailed in Appendix B. The models constructed by the experiment participants were then assessed against the model that we have built for the electronic smart card system to collect the necessary measures.

Two questionnaires were distributed to the participants; one before the experiment and another one at its end. The questionnaire distributed prior to the experiment inquired about the software engineering, industrial, and academic background of the candidate participants. The questionnaire was distributed to 15 candidates, and the answers of the candidate participants were used to select the ten participants in the experiment. The background questionnaire included 13 multiple-choice questions that focused on three areas; academic and industrial experience, industrial projects and the different roles played by the subjects, and the subjects' knowledge to the experiment-related technologies. Two questions queried the educational background and the industrial years of experience. Five questions inquired about the number of industrial software projects the subject has been involved in and his/her role in these projects (developer, designer, business analyst). Six questions asked about the subjects' own assessment to their knowledge levels in the experiment-related methods and technologies. The assessment questions focused on the security, formal methods, first-order predicate logic, KAOS, B, and test case design background. The background questions are summarized in table 3 while the whole questionnaire is detailed in Appendix B. Each question has one of four possible choices defined as follows:

- *No knowledge*: the subject does not have any knowledge about the method or technology in question.
- *Fair knowledge*: the subject theoretically knows the concepts through courses or readings, but he/she hasn't practiced this knowledge in an industrial project.
- *Moderate knowledge*: the subject theoretically knows the concepts and has applied them in industrial projects.
- *Good knowledge*: the subject has been applying the concepts for a while in a number of projects to the extent that he/she becomes a matter expert.

| To Determine.. | Question (s) Used |
|---|---|
| Academic & Industrial Experience | - What is your Computer Science-related education and academic background?<br>- How many years have you been in industrial software engineering? |
| Industrial Projects Experience | - How many software projects have you been involved in during your career?<br>- How many projects have you acted in as a developer?<br>- How many projects have you acted in as a designer or architect?<br>- What was the average size of the projects you have been involved with?<br>- How many projects have you acted in as system analyst? |
| Knowledge of FADES | - What is your assessment of your security background? |

|  |  |
|---|---|
| Related Technologies | - What is your assessment of your formal methods background?<br>- What is your assessment of your first-order predicate logic background?<br>- What is your assessment of your knowledge of the KAOS (Knowledge Acquisition for automated Specifications) requirements analysis method?<br>- What is your assessment of your test case design background? |

**Table 3: Participants' Background Questions**

The second questionnaire was a multiple choice one distributed to the participants at the end of the experiment inquiring about their experience with the method applied in the experiment whether it is FADES or B. The questionnaire included 12 quantitative questions based on 5-point Likert summated scale, from 1 "strongly disagree" to 5 "strongly agree". The 12 questions addressed two areas namely the effectiveness of FADES in handling security concerns and the usability of the approach. Though the main focus of the validation experiment is to substantiate the usefulness and effectiveness of FADES, validating the approach's usability increases the confidence in its ability to produce better security since the quality of the resulting security is a function of the approach's usability. The first nine questions addressed the effectiveness of each approach while the last three questions addressed the usability concerns of each approach (learnability, practicality, tool support, convenience of applicability).

The effectiveness questions, addressed two research questions:

1. How feasible and effective is the coupling between semi-formal requirements specifications and formal design an implementation in producing more complete, consistent and secure software?
2. How does the integration of semi-formal and formal methods in FADES compare to purely applying formal methods from the completeness, consistency and security quality perspectives?"

The usability questions addressed the effect of the approach usability on the resulting models' quality from the completeness, consistency and security quality perspectives. The questions concerning the approach effectiveness and usability are summarized in Table 4 while the questionnaire is detailed in Appendix B.

| To Determine.. | Question (s) Used |
|---|---|
| Effectiveness | The approach provides useful constructs to sufficiently reason about requirements. |
|  | The provided constructs to reason about requirements are effective. |
|  | The approach provides useful constructs to capture environment properties. |
|  | The provided constructs to capture environment properties are effective. |
|  | The approach provides useful constructs to identify and analyze security vulnerabilities during analysis of requirements. |
|  | The provided constructs to identify and analyze security vulnerabilities during analysis |

| | |
|---|---|
| | of requirements are effective. |
| | The approach provides effective means to construct a near-complete requirements model. |
| | The approach allowed you to separate security concerns (e.g. access control, authentication) from the rest of system requirements. |
| | The approach helped you build a better quality requirements model compared to the other requirements modeling methods you have used before. |
| Practicality | The approach is practical and has good tool support. |
| | The approach is easy to learn. |
| | The approach is fairly convenient and easy to apply. |

**Table 4: Participants' Questions**

The questions concerned with the effectiveness of the approach focused on quantifying the participants' assessment to the effectiveness of the approach constructs in modeling security requirements and producing a comprehensive, well-analyzed and organized requirements model. The effectiveness questions also inquired about the participants' own assessment of the quality of the models they constructed during the experiment. The participants' assessment of their models was used to compare to the research team assessment of the same models, which increased the confidence in the results of the experiment. The second area addressed by the questions is the usability of the approach including the approach applicability, tool support, easiness of learning, and convenience of application.

The quantitative multiple choice questions were then complemented by qualitative discussions through interviews hold with each participant after collecting the questionnaire answers. The major objective of the interviews was to elaborate on the questionnaire answers and to elicit a deeper feedback from the participants about their own experience when applying any of the two approaches (FADES or B). Further, the qualitative results collected during the interviews substantiated FADES since it was elicited from a sample of the target users of the approach. This allowed for identifying and discussing the potential strengths and limitations of FADES as opposed to fully-fledged formal methods. The depth of the qualitative discussions overcome the limitation of the validation mechanism in this research work that only experimented FADES in a controlled setting and on a small group of software engineers.

### 5.2.3 Evaluation Criteria

The results of the practitioners' experiment are obtained in two stages; the first is after the experiment using the questionnaire and the interviews with the participants to give their own assessment and report their individual experience with the approaches. The second stage to collect the results is through our assessment to the models constructed by the participants. The reference model against which the participants' models are assessed is that we have constructed using FADES in section 4.2 to model the electronic smart card system. The reference model produced the following numbers:

1. Four security requirements have been specified for the security problem used in the experiment as illustrated above.
2. The total number of conflicts revealed in the reference model for the money exchange part of the system is six conflicts. A conflict is defined as a semantic contradiction between the definition of one requirement with the definition of another requirement.
3. The number of security vulnerabilities identified in the reference model for this part of the system is seven.

The following metrics are defined for the assessment of the participants' models:

- *Requirements completeness:* this metric evaluates the completeness level of the requirements model and is measured using the requirements coverage measure that identifies the number of security requirements included and those missed from the model compared to the total number of security requirement defined for the system.
- *Requirements consistency*: this metric evaluates the level of consistency among the requirements and is measured by the number of conflicts identified in the model compared to the total number of conflict defined in the reference model. The conflicts are an indicative measure of the requirements model consistency since they reflect the contradictions between the requirements' definitions that result in inconsistency of requirements.
- *Security quality*: this metric evaluates the extent to which security aspects are defined and analyzed properly during requirement as a robust base for design and implementation that ensure security. The security quality metric is measured using the number of security vulnerabilities identified during requirements elaboration and analysis since this measure indicates the possible security breaches that might appear when the system runs into production.
- *Development cost*: this metric evaluates the cost of development from a time and effort perspectives and is measured by the number of man hours spent by each participant to construct his/her model. The financial cost is beyond the scope of this metric; however, the time cost might be used to estimate the financial cost.

### 5.2.4 Experiment Results

This section depicts the results obtained from the empirical study to validate FADES. The results have been collected in three stages. First, each participant has filled in a multiple choice questionnaire at the end of the experiment. Second, each participant has been involved in some interviews to elaborate on his/her questionnaire answers and dig into his/her individual experience with the applied approach during the experiment. Further, the model constructed by each participant has been discussed during the interviews to justify the decisions and evaluate the effective features and limitations of the approach used to build their requirements models. Third, the B models constructed by the participants in both the FADES group and the B group have been assessed by the research team to evaluate the quality of models using the four metrics outlined in the previous section. Section 5.2.1 describes the results obtained directly from the participants through the questionnaires and the interviews while section 5.2.2 outlines the assessment of the models constructed by the participants.

#### 5.2.4.1 Quantitative Results (Multiple-Choice Questionnaire)

The multiple choice questionnaire given to the participants at the end of the experiment has 12 questions inquiring about the participants' assessment to the approach he/she has applied (FADES or B). Each question has five choices, which are strongly disagree, disagree, I don't know, agree, and strongly agree. An ordinal ranging from 1 to 5 has been assigned to each choice so that results could be put on a graph format. The questionnaire is outlined in details in Appendix B. The KIVIAT diagram has been chosen to represent the answers of the 10 participants to each question. The rationale for representing the questionnaire answers in the form of KIVIAT diagrams is that they are usually used to compare the performance of multiple entities on the same set of axes while providing a simplified presentation of multiple performance indicators, which are the participants assessment of the two approaches (FADES and B) [92]. Further, KIVIAT charts are helpful in recognizing patterns of answers in each group, if any. The following KIVIAT charts represent the answers of the 10 participants in each of the 12 questions.

Questions 1 and 2 in the experiment questionnaire inquire about the participants' assessment of the usefulness and effectiveness of the approach constructs to reason about requirements. As depicted in Figures 34a and 34b, all the participants applying FADES are in agreement (4) and strong agreement (5) that FADES supports effective and useful constructs to reason about requirements. The B group participants on the other side are in disagreement (2) to the ability of the B method to provide means of reasoning about requirements.

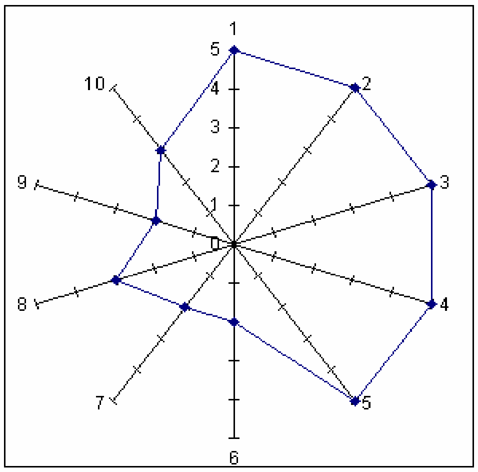


**Figure 34a: Question 1**

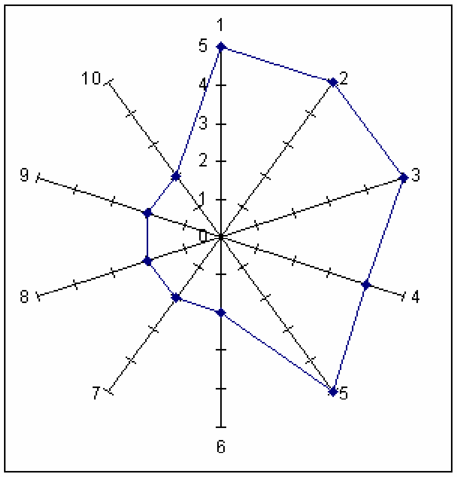


**Figure 34b: Question 2**

**Useful and effective constructs to reason about requirements**

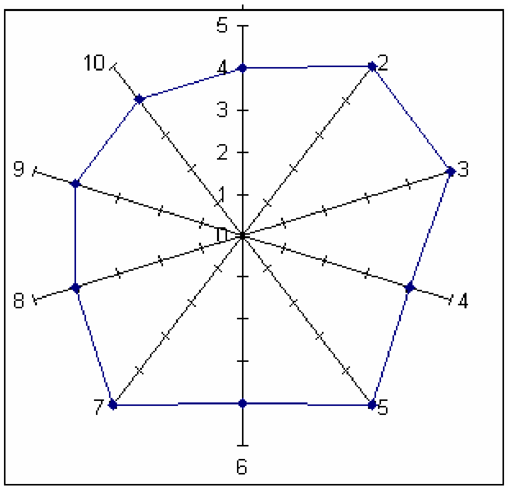
The answers for questions 1 and 2 show a pattern for the answers in each group in two extremes. The FADES group members' consensus on questions 1 and 2 reflects their positive attitude towards the effectiveness and usefulness of FADES features. They elaborated on which features in FADES were helpful in constructing effective requirement models during the interviews as indicated in the next section. Further, the constructed models during the experiment reflect effective reasoning about requirements that resulted in models close to the reference model with respect completeness, consistency and security quality.

The B group members have shown a consensus in the other extreme of the spectrum for questions 1 and 2. They showed a pattern of disagreement with the effectiveness and usefulness of the features provided by the B method to specify requirements. One of the main concerns in the interviews was to discuss why the B group members disagreed with questions 1 and 2 and the limitations of the B method that disabled them to effectively reason about requirements. While assessing the models resulted from the B group, the main focus has been in highlighting the portions of the model that might result from ineffective reasoning about requirements and see if these portions go along with the limitations of the approach discussed during the interviews. Also, the interviews attempted to discuss how much the limitations of the approach itself have contributed to the ineffective reasoning about requirements and how much resulted from the unfamiliarity of the members with the B formal method.

Questions 3 and 4 in the experiment questionnaire inquire about how each approach provides useful and effective constructs to capture domain properties that refer to constraints in the deployment environment of the software-to-be. The rationale for including questions about the ability of the approach to capture domain properties is that the software-to-be is actually composed of the software system itself and the environment in which it will be deployed. Therefore, the ability of the approach to capture environment properties is crucial to the construction of a comprehensive and thoroughly-analyzed requirements model.

As indicated in Figures 35a and 35b, participants in the two groups either agreed or strongly agreed that both FADES and B are effective in capturing environment properties.

This result has been expected since both FADES and B allow for formally modeling domain properties as predicates in the form of operations pre/post conditions and object invariants. If FADES had been compared to a semi-formal object-oriented analysis approach like UML, the result would have been different. One of the limitations of informal modeling of requirements is that it models the software-to-be in isolation from its environment [93].



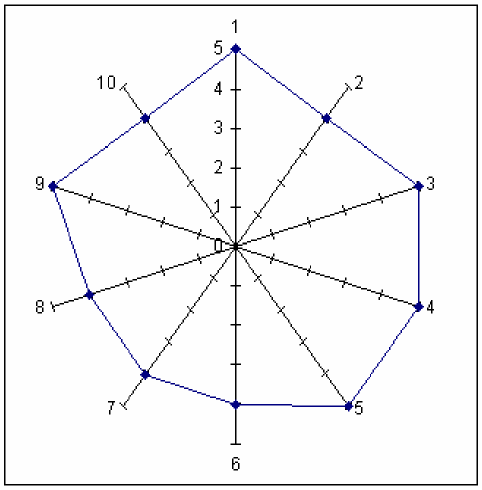
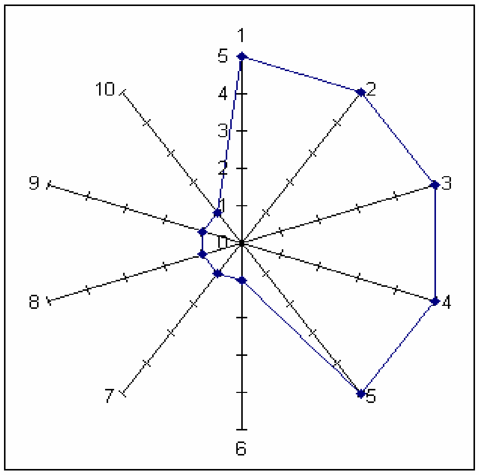


**Figure 35a: Question 3** **Figure 35b: Question 4**

**Useful and effective constructs to capture environment properties**

Questions 5 and 6 in the experiment questionnaire inquire about the usefulness and effectiveness of the constructs provided with the approach to detect and analyze security vulnerabilities. The two questions inquire about the effectiveness and impact of performing threat analysis to the requirements model when detection of security breaches is much less costly than later stages of development. As shown in Figures 36a and 36b, the FADES group is in consensus with strong agreement with the effectiveness of the threat analysis capability of the approach to provide focus and means to identify possible security vulnerabilities and analyze them for possible resolutions. The B group, on the other side, has expressed their strong disagreement with the availability of threat analysis constructs in the B method.



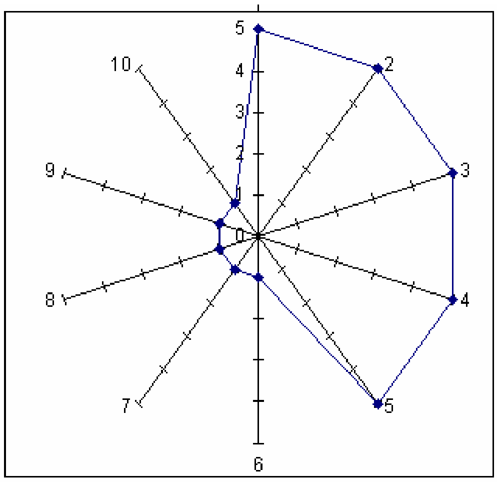


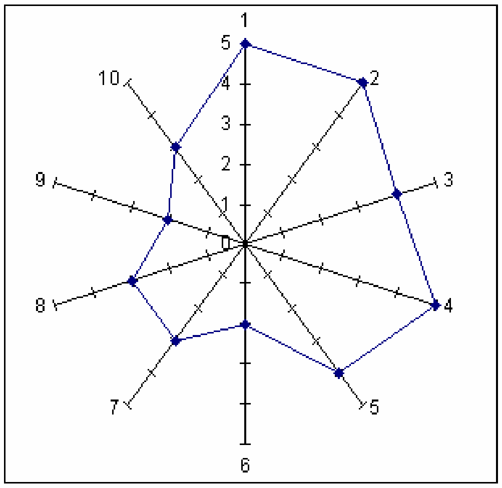
**Figure 36a: Question 5** **Figure 36b: Question 6**

**Useful and effective constructs to identify and analyze security vulnerabilities**

The two patterns shown in the two groups for their answers to questions 5 and 6 reflect the difference in impact between using a requirements analysis model that integrates threat analysis in its procedure like KAOS and using an approach that is not concerned with analyzing possible threats. The impact came clearer in the participants' models as illustrated in the coming sections where the FADES group members were able to systematically detect more security vulnerabilities. The B group members, on the other hand, have used their experience and background in security to detect possible security vulnerabilities.

Question 7 is concerned with asking the participants about their ability to build near-complete requirements model. This question assesses the opinion of the participants about the models they constructed using the two approaches. The results of this question are intended to compare to the results obtained from the research team assessment to the participant's requirements models. The FADES group members were satisfied with the completeness of their models as indicated by their agreement with the question in Figure 37. Two of the B group members, nevertheless, were dissatisfied with the completeness level of their models while the other three members in the group answered "I don't know". The three members in the B group who answered "I don't know" said that they chose this answer because they were unable to assess the completeness of the models they constructed as indicated in their interviews' discussions. This agreement of the FADES group member to question 7 reflects their clear satisfaction about the completeness level of their models, and this result is actually aligned with our assessment to the models of the FADES group members.

Question 8 assesses the ability of the approach to separate security concerns from the rest of the system requirements. The rationale for asking this question to the participants is to assess the ability of KAOS to model security goals in isolation from the rest of the functional and non-functional requirements of the system. The results of question 8 came supportive to the KAOS claim of separating security concerns as indicated in Figure 38. The FADES group came in strong agreement with the ability of security concerns separation while the B group members came with variant assessments to the issue. Two of the group members disagree with the ability of B to separate security requirements from the rest of the system requirements while two of the group members answered “I don’t know” and only one member agreed. This means that the FADES group has shown a pattern of consensus among its members about the issue of security concerns separation while the B group did not have a specific pattern, but rather came in variant directions for the answers. The Interviews following the multiple-choice questionnaire discussed the answers of this question with the B group members in more details in order to find a justification for the variance in the answers.



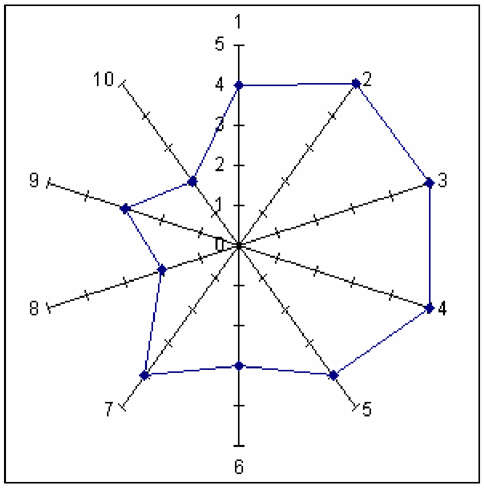


**Figure 37: Question 7, ability to build near-complete model**

**Figure 38: Question 8, separation of security concerns**

Question 9 inquires about the ability of the participants to build a better quality requirements model compared to their currently used methods. Better quality requirements model has been defined prior to the experiment to be more complete, consistent and clear requirements model. This question meant to assess the eligibility of the approach to be applied in industry. The question allowed the participants to compare their models constructed during experiment to their current practice of security engineering. Figure 39 reflects the strong agreement among the FADES group members on that the approach allowed them to build better quality requirements models compared to their current practice. The majority of the B group (3 members) disagreed with the fact that B allows them to build better quality requirements models while the other two members answered “I don’t know”. The interviews described in the coming section show that the B group members who answered “I don’t know” were not able to assess the quality of their models and compare them with current practice due to usability issues of the B method. The FADES group members, on the other hand, were more confident about their assessment to the models they constructed in the experiment and were able to compare it to their current practice of security engineering. This positive attitude towards FADES is expressed in the interviews.

Question 10 assesses the practicality and tool support provided with the approach. This question was somewhat confusing to the participants since it combined two aspects (practicality and tool support) in one question. It would have been clearer if the question was separated in two questions. However, the interviews following the questionnaire clarified the question and discussed the answers with the participants. Figure 40 depicts the agreement of the FADES group members on the practicality and tool support of the approach. The B group members came with variance in their answers since two of them agreed and three disagreed. The interviews following the questionnaire focused on clarifying the question and discussing the answers of the B group to be able to explain the variance in the answers between agreement and disagreement. For the FADES group, the situation is clearer since their answers came into a pattern of agreement with the two issues of practicality and tool support. However, the interviews still focused on elaborating the answers of the FADES group members after clarifying the question in order to explain what it is exactly that they agreed with, practicality or tool support or both.

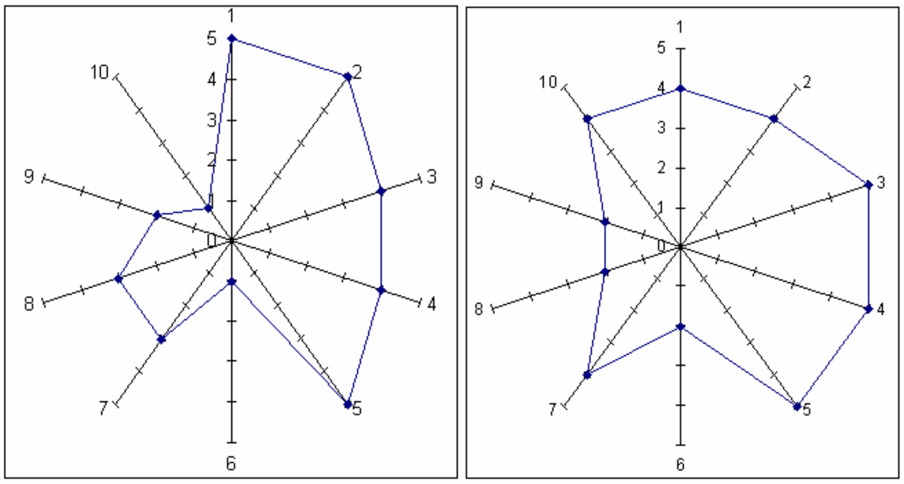
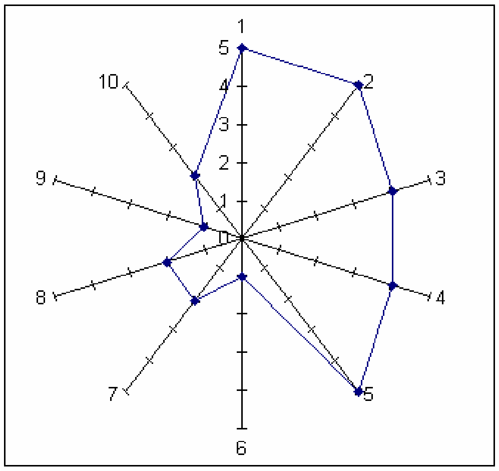


**Figure 39:** **Question 9, ability to build a better quality requirements model compared to currently used methods**

**Figure 40: Question 10, practicality and tool support**

Question 11 assesses the learnability of the approach in each group. This question assesses the ease of learn of each approach and the effort the participant needs to spend in order to start modeling with the approach without previous knowledge in it. Figure 41 depicts the answers of the 10 participants and shows consent in the FADES group as opposed to disagreement in the B group. The training sessions hold prior to the experiment to explain the approaches to participants align with the result obtained for question 11. The training session for FADES was planned for 2 hours and it actually went for 1.5 hours only while the training session for the B method was planned to 2 hours and actually took 2.5 hours.

Question 12 inquires about the convenience and easiness of use of the approach. Figure 42 illustrates the agreement of the FADES group regarding the question and the disagreement of the B group with it. The two groups have shown patterns for their answers to this question. The patterns came in two opposite extremes of agreement. The consent of the FADES group members to the question reflects their satisfaction with the application of the approach though new to them. In the other extreme, the B group members' disagreement with the question indicates their disapproval to the convenience of applying the approach and this has been clear in their elaboration on this question in the interviews as depicted below.



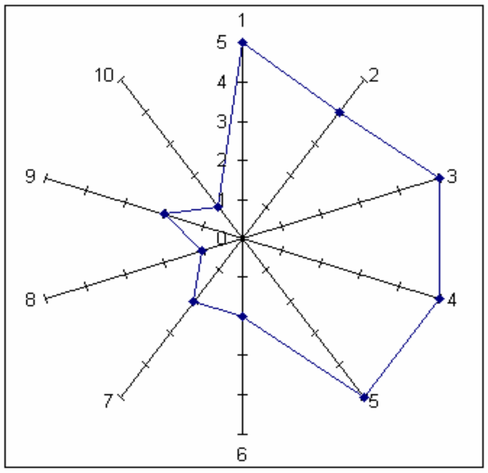


**Figure 41: Question 11, approach learnability**

**Figure 42: Question 12, convenience and easiness of application**

From the answers of the 12 questions described in this section, it is observed that there the FADES group members are positive towards the approach and they found it useful and effective in producing good security quality through a convenient and easy to learn and use approach. The pattern of the answers of the B group members, on the other hand, reflects their dissatisfaction with the usefulness and practicality of the B method in specifying security requirements due to its complexity and rigid formality that does not suit the requirements analysis phase characterized with some unknowns that are difficult to specify. Figure 43 summarizes the answers of the 12 questions for the two groups.

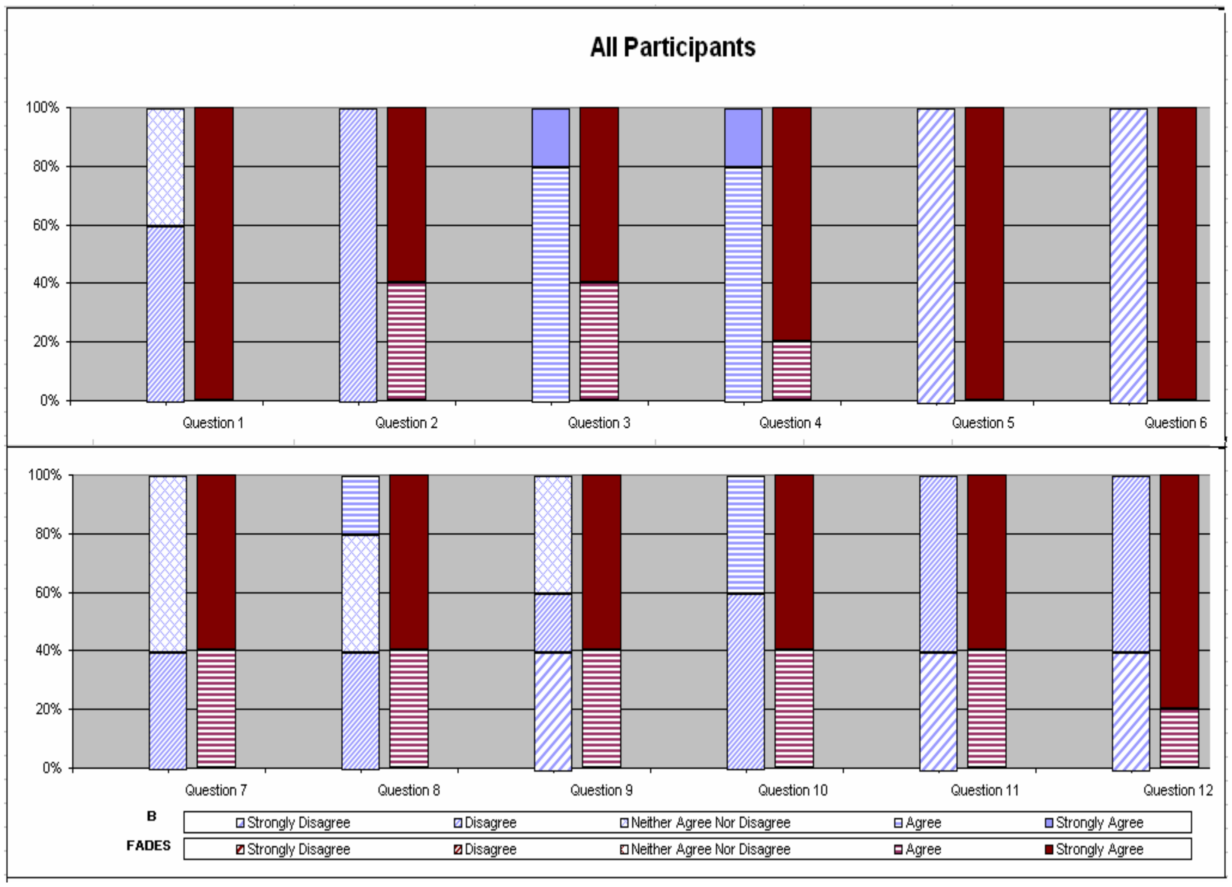


**Figure 43: Practitioners' Questionnaire Results**

The results obtained from the questionnaire reflect the assessment of the participant to the two approaches. These results need to be complemented with interviews conducted with the each participant following the questionnaire. The interviews meant to elaborate on the single, one-shot answer given for the multiple choice kind of questions.

### 5.2.4.2 Qualitative Results (Interviews)

While the questionnaire answers convey the general attitude of the participants towards each approach, the interviews were intended to dig into the individual experience of each participant and get his/her feedback about the strengths and limitations of the approach applied. The discussions that came up during the interviews were very helpful in evaluating FADES who reported their experience with learning and applying the approach. These discussions were also effective in answering some of our research questions like "How feasible and effective is the coupling between semi-formal requirements specifications and formal design an implementation in producing more complete, consistent and secure software?" and "What are the strengths and limitations of FADES?". The interviews reflected in more depth the the effectiveness of FADES compared to other formal methods in producing more complete, consistent and secure software. This led to further examination of the approach and indicated directions for future work.

The first nine questions target the organization of how to address systematic goal-directed requirements identification, refinement and elaboration in comparison with the B method by its own. The other three questions in the experiment questionnaire target the effect of the approach usability on the completeness, consistency and security quality of the resulting models.

Questions 1 and 2 inquired about the effectiveness and usefulness of the constructs provided by each approach to reason about requirements. The answers for the two questions came in agreement in the FADES group. The five participants of the FADES group justified their agreement with the two questions through highlighting and discussing how specific features of the approach enabled them to effectively reason about requirements. The FADES group participants first indicated that the goal-oriented KAOS approach used in FADES regards requirements as goals, which is close to the human thinking of requirements as goals for the system to be achieved in its provided functionality. The participants referred to goals as effective means to reason about requirements and relate requirements to each other based on the concept of goal achievement. The ability to reason about requirements is particularly useful and critical in the security engineering context in which security requirements need to be understood and reasoned about in the proper context while keeping the focus on completeness, consistency and clarity of requirements relate to each other in a way that allows for the detection of possible security vulnerabilities.

Second, they highlighted that the refinement mechanism that results in decomposing higher-level goals into subgoals enables reasoning about requirements at different levels of abstraction. The decomposition mechanism is close to the way people solve problems in general through decomposing large problems into smaller pieces that could be more easily and precisely solved. The participants, further, said that decomposing high-level goals into more fine-grained goals gave them sufficient space to rationalize about high-level goals that are very abstract and realize them more concretely through the decomposed subgoals. One of the participants indicated that the refinement mechanism enabled him to think gradually of what it is precisely that the system requirements aim to achieve. Another participant said “thinking of requirements in terms of goals that could be decomposed into subgoals helped me focus on the business semantics of requirements rather than getting involved in the modeling language constructs”. A third participant indicated that the decomposition of goals is specifically crucial to security requirements that are critical aspects of the software. That participant added that the provision of refinement patters for security goals in the KAOS framework facilitated the task of decomposing high-level security requirements into more fine-grained ones.

The decomposition of goals in FADES is an advantage of the approach to enable thorough reasoning of requirements. However, it could be argued that a software engineer with good knowledge of formal methods could use the refinement mechanism of formal methods to refine an abstract model into more detailed ones, which enable reasoning about requirements. It should be noticed that many years have passed since having formal methods and few practitioners have reached the level of knowledge in formal methods that could enable them to make use of the merits of formal methods. The explanation of this is that formal methods are not easy to learn and apply, especially for practitioners who are not used to develop software using mathematical models.

Third, the FADES group participants depicted that the responsibility assignment of the leaf goals to agents in the software system allowed for reasoning about the answers of the “WHO” questions that inquire about which agents in the system are responsible for achieving which goals. Moreover, the participants referred to the goal operationalization step in the requirements analysis process as an effective step to the realization of goals into concrete operations that could be propagated to later stages of development. One of the participants indicated that linking the triangle of leaf goals assigned to agents that achieve these goals through means of operations allows for thorough analysis of the requirements and their relations to system actors and functionalities. Another participant indicated that the goal operationalizion step is important since it is used as a criterion for the analyst to ensure completeness of security requirements, which enhances the requirements completeness. In the security engineering context, completeness of security requirements is a major factor in producing good security in the resulting software.

Forth, the participants of the FADES group designated the organization of goals in the structured goal graph format as one of the helpful features to reason about requirements. Two of the participants said “the visualization of requirements in a goal graph format strongly helps viewing the overall picture of the system requirements. This integrated view of the requirements allows for reasoning about the relations among requirements and their interoperability and interdependency”.

While the FADES group members showed consensus to questions 1 and 2, the B group members were in disagreement about the two questions. Interviews with the B group participants elaborated on the rationale for their disagreement in their answers to the questions. In general, the five participants of the B group raised the concern of the complexity of the formality involved in the B language. The participants referred to both the language constructs and syntax complexity and the complexity of representation of the requirements specifications using a mathematical model, which is not natural to their perception of requirements. They indicated that the complexity forced them to be more concerned with the language constructs over the business semantics of the requirements. This problem disabled them to reason about the requirements into their business context.

Four of the five participants indicated that they think the B language is not suitable to the requirements specification phase of software development. They said “this is especially true for requirements engineers who often do not have strong background in formal languages that let them bypass the language barrier and be able to think in the language that is inherently hard to learn and apply”. Further, the five participants depicted that they were not able to model requirements and analyzed them using the mathematical model and notation of the B language. It could be argued that with sufficient training, formal methods could be smoothly employed, but the current state of practice of formal methods goes against this argument since it is rare to employ formal methods in industrial projects.

Three of the participants indicated that specifying requirements using a mathematical notation did not enable her to view the relations among requirements and reason about each requirement with respect to the overall system structure. Another participant indicated that the B language did not provide sufficient flexibility to handle open questions and questions with no single specific answer that characterize the requirements phase. This could be justified by the fact that the rigid rigor encountered while using formal methods constraint the way software engineers could rationalize about requirements.

One of the participants designated the non-determinism provided by the B language as a construct giving some flexibility to the specifications, but in order to specify a certain aspect abstractly and non-deterministically; one needs to precisely answer multiple questions concerning the abstract format that describes that aspect space. That participant gave an example supporting his comment from the model he constructed. In his model, he wanted to specify the attributes of the purse object, he had to specify them as relations from the purse set to the types of these attributes. While the specification of the attribute as a relation provides some flexibility to make this specification more concrete at later stages of development, it did require the specification of the type early in the analysis. He indicated that this example shows that getting occupied with seeking specific and precise answers to some questions distract from reasoning at a higher level of abstraction that is appropriate to the requirements analysis phase. It could be expected that this situation could have been better if the participant was more experienced with the B language to the extent that he could think in the language. When that participant was further asked about whether the situation could be better with more training and experience, he highlighted that the difficulty of learning the language might make it difficult for one to become well-experienced with the language. However, if one reaches that level of experience, it could be easier to bypass the formal language barrier and focus more on the business semantics. This is particularly critical to the security engineering context where it mandates the security engineers to be very well-experienced with the formal language if they are to employ it in order to produce well-specified security requirements and high quality software security. This leads again to the same problem of the lack of expertise to apply formal methods in the security domain [1].

Questions 3 & 4 inquire about the effectiveness and usefulness of the constructs provided by the approach to capture environment properties. The answers of the two group members came almost in the same range of agreement in the experiment questionnaire. When interviewed, the FADES group members indicated that the KAOS requirements framework stimulated their reasoning about the environment of the software-to-be while modeling the requirements of the system itself. The participants referred to the effectiveness of KAOS in being able to integrate the environment properties into the requirements model both implicitly through incorporating constraints into the model and explicitly through the domain property construct. They further referred to modeling environment constraints in the form of operations' pre/post conditions and object invariants as an opportunity for considering these constraints early in development and propagating them to the later stages of development. These participants depicted the effectiveness of modeling environment constraints early in development as they referred to their previous experience in how ignoring these constraints result in building software that does not function properly in its deployment environment.

One of the participants in the FADES group said “the KAOS framework allows for reasoning about requirements in their proper context”. Another FADES participant said “the ability of KAOS to assign different responsibility among software/environment components allows for specifying the software-environment boundaries and interactions”. A third participant illustrated that KAOS incorporates the environment properties in the requirements analysis process during the obstacle analysis step in which exceptional conditions in the environment that may prevent critical goals from being achieved can be identified and produced in order to produce more consistent requirements model”. A forth participant referred to the ability of KAOS to explore different alternatives in goal refinement and responsibility assignments that lead to exploring the multiple alternatives of drawing the boundaries between the software and its environment.

In the interviews, the B group participants indicated that the B formal method provided them with effective constructs to model the environment constraints during requirements analysis. This includes the machine invariants that model the constraints on the machine state and the pre-conditions of the machine operations that change the machine state. However, the B group participants highlighted that modeling the constraints of the environment is not the only aspect of the environment. One of the participants depicted that she found it a limitation in the B method not to be able to investigate the “WHO” questions that result in modeling the actors in the environment and their relations to the requirements. This is specifically important to security engineering since it allows for specifying the internal agents among whom there might be internal attackers. Further, modeling the agents of the system allows for differentiating legitimate agents from possible external attackers. Agents’ modeling is one of the distinguished features of the goal-oriented paradigm for requirements analysis compared to other formal methods that lack this feature, which is even more important in the security context.

Another participant referred also to the fact that modeling environment constraints sometimes fall short with respect to modeling non-functional requirements that are heavily integrated with the environment and need dedicated techniques in the method to model them explicitly. Such techniques do not exist in the B method.

Questions 5 and 6 inquire about the effective constructs provided with the approach to identify and analyze security vulnerabilities. The answers of the two questions came in the two opposite extremes for the two groups in the experiment questionnaires. All the FADES group members strongly agree with the two questions while all the B group members strongly disagree with them. The interviews with the two groups clarified the reasons for the two extreme answers and highlighted the features of each approach that led to these answers.

The FADES group members referred to the obstacle analysis step in the KAOS security extension as a useful technique for performing threat analysis during the requirements phase. They considered this feature that identifies and resolves security vulnerabilities that early in development very promising in producing more secure software. The FADES group participants further indicated that the KAOS security extension provides a constructive procedure that they found very effective in identifying and resolving vulnerabilities. One of the participants in the FADES group said “threat analysis not only provides means for technical detection of possible security vulnerabilities, but also dedicates a step in the requirements analysis process for system analysts to reason about security vulnerabilities and their wider impact on the system”. Another participant highlighted that the obstacle analysis step allowed her to detect confidentiality and integrity threats to the messages being communicated in the electronic smart card system. That participant indicated that though her security experience is not that strong, she was able to detect confidentiality and integrity vulnerabilities that mandated the introduction of encryption/decryption and digital signatures in her requirements model. This reflects the effectiveness of the obstacle analysis in KAOS, especially in the security engineering context since it allows software engineers with no strong background in security to produce good quality security requirements models though the analysis of possible security breaches.

Another participant indicated that the threat analysis step in KAOS has stimulated his reasoning about possible security breaches in the system through identifying possible threats to individual goals followed by analyzing these threats with respect to the rest of the system goals and domain properties. That participant added that this allowed him to effectively put the security vulnerability into context. A third participant referred to the security attack patterns and resolution patterns provided with the KAOS security extension as a valuable source of stimulus to analyze and detect security breaches. Moreover, these patterns motivate reuse and saves effort spent in the threat analysis phase leading to more cost-effective products. A forth participant indicated that resolving security vulnerabilities early in development would save the losses that often result from security threats when security is dealt with after the fact.

The fifth participant who had a good experience in the maintenance of four projects indicated that the KAOS goal graph would enhance the maintainability of the system. The relations between requirements provide traceability information that shows which requirements are achieved by which operations in the system. Further, the automated transition from KAOS to B provides traceability information between requirements and design through the mapping between KAOS operations and their equivalent operations in B. It could be noted that the participants in the FADES group did not comment a lot on the transformation step from KAOS to B. This could be justified by the fact that the transformation was automated, so the participants did it seamlessly and they did not have much to say about it since they did not know how it happened. The participants only generated an XML from the Objectiver tool for KAOS and fed it into the Goal Graph Analyzer in order to generate the initial B model. This demonstrates the ability of FADES to facilitate the transition from requirements to high-level design, which is one of the most difficult steps in the software engineering life cycle [3, 96].

In the interviews, the B group members indicated that the security vulnerabilities they identified in their B models are due to their security knowledge and experience and not to the support of the approach. One of the B group members said “the lack of a constructive procedure for threat analysis in the B method leaves the production of good security quality to the expertise of the requirement engineer”. Another participant indicated that there is no difference between B as a formal method and the current informal practices of security engineering as far as threat analysis is concerned since both techniques do not dedicate a certain procedure for threat analysis.

Question 7 inquires about the ability to build near-complete model. This question has the purpose of obtaining the participants' own assessment of the models they constructed during the experiment. The results of the questions showed an agreement among the FADES group members. The FADES group members indicated that the simplicity and ease of use of FADES helped them overcome the barriers of the approach and focus on analyzing the requirements in their business context. The FADES participants referred to the visual notations of the KAOS framework and the Goal Graph Analyzer tool that automates the transformation from KAOS to B as means to facilitate learning and using FADES. They indicated that the facilitating means of learning and using assisted them in effectively building near-complete requirements models.

The FADES group members referred to the completeness criteria provided with the goal-directed KAOS approach employed in FADES as effective in providing assessment criteria for them to build near-complete models. One of the participants said "I can assess the model completeness when each leaf goal is assigned to an agent and operationalized through means of operations realizing the goals". Another participant indicated that the KAOS completeness criteria allow the requirements model to be communicated with the customers who can see evidence to the satisfaction of all their requirements.

The answers of the B group members to question 7 came between disagreement and "I don't know" answer. The B group members indicated that their focus on the complexity of B as a formal language has diverted them from focusing on the business security requirements resulting in requirements specifications that they are not satisfied with their completeness. The participants who answered "I don't know" depicted that they chose this answer because they were confused about the assessment of the model completeness and were more focused on the language syntax. They indicated, further, that they were unable to assess the completeness of their models due to the lack of a constructive procedure and completeness criteria to ensure completeness.

In his viewpoint, one of the participants in the B group highlighted that one of the limitations of the B method when used to specify requirements is the complexity of the resulting model to the extent that it could not be shown to and communicated with the stakeholders to get their feedback on its completeness and consistency. Another participant indicated that even though he thinks that he could build a more precise requirements model with a formal language like B given that he gets more experienced with the language, the lack of a visual notation that could be communicated with the stakeholders might lead to missing or conflicting requirements from the stakeholders' standpoint. A third participant said "based on my experience in customer interfacing, one of the most effective ways to build a near-complete requirements model is to get the feedback of the customers on the model in multiple iterations to ensure that the system will do what the customers precisely want".

Question 8 inquires about the ability of the approach to separate security concerns from the rest of the system requirements. This question is important to assess FADES since it is a security-specific approach whose objective is to provide a constructive procedure to build more secure products. The FADES group members emphasized their strong agreement with the ability of the approach to separate security concerns. The FADES group members indicated that modeling security requirements as goals elaborated and refined in separation from the rest of the system requirements with the assistance of the security patterns provided with KAOS security extension was effective in giving them an opportunity to focus on security concerns only. They also referred to dedicating a step to threat analysis and resolutions with the assistance of some patterns as this allowed them to pay more attention to security requirement at a moment over other system requirements.

One of the participants highlighted that KAOS allowed her to achieve the balance between separating security requirements and maintaining the dependencies between security requirements and the rest of the system. She referred to the KAOS object model constructed during the process as the common set in which security requirements and the rest of the system requirements intersect when manipulating and changing the same set of objects. Another participant said “I found the KAOS security extension very effective as it provides some steps and constructs that are specific to security requirements such as the threat analysis step and the patterns of security requirements and security threats. Based on my knowledge, that is the first approach I use that focuses specifically on security requirements that used to be handled late in development”.

The B group members came up with different answers to question 8; two participants disagreed with the assertion, two answered “I don’t know”, and only one agreed with the assertion. The B group members who disagreed indicated that the rigid formality of the B method did not allow them to think abstractly about security requirements in isolation from the rest of the system requirements. Moreover, they needed to ask about some low level details that are not security-specific but they found necessary for the formal specifications of security requirements. One of the two participants who disagreed with the assertion gave an example from his model when he found it necessary to specify the exception log of the electronic purse in order to log the messages of the money exchange scenario. The specification of the exception log had to be given in some level of details like the size and type of the exception log. The participant commented that the approach did not allow him to draw a line separating security concerns from the rest of the system requirements.

The other two members in the B group who answered “I don’t know” highlighted that they were confused between security requirements and other system requirements and they did not know exactly when to focus on security requirements and when to focus on other system requirements. The B group member who agreed with the question’s assertion indicated that his security experience in more than three industrial projects helped him focus on security concerns and separate them from the rest of the system requirements. He added that he still thinks that the B method considers security requirements like any other type of requirements and does not provide specific mechanisms to separate security requirements from the rest of the system requirements.

It could be justified that much of the variance in the answers to question 8's assertion in the B group is due to the fact that B is not definitive in departing security concerns from the rest of the system requirements. This is because B is not specific to security and it regards security requirements like other system requirements.

Question 9 inquires about the ability of the participants to build better quality models compared to their current practice of security engineering. As indicated above, better quality requirements model was defined prior to the experiment to be more complete, consistent and clear requirements model. This question meant to assess the eligibility of the approach to be applied in industry.

The FADES group members all agreed with the assertion in question 9 and indicated that their current practices of requirements engineering are based on informal and unstructured methods. They were very positive about employing FADES in their projects because they think that employing goal-directed modeling in FADES will help them build more structured and well-organized requirements models. One of the participants in the FADES group highlighted that the model he built with FADES was clearer than his current informal practice. That participant justified his viewpoint by the fact that with a KAOS model, each concept is defined only once resulting in no ambiguity. Another participant indicated that when he followed the KAOS methodology, it enabled him to build more complete and consistent requirements model in which each goal is justified by higher-level goals and refined into requirements that are placed under the responsibility of agents and operationalized in the operation model. A third participant referred to the ability of the approach to construct a model represented in an convenient visual notation that facilitates communicating the model to the stakeholders whose feedback greatly enhances the quality of the model.

Three of the B group members disagreed with question 9 while the other two of the group answered "I don't know". In the interviews with the B group members, they generally expressed their lack of understanding to the B language as a method for specifying requirements when compared to their current informal practice. One of the participants who disagreed with the question's assertion in the B group said "our current practice is informal but more flexible and suitable to the requirements analysis phase when there are questions that do not have definite answers. The B method, on the other hand, is based on mathematical models that need things to be specific and clear in order to be modeled, which is kind of difficult during requirements analysis".

The two participants in the B group who answered "I don't know" indicated that they were not able to assess the quality of their models and compare them to the current practice due to the usability concerns of the B method. One of the two participants who answered "I don't know" referred to the fact that he had to focus on the language syntax rather than the business semantics of the requirements. That participant added that his distraction from the business semantics led to a requirements model that he could not assess its quality and compare to his current practice of requirements engineering.

Question 10 inquires about the practicality and tool support of the approach. The FADES group members were in agreement with the question's assertion. In the interviews following the questionnaire, the FADES group members indicated that the approach is very practical and could be effectively applied to their industrial projects. One of the participants said "the KAOS framework is supported by a visual notation with a formal infrastructure that increases its robustness. This makes the approach practical from the perspective of the software engineers who are acquainted with modeling using visual notations".

The FADES group members referred to the Objectiver tool supporting the KAOS framework, which they used for constructing the requirements goal graph as a full-fledged tool for requirements analysis. Further, they added that the tool is aligned so well with the KAOS framework since it provides constructs for all the concepts of the framework along with the different ways to integrate them. One of the participants indicated that the Objectiver tool supports multiple views of the requirements model such as the goal graph model, the object model, and the operational model. That participant referred to the multiple views feature of the tool as very helpful in analyzing and communicating the requirements model from the different business perspectives of the stakeholders. It should be noticed that the Objectiver tool does not provide any automation for construction of the requirements model except for the generation of the requirements document and exporting the model into an XML format. The construction of the requirements model and the rationale behind the decision made in the model are the responsibility of the requirements engineer.

The FADES group members also referred to the Goal Graph Analyzer tool as facilitating the transition from requirements to design through automation. They highlighted that automation of the transition reduces the error-proneness of the transition and saves the development effort. One of the participants said "FADES provides means and tool support for organizing the security requirements using the KAOS framework that relaxes formality in the requirements analysis phase in order to pave the grounds for formal design. This makes the approach effectively practical especially for security requirements that used to be ignored due to the lack of practical approaches for security engineering". Another participant highlighted that the transition from KAOS to B provides the analysts with means to automatically construct an abstract formal design model from the requirements model that is thoroughly analyzed using the KAOS security extension without the cost and effort of formal specifications.

Two of the B group members agreed with question 10 while the other three members disagreed. Interviews with the B group members showed that people who disagreed with question 10 mainly disagreed with the practicality of the approach in being effective for requirements analysis while they think that the B tool support is quite strong. One of the participants who disagreed with the assertion in question 10 indicated that the B method is impractical to specify requirements since it needs people who can think in the language, which is not highly available in the software engineers market. Another participant highlighted that B is not practical to specify requirements that are characterized by being imprecise and vague. A third participant added that the B method might be more suitable to the design and implementation phases during which all the questions propagated from requirements are precisely answered.

The other segment of the B group that agreed with the assertion in question 10 clarified their answer by indicating that they agreed with the strong tool support part of the question, but disagreed with the practicality of the B method as an effective approach to analyzing security requirements. One of the participants who agreed with question 10 said "the rigid formality associated with the B method makes it less practical especially for the requirements phase that favors reasoning and structure of requirements over formality". This means that the answers of the B group members to question 10 are almost aligned together in an agreement with the tool support of the approach and disagreement with its practicality.

Question 11 inquires about the ability of the approach to be learned. All the FADES group members agreed in their answers to the question. The FADES group members elaborated on their agreement with the assertion in question 11 in the interviews depicting that the employment of KAOS with a visual notation and relaxation of formality to analyze requirements is a clever choice because it makes FADES easy to learn and apply. One of the participants in the FADES group said "I was able to apply FADES given only one training session on the approach and I am personally satisfied with the quality of the output model I produced". Another participant indicated that the KAOS framework is not difficult to learn because it uses a visual notation, which is one of the most preferred formats of presentation to software engineers.

All the B group members disagreed to the assertion in question 11, and they elaborated on their answers in the interviews. They expressed the difficulty they encountered in learning the approach due to the formality entailed in the B method. One of the participants highlighted that they needed longer time in the training session that the planned time. Another participant indicated that after the training session, he was not confident of his knowledge in B and he was not sure that he can apply it directly. A third participant said "after the training session and when I started the experiment, I did not feel I was able to write in the language and I still need more knowledge and training".

Question 12 inquires about the convenience and easiness of applying the approach. All the FADES group members agreed with the question. In the interviews with the FADES group members, they indicated that the relaxation of formality in FADES during requirements analysis along with the ability to organize requirements in the goal graph structure make the approach convenient and easy to apply. Moreover, the automation of the transition from KAOS to B makes the approach easier to apply since the system analyst does not have to be aware of the transformation rules.

All the B group members disagreed with question 12. In the interviews, the B group members showed their dissatisfaction with the use of the approach and they justified this by the complexity of the rigid formality of the B method and the difficulty of specifying requirements using a mathematical notation. One of the participants in the B group highlighted how difficult it was for him to map requirements to the mathematical notation of the B method.

While the multiple-choice questionnaire and the interviews with the participants reflected a positive attitude towards FADES, the participants outlined some limitations to the approach during their elaboration on the approach in the interviews. One of the participants in the FADES group indicated that the KAOS framework did not give a specific procedure for selecting which goals to be decomposed and which goals to be weakened or ignored. That participant added that the limitation of the KAOS framework to document the decomposition decisions made during the goal elaboration and refinement step on the goal graph itself made him unable to review his decomposition decisions. He indicated that this limitation would mostly affect the precision of the impact analysis needed when changes to security requirements are introduced.

Another participant in the FADES group highlighted a limitation of the KAOS framework in assigning priorities to the conflicts of a certain goal especially that these conflicts include the conflicts systematically detected using the obstacle analysis technique of the KAOS framework and the conflicts depicted by the stakeholders in the feedback sessions that communicate and review the requirements model with the stakeholders. That participant said "I did not experience the conflicts to a goal that arise among stakeholders in the experiment. However based on my experience, I expect that the approach falls short in prioritizing the conflicts of a goal and their resolutions based on the priorities of the stakeholders".

A third participant argued that since KAOS does not provide a formal guarantee for the requirements model completeness and consistency, missing and conflicting requirements might propagate to the later phase of development in FADES. She said "However, the requirements phase is typically characterized by incompleteness and inconsistency of requirements and FADES pays good attention to this problem and tries to limit its effect through the generation of the acceptance test cases that might assist in detecting any propagated requirements problems".

The interviews have clarified and explained in more depth the answers obtained in the experiment questionnaire. The interviews and the questionnaires need to be complemented with the assessment of the models produced by the participants in each group as described in the coming section. The results obtained from the models assessment are valuable in comparing the performance of the participants in the two groups and in comparing the assessment of the participants to the two approaches with the quality of the models they produced and if the two assessments are aligned.

### 5.2.4.3 Requirements Models Assessment

Ten requirements models have been assessed using the four metrics of completeness, consistency, security quality and development cost. Half of the models have been constructed using FADES in which security requirements are elaborated and analyzed using the KAOS security extension. The KAOS requirements models are transformed to B through the Goal Graph Analyzer tool to produce the equivalent abstract B models. The other half of the ten requirements models have been constructed by the B group members who directly specified the models in B and produced abstract B models of security requirements.

The models' assessment addressed two research questions:

1. How does the integration of semi-formal and formal methods in FADES compare to purely applying formal methods from the completeness, consistency and security quality perspectives?

2. How much is the cost of development affected in FADES as opposed to the cost in other formal methods?

The first research question is answered in the models' assessment through comparing each model to a reference model, which is the model constructed for the electronic smart card case study to demonstrate FADES as described in section 4.2 in this dissertation. The comparison with the reference model allows for obtaining concrete numerical values for the completeness, consistency and security quality metrics in order to compare FADES to other formal methods. The second research question concerned with the cost of development has been answered by our recording to the development time of each model during the experiment and the results are described below.

The reference model consists of four security requirements for which six conflicts have been detected and seven security vulnerabilities have been identified. These numbers in the reference model have been defined prior to the experiment and are used to calculate the percentage completeness, consistency and security quality in the participants' models. The development cost metric measured by the development time of each participant has been recorded by the research team and not by the participant.

The number of participants in the study is quite small. So, the values obtained from the assessment of the resulting models are based on simple statistical descriptions, data trends and comparisons of percentages. Further, we suspect that the results would not be statistically significant due to the small size of the selected sample. Rather, the results give notable indications about how FADES is expected to perform when it is applied to larger projects. The evaluation of the domain experts to models assessment results in section 5.3 raises the confidence in their scalability.

Figure 42 depicts the assessment of the ten models in the FADES group and the B group with respect to the requirements completeness metric. Completeness has been measured by the amount of the security requirements covered by each participant's model with respect to the amount of requirements covered in the reference model. The bar chart in Figure 44 compares the assessment of the models in each group with average completeness of 97% in the FADES group with respect to the reference model. 60% of the FADES group members were able to achieve 100% model completeness meaning that they have achieve the same level of completeness as the reference model. The other 40% of the FADES group members have exceeded 90% completeness. On the other hand, the average completeness percentage in the B group has been 74% with none of the group achieved 100%.

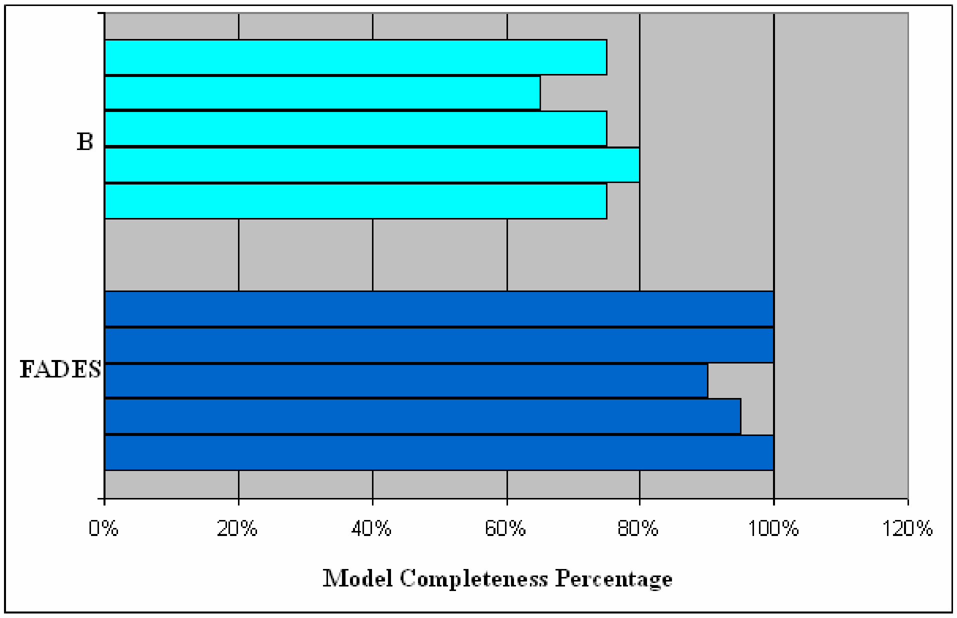


**Figure 44: Requirements Completeness**

The chart in Figure 44 shows better completeness of the models constructed with FADES over those constructed with B by about 23% on average. Further, the standard deviation of model completeness in the FADES group is 0.04 showing a tiny level of variance among the performance of the participants in the FADES group that are all almost close to 100% completeness. The standard deviation of model completeness in the B group is 0.05, which also reflects very small variance around the average completeness, 74%, which is already not high. Given that all the subjects in each group have the same base line of knowledge and experience and the variance in completeness in each group is tiny, the completeness results strongly support the dissertation hypothesis that FADES is capable of producing more complete security requirements compared to other formal methods employed in security engineering such as the B method.

Figure 45 illustrates the assessment of the participants' models with regard to the requirements consistency metric. Requirements consistency is defined by the absence of semantic conflicts between requirements and is measured by the number of conflicts detected in the model since the more conflicts detected and resolved in the requirements model, the more consistent the resulting model will be. The bar chart in Figure 45 shows an average consistency of 92% in the FADES group with 40% of the group achieved 100% consistency while the other 60% have exceeded 80% consistency. On the other hand, the average consistency in the B group was 56.7% while none of the group achieved 100% consistency and the maximum consistency achieved in the B group was 66.7%. This means that the average model in the B group has propagated 43.3% of the conflicts between requirements undetected to the later phases of development like design, implementation, and testing when it is more costly to resolve them.

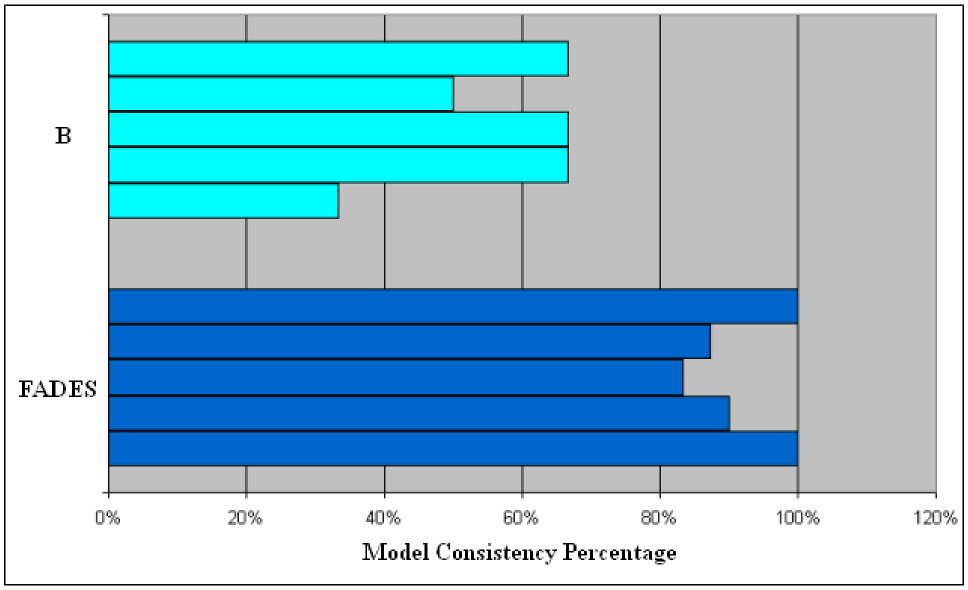


**Figure 45: Requirements Consistency**

The standard deviations in both the FADES and B groups are 0.08 and 0.14 respectively showing very slight variance of the consistency percentages around the average values. This result is aligned with the participants' assessment to both approaches as reported in the interviews. As subjects reported in the interviews, the obstacle analysis feature of FADES and the organization of requirements in the goal graph allowed people in the FADES group to detect inconsistencies and resolve them better than in the B group.

Figure 46 describes the assessment of the participants' models with respect to the security quality metric, which is measured by the number of vulnerabilities detected and resolved in the requirements model. The more vulnerabilities detected and resolved during requirements compared to the vulnerabilities detected in the reference model, the better the security quality of the model since vulnerabilities are detected and resolved very early in development with the least cost. The bar chart in Figure 46 shows an average quality of 88.6% in the FADES group with 40% of the group were able to detect 100% of the vulnerabilities detected in the reference model. The average security quality in the B group was 48.5% indicating that more than 50% of the security vulnerabilities existing in the reference model have been left undetected and will propagate to design and implementation at which time resolution is more costly. It could also be observed from Figure 45 that none of the B group was able to detect all the vulnerabilities of the reference model and the maximum security quality was 57.1%.

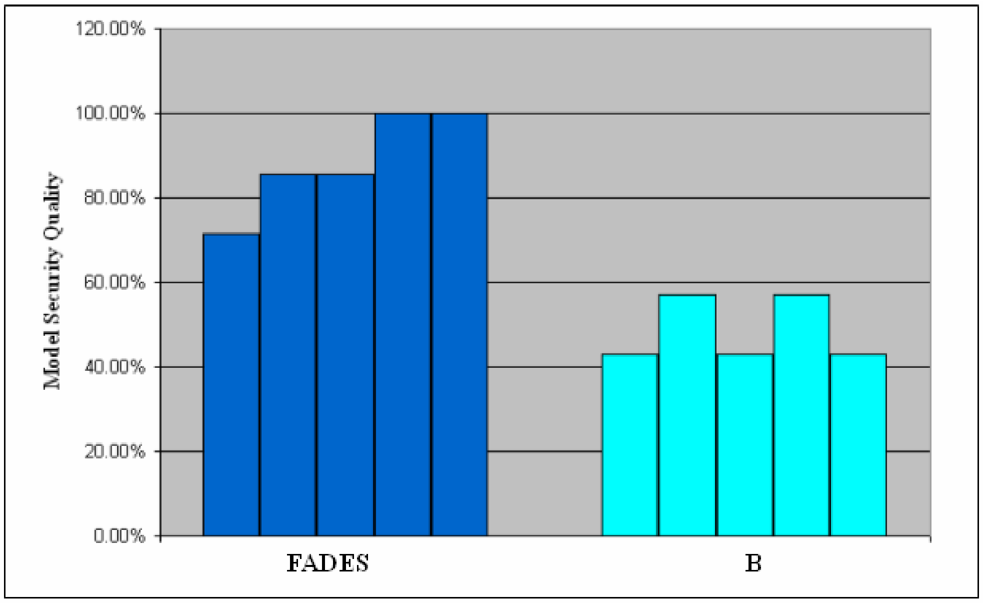


**Figure 46: Security Quality**

The standard deviation of the security quality metric in the FADES group was 0.12 while it was 0.07 in the B group. The small standard deviation values in both groups reflect the closeness in security quality of all the models in each group to the average value. Therefore, the models constructed with FADES had better security quality than those constructed with the B method by about 40% on average.

Figure 47 illustrates the development cost of each of the ten models. The development cost values plotted in the below graph are in hours measured by the development time each participant spent to develop the requirements model. The development cost metric focuses only on cost from the effort perspective rather than from a financial perspective. However, the effort cost could be used to estimate the financial cost.

The average development cost in the FADES group was 3.3 hours while it was 3.95 hours in the B group. This shows that FADES participants managed to develop their models with less cost than the B participants by about 40 minutes on average. This result is aligned with the dissertation hypothesis in that it demonstrates that FADES is more capable of producing better quality requirements models at less cost.

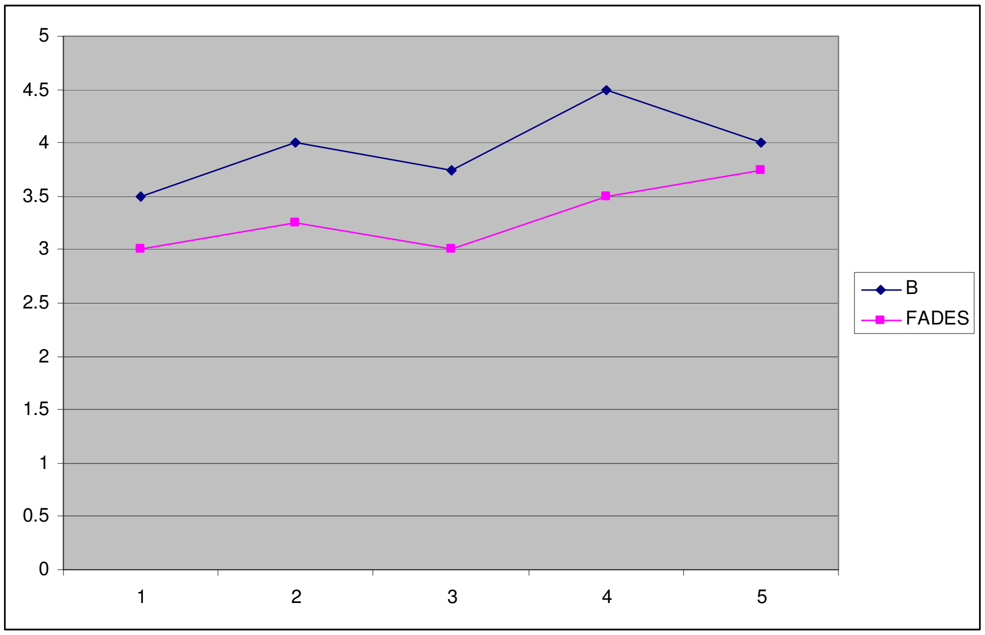


**Figure 47: Development Cost**

The results of assessing the participants' models using the four metrics of completeness, consistency, security quality and development cost are summarized in Table 5.

| | **Requirements completeness** | **Requirements consistency** | **Security quality** | **Development cost** |
|---|---|---|---|---|
| **Subjects applying FADES** | | | | |
| Subject 1 | 100% | 100% | 71.4% | 3 |
| Subject 2 | 95% | 90% | 85.7% | 3.25 |
| Subject 3 | 90% | 83.3% | 85.7% | 3 |
| Subject 4 | 100% | 87.3% | 100% | 3.5 |
| Subject 5 | 100% | 100% | 100% | 3.75 |
| **Subjects applying the B method** | | | | |
| Subject 6 | 75% | 33.3% | 42.8% | 3.5 |
| Subject 7 | 80% | 66.7% | 57.1% | 4 |
| Subject 8 | 75% | 66.7% | 42.8% | 3.75 |
| Subject 9 | 65% | 50% | 57.1% | 4.5 |
| Subject 10 | 75% | 66.7% | 42.8% | 4 |

**Table 5: Summary of Participants' Models Assessment**

### 5.2.5 Comparative Analysis Between FADES and B

The multiple choice questionnaires distributed to the experiment participants and the interviews held with them after the experiment had the objective of assessing the participants' experience of applying the approach whether it is FADES or B. The experiment participants are practitioner software engineers with reasonable industrial experience. Therefore, obtaining their evaluation of FADES is valuable in identifying and emphasizing the potential strengths and weaknesses of the approach.

There has been a pattern for the answers in both the FADES and the B groups for most of the questions especially the ones concerned with the approach effectiveness, practicality and usability. It could be observed that the FADES group members had very slight variance in their answers to each question. Their answers to all the questions were either "strongly agree" or "agree". All the questions were asking about the effectiveness and usefulness of the features of each approach in building a complete, consistent and more secure requirements model. This means that the FADES group members were in consensus about the effectiveness and potential strength of the approach to handle security-specific elements of software. The interviews following the questionnaire provided a better opportunity for participants to elaborate on their positive opinion about the different aspects of FADES.

Like the variance that was slight in the experiment questionnaire answers in the FADES group, the variance in our assessment to the participants' models was very slight as well from the completeness, consistency and security quality perspectives. The FADES group participants' models were in general better in the three assessment metrics, which shows alignment between the positive attitude of the participants toward FADES and our assessment to the participants' models.

The B group members, on the other hand, showed a pattern of disagreement with the usefulness of the B method to elaborate and analyze security requirements. The answers of the group members to some questions came with slight variance while it showed more variance for some other questions. For example, their assessment to the effectiveness of the B method in analyzing possible security threats consistent among all the group members who answered "Strongly disagree". However, the group answers came with some variance to question 9 that inquires about their assessment to the quality of the requirements models they constructed with the B method compared to their current industrial practice of requirements analysis. The interviews following the questionnaire clarified the different perspectives of the participants in the B group to justify the variance in their answers to some questions.

The coming paragraphs integrate the participants' answers to the multiple choices questions with their elaboration in the interviews and our assessment to the participants' models to analyze the results. Further, justification to the results is provided so that the potential strengths and limitations of FADES when compared to other formal methods could be highlighted.

Questions 1 and 2 inquire about the effectiveness and usefulness of the constructs provided by the approach to reason about requirements. The two questions addressed two of the research questions raised in this dissertation, which are how FADES is effective in constructing better software security, and how

FADES compares to other formal methods in the resulting software security quality. The ability of the approach to provide effective constructs to reason about security requirements raises the opportunity of the approach to produce better software security. This is because reasoning about requirements result in deeper analysis of the requirements; therefore, more comprehensive and correct requirements model. Further, a well-analyzed and reasoned about requirements model improves the quality of later phases of development [2]. The interviews with the participants, as indicated in the coming paragraphs, favor FADES over using the B method by itself in its ability to effectively reason about requirements and deeply analyze them into the proper business context.

The questionnaire answers to questions 1 and 2 and the interviews show that the FADES group members agreed while the B group members disagreed. The resulting models assessment supports the position of both groups. The models of the FADES group participants achieved an average completeness of 97% as opposed to 74% in the B group, and average consistency of 92% as opposed to 56.7% in the B group. Further, the models of the FADES participants demonstrate that the participants constructed requirements models that are thoroughly analyzed and reasoned about and this is reflected in the completeness of their models. Most of the FADES group members were able to achieve 100% completeness while none of the B group was. This indicates that FADES allowed participants to reason about requirements in order to cover them all while the B method did not provide enough support to participants to make rationale about their requirements analysis. The average consistency of the FADES group models also show that the FADES participants not only constructed complete models, but correct ones from the stakeholders' standpoint while the B participants' models resulted in less correct ones.

Some of the interview answers explained this result and it could be referred to two reasons. First, FADES relaxes formality during requirements allowing analysts to focus on and reason about the business semantics of requirements rather than getting involved in the modeling language constructs. Second, the construction mechanism of the requirements goal graph in KAOS allows for refining high level goals into more fine-grained subgoals in which the analyst is given a space and focus to justify how the subgoals result in the achievement of the high level goals. This type of mental activity imposes effective reasoning about requirements.

The disagreement among the B group members to questions 1 and 2 and the results of their models could be justified by the fact that rigid formality of the B language and the complexity of the mathematical notation associated with it made the mission more difficult to the B participants. Further, the lack of knowledge and experience with the language disallowed them from making good use of the language. It could be expected that if the participants in the B group had been more knowledgeable about the B language, they still would have disagreed with the questions because the language lacks the reasoning features, but they would have constructed more complete and consistent requirements model. This expectation is justified by the fact that if the participants had been able to think in the language, the barrier of the modeling requirements using a mathematical notation would not have been that much effective on the resulting models. However, the application of the Z language to the electronic smart card system [6] demonstrates that even with good knowledge of the formal language, the developers of the case study missed the message integrity requirement. This could be justified with the fact that the lack of a constructive procedure to reason about requirements and completeness criteria to evaluate the requirements model by the developer might result in missing some requirements even with experienced engineers.

Questions 3 and 4 inquire about the effectiveness of the constructs provided by the approach to capture environment properties. The two questions address the same research questions as questions 1 and 2, which are concerned with the effectiveness of FADES and how it compares to other formal methods in producing better software security. The ability of the approach to capture environment properties allows the requirements engineer to analyze and model the environment of the software and its constraints early in development so that these constraints are not violated throughout the development. This enhances the comprehension of the resulting model and reduces the possible problems that might arise when the software is deployed into its environment. The interviews show that FADES is similar to the B method in being able to model the environment constraints early in development. However, FADES provides explicit constructs to the environment properties making the modeling more effective.

The questionnaire answers and the interviews showed agreement in both the FADES and the B groups. The models of the two groups reflected their ability to capture environment properties in both approaches through object invariant predicates and operations pre conditions. In the FADES models, domain properties were more explicit since the participants employed the DomProp KAOS construct used to model domain properties. This means that FADES is comparable to other formal methods with respect to domain properties while FADES is distinguished in being able to model such properties explicitly. Providing an explicit construct to capture domain properties in KAOS draws the attention of the requirements engineer to the need and importance of modeling the environment in which the software will be deployed.

Questions 5 and 6 inquire about the effectiveness of the constructs provided by the approach to identify and analyze security vulnerabilities. The two questions address the same research questions as questions 1. 2. 3, and 4 that are concerned with how FADES is effective in constructing better software security and how it compares to other formal methods from the completeness, consistency and security quality standpoints. However, questions 5 and 6 focuses on the better security quality part of research questions since the two questions inquire about the effectiveness of the threat analysis feature performed during requirements analysis. The more security vulnerabilities detected during requirements analysis, the fewer the ones propagated to later phases of development and the less the cost of correcting them. This implies a better overall security of the resulting software compared to current practices of security engineering. Questions 5 and 6 also address the research question of what the strengths of FADES. The questions address how effective the threat analysis of requirements, which is one of FADES substantial features, is from the standpoint of experienced software engineers who would be applying FADES.

The answers of questions 5 and 6 came in the two opposites since the entire FADES group strongly agreed and the entire B group strongly disagreed with the questions. The resulting models also reflect the position of the participants. The average security quality measured by the number of security vulnerabilities detected in the model is 88.6% in the FADES group while it is 48.5% on average in the B group. Two participants in the FADES group achieved 100% security quality meaning that they were able to detect all the vulnerabilities detected in the reference model while the maximum quality achieved in the B group was 57.1%. The answers of the questions in the questionnaire and the resulting models emphasize the strength of FADES in employing the KAOS security extension that performs threat analysis during requirements elaboration and analysis. The results depicted in questions 5 and 6 shows the effectiveness of the obstacle analysis technique provided by KAOS in identifying and resolving possible security vulnerabilities early in development with minimal cost. In the other formal methods, there is no constructive procedure to detect security breaches, and it is totally dependent on the expertise of the requirements engineer and system analyst to identify such breaches. Propagating security vulnerabilities undetected to later stages of development increases the cost of resolving them.

Question 7 inquires about the ability to build near-complete models, and the main purpose of this question was to get the participants' assessment to the completeness of the models they built. This question directly addresses the research question of how FADES compares to other formal methods from the completeness standpoint. The entire FADES group agreed with the question while the B group participants were between disagree and "I don't know". The B group participants justified their answers in the interviews by indicating that they were either not satisfied with the completeness of their models or were confused about assessing the models. When comparing the participants' assessment to our own assessment to the same models from the completeness perspective, it could be noticed that the FADES group members were in general satisfied with the completeness of their models and their models reflected this satisfaction. The assessment of the FADES participants' models showed that 60% of the group participants were able to achieve 100% completeness while the other 40% achieved more than 92%. The three participants who achieved 100% completeness in their models are the same participants who strongly agreed with question 7. Their discussion in the interviews concerning the completeness issue reflected the effectiveness of the completeness criteria provided with the KAOS framework to give an opportunity to the requirements engineer to accurately assess the completeness of his/her requirements model. The B group participants, on the other hand, were dissatisfied with the completeness of their models that reflected this dissatisfaction since the maximum completeness achieved was 80% and the average was 74%. The B method like other formal methods does not provide means to assess the completeness of the requirements specifications, and this shows the distinction of the KAOS method when compared to other requirements engineering approaches.

Question 8 inquires about the ability of the approach to separate security concerns from the rest of the system requirements. Question 8 validates the research question of what the strengths of FADES are and how effective it is in handling security requirements. Question 8 depicts the participants assessment of a distinguished feature of FADES, which is the approach being security-specific and being able to handle security concerns in isolation from the rest of the system requirements.

The answers of this question and the following interviews with the FADES group reflected their agreement with the ability of the approach to separate security concerns. The participants outlined the features in FADES that enabled the separation of security concerns such as the security patterns available for elaborating security requirements and for threat resolution. The participants were enabled to produce better security using FADES since the approach is security-specific and supports means to constructively manipulate security concerns. Most of the B group, on other hand, disagreed with the ability of the B method to separate security concerns from the rest of the system requirements and this partially justifies the less security quality in the B group models. The results of question 8 refers to the strength and distinction of FADES in focusing on security-specific element of software in order to produce better security quality and enhances the current practice of security engineering.

Question 9 inquires about the ability of the approach to build a better quality requirements model compared to the currently used methods from the completeness and consistency perspectives. Question 9 addresses the research question of how FADES compare to other formal methods in building better quality software security from the completeness, consistency and security quality standpoints. Further, question 9 like question 7 attempts to obtain the participants' own assessment to their resulting models and compare it with our assessment to these models.

The entire FADES group agreed with question 9 and gave valuable comments during the interviews outlining the features of the approach that justify their agreement. Most of the B group members disagreed with the question and elaborated on the limitations of the B method compared to their current practice of security engineering to justify their disagreement. The assessment of the models of the two groups showed that the FADES group participants were able to produce better quality requirements models from the completeness and consistency perspectives compared to the B group participants. The FADES group achieved 23% better average completeness than the B group, given that the two groups have no previous knowledge about FADES and B. Further, the FADES group achieved 35.3% better average consistency compared to the B group. These results align with the answers of the two groups to question 9 highlighting the fact that FADES is more capable of constructing better quality requirements model using the KAOS security extension compared to currently used methods like the B formal method.

Questions 10, 11, and 12 inquire about the practicality, tool support, learnability, convenience, and ease of use of the approach. This punch of questions assesses the usability of the approach. However, these questions are subjective and cannot be directly inferred from the participants' models. The subjectivity of the questions could be justified by the fact that the participants assessing the usability of the approach are experienced software engineers and have assessed the usability of similar approach. Further, the discussion in the interviews increased the objectivity of the question when the participants explained the reasons for their usability assessment to each approach.

The three questions indirectly address the research questions of how effective FADES in building better quality security and how it compares to other formal methods from the completeness, consistency and security quality standpoints. The rationale for the indirect addressing of the research questions by the three usability questions is that the resulting software security completeness, consistency and quality are a function of the approach usability. The more usable the approach is from the users' standpoint, the more effective they can utilize its features to produce better completeness and consistency and security quality. It is not usually the case that more usable approaches are able to produce better results compared to less usable approaches. In the case of FADES, the comparison is between the B formal method by its own, which is complex and needs a lot of experience to apply and FADES that still relies in its core on formal methods. However, FADES provides a more usable interface in the goal-oriented KAOS requirements framework and automates the transition from KAOS to B so that the more rigid formality comes at later and more concrete phases of development, which are design and implementation.

The entire FADES group agreed with the three questions. Some of the B group agreed with the tool support aspect while the entire group disagreed with the rest of the usability aspects. The usability of FADES has been promoted by the participants over the B method and this result is an expected one since the KAOS framework employed in FADES provides a semi-formal interface with a visual modeling notation as opposed to a rigid formal mathematical model in B.

From the above discussions during the interviews, it could be observed that the participant's own assessment to each approach was aligned with our assessment to their models. This alignment comes as a result of the maturity and good experience of the participants as software engineers who have practiced the different disciplines of software engineering and assessed similar approaches. Both the participants' assessment to FADES and the resulting models reflect the strong potential of FADES to provide an effective and useful solution to the problem of poor security that mostly result from ineffective practice of security engineering.

Despite the fact that ten models are not sufficient to tease statistical significance, the scalability of the results obtained from the empirical study could be argued for the favor of substantiating FADES. The coming section provides a qualitative argument concerning the limitations of the results and how they could be generalized and scaled for larger size projects and larger number of participants.

## 5.3 Second Experiment: Domain Experts

This experiment is concerned with examining FADES from the academic and industry experts standpoint. The participants in this experiment are experts in the formal methods and security domains where FADES could be situated from the research perspective. The selected experts have the scientific background and the research experience in applying formal methods to high-assurance systems with security being a crucial concern to such systems. The experience of the domain experts in the applied formal methods area increases the confidence in their evaluation. The experiment is also meant to strengthen the results of the practitioners' experiment (section 5.2) by asking the domain experts about the scalability of the results obtained in the practitioners' experiment when FADES is applied to larger enterprise projects. Further, discussions with the domain experts about the strengths and limitations of the approach highlighted some future research directions.

The initial idea for gathering the experts' assessment was to prepare a comprehensive survey of FADES to mail it to the experts. However, since the academic and industry experts were not yet informed about the approach, the survey had to be supplemented with reading material explaining FADES. This idea was not practical given the tight schedules of the academic and industry experts.

Another idea, to visit and present FADES face-to-face to selected academic and industry experts. Visiting people in person ensured having their attention for at least the duration of the visit, which is around 75 minutes. However, the fact that the experts participated in the study were distributed in different countries in the United States of America and Europe made the idea infeasible.

Finally, the most practical and feasible approach to carry out the study was to interview the participants through the phone and video conference. The next challenge was to develop an approach for presenting the approach and collect results within the allocated time. A presentation about FADES consisting of 30 slides has been given to each participant at the beginning of the interview. Following the presentation, a two page multiple-choice questionnaire was done online with each participant so that results are guaranteed to be collected.

## 5.3.1 Experiment Questions and Participants

The experts' experiment involved 8 experts occupying academic and research positions and specialized in the domains of applied formal methods, developing tools for formal methods, security, software testing and reliability, evolving critical systems, verification and testing of critical systems, and model-driven engineering. Each expert has been interviewed for 75 minutes. At the beginning of the interview, each expert was given a presentation about FADES, how it has been demonstrated on cases studies and the results obtained in the practitioners' experiment.

Following the introduction of FADES, each expert has been given a questionnaire consisting of 11 quantitative multiple-choice Likert-scaled questions and 2 qualitative open-ended questions. The multiple-choice questions had the choices ranging from strongly agree to strongly disagree. Since the experts were asked to answer the questions during the interviews, they were able to give some comments on their choices for the multiple-choice questions.

The multiple-choice questions addressed three issues about FADES namely the effectiveness of the different features of the approach, the comparison of approach to the state of the art and practice in security engineering, and the scalability of the results obtained in the practitioners' experiment. The multiple-choice questions and the open-ended one are depicted in Table 6 and detailed in Appendix B.

The open-ended questions inquired about the scalability of the practitioners' experiment results and the strengths and limitations of FADES. The discussions about the strengths and limitations of the approach highlighted some directions for future research in FADES.

| **Likert-Scaled Questions** | |
|---|---|
| To Determine … | Question (s) Used |
| Effectiveness | FADES provides effective means to sufficiently structure and reason about security requirements. |
| | FADES employs the goal-oriented KAOS framework during requirements analysis. To what extent would you agree that the KAOS security extension is effective in constructing a reasonably complete, consistent, and secure requirements model at lower cost compared to other formal and informal requirements engineering methods? |
| | FADES could effectively focus on security concerns and separate them from the rest of the system requirements while maintaining the dependencies between security and non-security requirements. |
| | The threat analysis performed on the requirements model could enhance the security quality of the resulting software product. |
| | The automated transformation from KAOS to B in FADES provides an effective means to bridge the gap between security requirement and formal security design. |
| | The generation of acceptance test cases from the requirements model could raise the confidence in the resulting implementation as it complies with the requirements model. |
| | Given the tools for KAOS and B, both of which are strong, would this be sufficient to do effective security engineering with FADES. |
| Comparability | Overall, FADES is capable of effectively engineering security concerns and produce better software security compared to the current state of practice in security engineering. |
| | Overall, FADES is capable of effectively engineering security concerns and produce better software security compared to current state of the art in research of security engineering. |
| | Based on what you understood about FADES, FADES is more cost-effective than engineering security requirements using formal methods by themselves. |
| Results Scalability | To what extend would you agree that the results obtained from the comparative empirical study that compared FADES to the B formal method would scale when FADES is applied to enterprise industrial projects? |
| **Open-Ended Questions** | |
| What are the strengths and limitations of FADES? | |

**Table 6: Experts' Questionnaire**

Some of the multiple-choice questions in the experts' questionnaire are similar to some questions in the practitioners' questionnaire. The similarity between the questionnaires of the two experiments is mainly found in the questions that address the effectiveness of the different features of FADES.

### 5.3.2 Experts' Experiment Results

This section outlines the results collected during the interviews with the academic and industry experts. The results include both the formal answers of the questions listed in the questionnaire in Table 6 and the informal comments collected during each interview. Though most of the questions are in the form of multiple-choice questions, most of the experts gave justifications to their choices in each question. This is due to the fact that the questionnaire was hold online and there was an opportunity to ask each expert about his choice.

Figure 48 depicts the overall results obtained from the 8 experts concerning the 11 multiple-choice questions. The results in Figure 48 show that all the 8 experts either agreed or strongly agreed to the following aspects:

- The ability of FADES to reason about requirements and organize them in well-structured format (question 1).
- The effectiveness of the threat analysis performed to the KAOS requirements model (question 4).
- The effectiveness of the transformation scheme from KAOS to B (question 5).

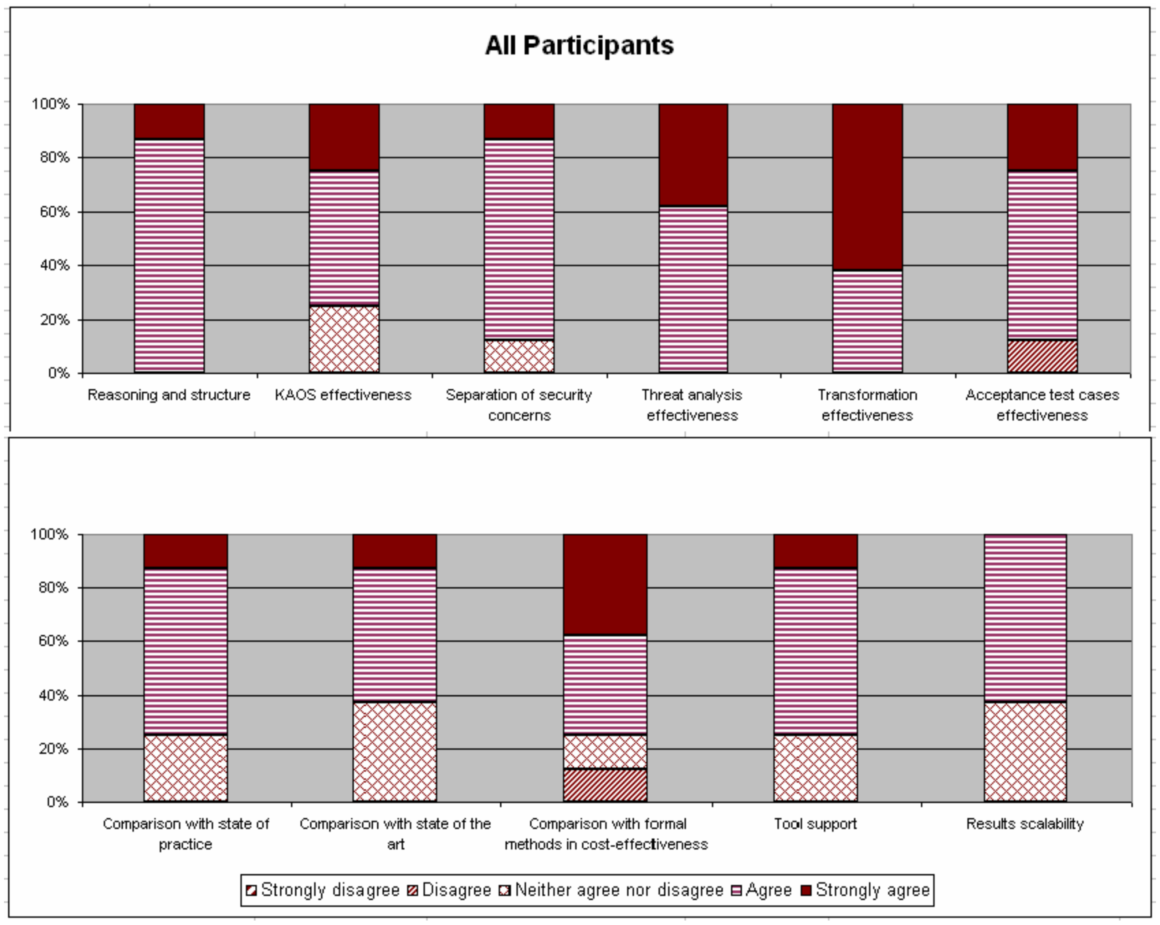


**Figure 48: Results of the Experts' Questionnaire**

For the rest of the questions, the results came as follows:

- For question 2 that inquires about the effectiveness of KAOS in doing the requirements analysis in FADES, 75% of the experts agreed or strongly agreed. 25% answered "Neither agree nor disagree". The 25% represent 2 out of the 8 expert participants and they justified their "Neither agree nor disagree" answer by the fact that they do not have sufficient knowledge and experience about KAOS, and they cannot judge how effective it could be. However, based on what they got to know about KAOS from the presentation given to them at the beginning of the interview, they said they cannot disagree to its effectiveness.
- For question 3 that inquires about the ability of FADES to separate security concerns from the rest of the system requirements, 88% of the experts agreed and strongly agreed. 12% answered "Neither agree nor disagree", which means that only 1 expert out of the 8 participants gave this answer. That expert justified his answer by his concern about the effect of security on other non-functional requirements especially quality of service.

- For question 6 that inquires about the effectiveness of the acceptance test cases in raising the confidence in the resulting implementation, 88% of the experts agreed or strongly agreed. 12% of the experts disagreed and the justification for the disagreement is that FADES employs the B formal method to elaborate design and implementation, which guarantees correctness by construction. Further, a well-trained developer in formal methods would not need testing to verify the correctness of the implementation. However, that expert who disagreed to the acceptance test cases effectiveness indicated that the test cases might be more useful for the customer to verify compliance between implementation and requirements.
- For question 7a that compares FADES to the current state of practice in security engineering, 75% agreed or strongly agreed that overall, FADES is capable of effectively engineering security concerns and produce better software security compared to the current state of practice in security engineering. 25% (2 experts) of the experts answered "Neither agree nor disagree". One of the two experts indicated his lack of knowledge of the current state of practice while the other indicated that he wants to experience FADES himself in order to formulate judgment about the approach and compare it to the current state of practice.
- For question 7b that compares FADES with the current state of the art in security engineering research, 63% either agreed or strongly agreed. 37% (3 experts) of the experts answered "Neither agree nor disagree". The justification of this answer by one of the three experts is that he already knows of two ongoing security engineering research projects that would neither be worse nor better than FADES, but almost the same. That expert indicated that he cannot favor FADES over the current state of the art in security engineering given the fact that the two ongoing projects are expected to give similar results to FADES. The other two experts indicated that their field of specialization is formal methods and not security engineering and they do not have sufficient knowledge about the current state of the art in security engineering research.
- For question 8 concerned with the cost-effectiveness of FADES compared to other formal methods, 75% either agreed or strongly agreed. 13% (1 expert) answered "Neither agree nor disagree" though that expert believes that KAOS is cost-effective; however, he indicated that the cost-effectiveness of the software product development is not only concerned with the requirements phase. 12% (1 expert) disagreed that FADES is more cost-effective than other formal methods and the expert indicated that he saves his judgment until he sees FADES in an industrial project.
- For question 9 concerned with the FADES tool support, 75% of the experts either agreed or strongly agreed to the strength of the tool support. 25% (2 experts) answered "Neither agree nor disagree" justifying the answer by the fact that the 2 experts did not have sufficient experience with the KAOS tool.
- For question 10 that is concerned with the experts' opinion about the scalability of the practitioners' experiment results based on their experience in other formal methods, 63% agreed

that the results would scale when the approach is applied to large industrial projects. 37% answered "Neither agree nor disagree", but none of them disagreed to the question.

The experts gave some qualitative comments about the scalability of the practitioners' experiment results. Two of the three experts that answered "Neither agree nor disagree" indicated that the scalability of FADES relies on the scalability of both KAOS and B. The two experts indicated that they are more confident in the scalability of KAOS but they are more concerned with the scalability of the B method that proven more suitable to moderate-size projects and it requires "bright" experts in B in order to obtain good results. One of the experts who agreed to the scalability issue indicated that based on his experience in multiple research and industrial projects, results of formal method-based approaches would scale. Another expert who agreed to the scalability issue indicated that his agreement is based on his experience with requirements engineering methods when applied to large-scale projects.

For the open-ended question concerned with the strengths and limitations of FADES, the following strengths have been mentioned:

- The approach is gentle in making the transition between semi-formal requirements and formal design. Also, it keeps formal methods away in requirements analysis allowing more flexibility in reasoning about requirements.
- The availability of sufficient traceability information along with the acceptance test cases that reflect the behavior of the B machines with respect to the KAOS model are signification to CC evaluators. Further, if problems are detected in the B model, they could be solved in the KAOS model with the help of the traceability links between requirements and design.
- The approach builds on top of existing proven approaches namely KAOS and B.
- Strong tool support based on the tool support of KAOS and B.
- FADES does add value to the B approach. It gives more structure to the B method to work with and makes it easier to ensure and meet all the requirements. Also, the B method cannot do it alone and it needs a method like KAOS to bridge the gap between informal requirements and the initial B specifications
- The automatic transformation to design is the key contribution of FADES. Experts indicated that they applied requirements methods to systems like the smart card and oracle security database but they never transformed the requirements models to design. Therefore, they found FADES distinguished from other security engineering approaches in extending requirements to design especially with the mechanical transformation. The uniqueness of FADES is in the automated transformation.
- Bridging the gap between semi-formal requirements and formal design with emphasis on security concerns.
- The breadth of coverage in the software development process from requirements capture down to implementation as opposed to other formal methods that do it from specifications to implementation but not requirements.

- The approach brings specifically security concerns in formal methods which has been a weak research area.
- The usability of the approach is quite good because of its relation with semi-formal methods.

The open-ended question in the experts' questionnaire also inquired about the limitations of FADES. The following limitations have been mentioned:

- FADES inherits one of the limitations of the B method as a formal method to handle availability and quality of service issues
- The approach inherits limitations of applying formal methods like scalability to larger size projects and still requires highly-skilled people
- If the KAOS semantics are weakly specified, that would affect the strength of the B semantics and would make the refinement in B weak
- There is a danger to start going into design in KAOS which makes the B specifications too concrete
- The approach does not eliminate the need for the analyst expertise in the problem domain

The coming section integrates the results obtained in the practitioners' experiment along with the results of the experts' experiment in order to analyze them in relation to each other and in relation with the research questions.

## 5.4 Practitioners and Experts Evaluation Analysis

This section discusses and analyzes the results obtained in the two experiments used to validate FADES (sections 5.2.4 and 5.3.2). The analysis integrates the results obtained from both experiments to show how these results would serve the research hypothesis and provide answers to the research questions. The results of the two experiments come in both quantitative and qualitative formats. Figure 49 depicts the results obtained from both practitioners and experts in response to the common questions between the two experiments. These common questions inquired about the effectiveness of the different features of FADES. The red bars refer to the experts'answers; the purple bars refer to the group of practitioners who applied FADES, and the green bars refer to the group of participants who applied the B method by its own.

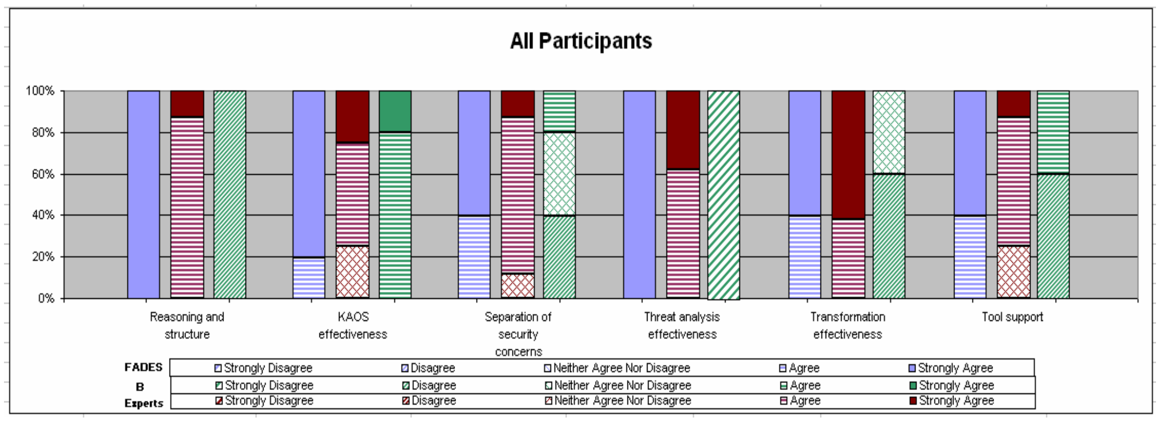


**Figure 49: Experts' Questionnaire Results in Comparison with the Practitioners' Questionnaire Results**

The results of the 6 common questions demonstrate an agreement on the effectiveness of the different features of FADES to handle security requirements. In general, practitioners have been more positive about the approach and they were more concerned with how the features of the approach would enable them to effectively apply it to an industrial project. The practitioners' evaluation to FADES was based on their practical experience with informal and semi-formal requirements engineering methods and software application security. Experts, on the other hand, were more critical to the scientific contribution of FADES and how it could enhance the state of the art in the security engineering domain. Experts were more conservative in their judgment about the different aspects of FADES and how it compares to other formal methods in the security engineering domain. This is justified by the experts' academic background that urges them to be more accurate and precise in their judgment.

The total number of practitioners and experts who participated in the FADES validation studies is 18 (10 practitioners and 8 experts). Both practitioners and experts shared agreement to the effectiveness of FADES to reason about security requirements and structure them in a well-organized format. This result could be justified by the fact that the KAOS element in FADES allows for organizing requirements in the goal graph structure. The refinement of high-level goals into subgoals while rationalizing the decomposition decisions shows the effectiveness of the refinement mechanism of KAOS to reason about requirements. The integrated results of both practitioners and experts concerning the effectiveness of FADES to reason about requirements participate in the validation of the research question of *how feasible and effective the coupling between semi-formal requirements specifications and formal design and implementation is in producing more complete, consistent and secure software.*

Both practitioners and experts were in consensus about the effectiveness of the threat analysis mechanism of KAOS. Both groups believe that performing threat analysis to the requirements model would effectively enhance the security quality of the requirements model through the early detection and resolution of possible security vulnerabilities. The good quality security requirements model would enhance the security quality of the later phases of development, design and implementation, resulting in a better security for the final software product. Further, the early resolution of security vulnerabilities during the development lifecycle is cheaper than resolving such vulnerabilities at later stages of development both from financial and business impact perspectives [1]. The integrated results of both practitioners and experts concerning the effectiveness of the threat analysis mechanism in FADES participate the validation of the following research questions:

- *How feasible and effective is the coupling between semi-formal requirements specifications and formal design an implementation in producing more complete, consistent and secure software?*
- *How effective is FADES in preserving security properties throughout the development?*

Both practitioners and experts agreed or strongly agreed to the effectiveness of the transformation from KAOS to B in bridging the gap between requirements and design. Both groups found the transformation rules sound and indicated that the automation of the transformation facilitates the task and reduces the probability of introducing errors. The experts group in specific referred to the automated transformation as the key contribution of FADES to enhance the state of the art in the science of security engineering. Further, the experts group highlighted the significance of the transformation scheme in enabling FADES to be comprehensive with respect to the coverage of the different stages of software development. This result has the effect of validating the following research questions:

- *How could the gap between semi-formal security specifications and formal security design and implementation be bridged in order to produce better security?*
- *How feasible and effective is the coupling between semi-formal requirements specifications and formal design an implementation in producing more complete, consistent and secure software?*

For the other 3 common questions (KAOS effectiveness, separation of security concerns, and tool support), the practitioners group was in more consensus while a small percentage of the experts group answered “Neither agree nor disagree”. However, none of the experts group disagreed to any of the questions. The justification of this result for each of the three questions is illustrated in the following paragraphs.

For the question concerned with the effectiveness of the KAOS element in FADES, all the practitioners agreed while only 75% of the experts agreed. The 25% of the experts who answered “Neither agree nor disagree” justified their answer by their lack of knowledge and practice of KAOS. Though the practitioners did not have a prior knowledge of KAOS as well, but their experience with it during the experiment got them to work with it and report their agreement on its effectiveness in analyzing security requirements. The results of the KAOS effectiveness question contributes to the validation of the research question of *how feasible and effective the coupling between semi-formal requirements specifications and formal design and implementation is in producing more complete, consistent and secure software?*

For the question concerned with ability of FADES to separate security concerns from the rest of the system requirements while maintaining the relation with the non-security requirements, all the practitioners group agreed. 88% of the experts agreed as well while only 12% (1 expert) answered “Neither agree nor disagree”. That expert has raised the concern of the impact of security on other non-functional requirements, especially quality of service. This concern could be validated in the future work of FADES when it is applied to projects that involve quality of service requirements. The separation of security concerns question results highlights one of FADES strengths as well as showing a limitation of the research to validate the impact of security concerns on the other non-functional requirements.

For the question concerned with the tool support, it could be observed from Figure 49 that the practitioners group found the tools of FADES quite mature and this has been reflected in their interviews. The experts group, on the other hand, had only 75% agreed while 25% answered “Neither agree nor disagree” justified by their lack of experience with the KAOS tool. This result shows that the practitioners who experienced the tools of FADES found them mature and useful. Not all the experts had the same opportunity to personally experience the tools of FADES and that is the reason why some of them were not sure about the strength of the FADES tool support. However, one of the experts who made some research about KAOS indicated that the tool support of KAOS is quite mature and another expert indicated that one of the strengths of FADES is that it builds on already existing and proven methodologies that have their own tool support.

For the rest of the questions that were specific to the practitioners group, more analysis of them is provided in section 5.2.5. For the questions that were unique to the experts group, their analysis is provided in the coming paragraphs.

For the acceptance test cases effectiveness question, 88% of the experts agreed to the ability of this feature in FADES to raise the confidence in the compliance between the resulting implementation and the security requirements. The other 12% (1 expert) who disagreed justified this by the fact that the B formal methods incorporated in FADES guarantees correctness by construction with no need to acceptance testing. However, his assumption is true under the condition that the developer working with B is an expert in the language and could deal with the mathematical aspects of it. The counter argument to this assumption is that formal methods are not commonly used among software engineering practitioners who might not be experts in B when applying FADES. This raises the need for acceptance testing both for developers and for customers to verify that what they get in the resulting implementation achieves what they desired in the requirements. Further, acceptance testing is also meant to detect any errors that might result from the transformation from KAOS to B and propagate to design and implementation. The results obtained in the acceptance test cases effectiveness question contributes to the validation of the research questions of:

- *How could the gap between semi-formal security specifications and formal security design and implementation be bridged in order to produce better security?*
- *How effective is FADES in preserving security properties throughout the development?*

For the comparison question between FADES and the state of practice in security engineering, 75% of the experts indicated that FADES is better than the state of the practice and highlighted that the state of practice is a mess and lacks a procedural method for handling security requirements. This makes FADES better in being able to provide a step-wise procedure that comprehensively covers the different stages of development while preserving security properties. The other 25% of the experts who answered “Neither agree nor disagree" are either not quite knowledgeable of the state of practice or needs to experience FADES in actual practice to be able to judge it in comparison with other methods. This question contributes to the validation of the research question of *how the integration of semi-formal and formal methods in FADES compares to purely applying formal methods from the completeness, consistency and security quality perspectives.*

For the comparison question between FADES and the state of the art in security engineering research, 63% of the experts agreed that FADES is better than the other existing methods in research. Some of the experts indicated that the state of the art in applying formal methods to security is very poor and they think that FADES would enrich this area. 37% of the experts answered "Neither agree nor disagree" either because of their lack of knowledge about the state of the art or because of their knowledge of other ongoing research work in the security engineering area that would be as good as FADES, but still has not reported any results. The fact that none of the experts disagreed to the question indicates that FADES is an enhancement to the state of the art in security engineering and it is either as good as some of the very few existing methods or even better as some experts depicted. The question contributes to the validation of the research question of *how the integration of semi-formal and formal methods in FADES compares to purely applying formal methods from the completeness, consistency and security quality perspectives.*

For the comparison question between FADES and other formal methods from the cost-effectiveness perspective, 75% of the experts agreed that FADES is more cost effective. They justified their answer by the fact that employing semi-formality during requirements analysis reduces the time and effort of that phase and this has been verified in the practitioners' experiment in which the FADES group managed to develop the requirements model in less time than the B group. Further, the experts highlighted the fact that building a good quality requirements model would facilitate the mission of the other development stages since the requirements would be complete, consistent and clear. 13% of the experts answered "Neither agree nor disagree" justified by the fact that cost-effectiveness of software development is not only concerned with the requirements analysis phase though he believes in the cost-effectiveness of the KAOS element in FADES. A counter argument would be that even if the cost-effectiveness of software development is based on multiple factors, saving some cost of the requirements analysis phase compared to the development in other formal methods would have the effect of enhancing the total cost-effectiveness even with small amounts. 12% of the experts disagreed to the question justified by the fact that the cost-effectiveness of FADES needs to be more fairly judged in actual industrial projects. This question contributes to the validation of the research question of *how much the cost of development is affected in FADES as opposed to the cost in other formal methods.* Though the cost question need to be validated on a larger industrial base of projects, the preliminary results obtained in the practitioners' experiment concerning the cost factor shows a good potential of the approach to be cost-effective. Further, the quantitative study provided by Lamsweerde and his team about the cost-effectiveness of KAOS (section 3.1) emphasizes the cost-effectiveness potential of FADES.

For the question that inquires about the expectation of the expert about the scalability of the results obtained in the practitioners' experiment, 63% of the experts agreed to the question. Further, they indicated that based on their research experience, they expect the results to scale when FADES is applied to larger industrial projects. The other 37% of the experts answered "Neither agree nor disagree" and none of the experts disagreed to the scalability aspect of FADES. Even the 37% whose answers were "Neither agree nor disagree" were more concerned with the inherited scalability limitation of the B element in FADES while they expect the KAOS element to scale well. This concern of the B scalability highlights a future research direction to apply FADES to industrial projects of different sizes in order to specify the range of project sizes for which the application of the approach would be effective and produce good results.

For the open-ended question inquiring about the strengths and limitations of FADES, the experts were positive about the approach. They highlighted a list of strengths and limitations mentioned in section 5.3.2. The main objective of this question has been to validate the research question of *what the strengths and limitations of FADES are.* The identification of the possible strengths and limitations provides directions for the refinement of the approach to overcome its limitations and helps differentiate the approach with its strengths when compared to other similar approaches.

The above mentioned analysis of the evaluation of practitioners and experts demonstrate a good potential of FADES to effectively handle security requirement. This allows the approach to enhance the state of the art in the security engineering area with an approach that integrates semi-formal goal-oriented methods with formal methods to enhance the quality of the requirements model that should subsequently enhance the overall security of software. The results were aligned together and with the research hypothesis that states that FADES is capable of deriving design specifications from security requirements that should provably preserve the requisite security properties while maintaining consistency and completeness at less cost.

## 5.5 Results Scalability Limitations

FADES has been validated with two empirical studies that involved ten practitioners working in a large enterprise and 8 academic and industry experts. The practitioners have applied FADES in comparison to the B formal method by its own. Therefore, some limitations to this validation mechanism are that the approach has been applied on a piece of a software system and the number of participants is small. The limitation of applying the approach to a piece of the electronic smart card system rather than the whole system is overcome by the fact that FADES has been applied to the whole smart card case study prior to the experiment as indicated in section 4.2. The application of FADES to the electronic smart card system highlighted the effectiveness of FADES when compared to the Z formal methods in specifying and designing the system. For example, the threat analysis mechanism performed to the requirements model in FADES detected an integrity threat to the messaged being communicated in the system and this threat was resolved by introducing digital signatures in the requirements model. This threat was not captured in the Z specifications, which led to the propagation of the integrity vulnerability to later stages of development and also to the final product. The coming paragraphs discuss how the results of the practitioners' empirical study are expected to scale.

FADES relies in its core on incorporating formal methods in the development of software security. Formal methods have long been applied in different domains of research in computer science including security engineering [4, 11, 12, 13, 21]. Further, formal methods have been successfully employed in the development of software security in large-scale industrial projects [6, 8, 10]. The results obtained from applying formal methods in the domain of software security at the application layer based on Wing layers of security [9] support our claim that nothing prevents the results obtained in the empirical study that validates FADES from being scalable.

FADES is based on three underlying methodologies namely the KAOS framework for requirements analysis, the transformation scheme that transforms the KAOS requirements model to B and the B formal method for design and implementation. If each of the underlying methods could scale when applied to larger industrial projects, then it is highly expected that the whole FADES approach would scale. The KAOS framework has been applied in a more than thirty-five industrial projects and reported reduction in the project cost of 30% on average with only 10% of the cost reduction in the requirements analysis phase while the other 20% is in the later phases that are based on a well-analyzed and well-specified requirements model [51]. The B method rarely needs to make assumptions about the size of a system being modeled, e.g. the number of nodes in a network [7]. This is strongly supported by the results obtained from applying the B method on research and industrial projects. However, two experts that participated in the validation of FADES indicated that B is more appropriate to medium-size projects and might have a scalability limitation with large size projects. This means that FADES has to be applied to different project sizes in order to identify the scalability range of the project size that would produce good results. This is a limitation of this research that needs to be overcome in the future research work of FADES. The transformation scheme from KAOS to B is generic and comprehensive since it maps every construct in KAOS to an equivalent one in B without being specific to a certain business domain. The case studies in Chapter 4 that demonstrated FADES support the claim that the transformation scheme is not specific to a certain domain. Further, the automation of the transformation from KAOS to B increases the confidence in the comprehension and completeness of the transformation. Therefore, it could be concluded that nothing is expected to prevent the validation results of FADES to scale and generalize.

Four approaches other than FADES employed the goal-oriented KAOS framework for requirements analysis before stepping into design. These are the case study of a real world industrial application for a biological database [94], the approach by Brandozzi and Perry that transforms KAOS specifications into an Architecture Prescription Language (APL) [64], Van Lamsweerde approach that derives architectural design from KAOS specifications [95], and the approach by Nekagawa et. al. that transforms KAOS to VDM++ for later stages of development [4]. The four approaches that extended KAOS are described in more details in Chapter 1. The approaches that extended KAOS for further stages in development like architecture or design have shown good results at later stages of development due to employing goal-orientation during requirements analysis. More specifically, the results obtained from the case study on the biological database system [94] and the transformation from KAOS to APL [64] reported strong and scalable results on employing KAOS in their approaches for requirements analysis. Therefore, the results obtained from similar approaches that incorporated goal orientation support the generalization of the results obtained from demonstrating and validating FADES that show the same pattern of effectiveness and usefulness.

The argument provided in the following paragraphs is basically concerned with highlighting the similarities between FADES and the other methods that extended KAOS that led to obtaining similar results. The discussion also explores the unique characteristics of FADES and argues for their own merits. The chief objective of outlining the similarities and differences between FADES and the other approaches that extended KAOS is to be able to generalize FADES and estimate the scalability of the obtained results when the approach is applied to large scale industrial projects. This argument goes in the direction of justifying the limitations of this research and being able to generalize the results and scale them.

Like other methods that extended KAOS, the main objective of FADES is to produce better design and implementation at reasonable cost through performing deeper analysis of requirements resulting in a more thorough and well-analyzed requirements model. Therefore, one of the major contributions of FADES is in bridging the gap between requirements and the later phases of design and implementation. Like most other approaches that extended KAOS, FADES extends KAOS to a formal language providing stronger guarantees of correctness especially for crucial aspects like security ones. Further, the ability of KAOS to formulate requirements in a graph structure makes it more eligible for transformation to formal languages. This means that introducing KAOS during requirements analysis allows for relaxing the rigid formality of formal languages that are more suitable to later phases of development. This gives a better opportunity for formal languages to be applied in their appropriate situations while avoiding their complexity and cost at the requirements analysis phase that favors reasoning over precise specification. This was emphasized in the interviews with the participants in the empirical study who indicated that formal methods constrain the way they think while a semi-formal method with a visual notation like KAOS provides sufficient flexibility to reason about requirements.

FADES automates the transformation from KAOS to the B formal language allowing for more convenient application and less error-proneness in applying the transformation rules. FADES is comparable to some of the other methods that extended KAOS in this respect. For example, the generator that transforms KAOS to VDM++ developed in [4] automates the transformation.

Like the other methods that extended KAOS, FADES is requirements-driven meaning that all requirements are considered very early in development and reasoned about in order to drive later stages of development. KAOS as a goal-oriented language allows for modeling constraints of both the software-to-be and its environment. These constraints are propagated to the later stages of development and have to be satisfied before making any design decision. All the approaches that applied goal-orientation before stepping into architecture and design have made requirements the driving force for development so that the resulting product satisfies the specified requirements.

FADES has been demonstrated on some industry-related case studies of reasonable size showing the potential strength of the approach in producing implementation for security requirements in a provable and constructive way. Further, the approach has been validated with an empirical study that focused on the individual experience of the participants in using the approach. The results obtained from both the case studies and the empirical study has been similar to the ones obtained from the other methods that extended KAOS. The results in this research show strong potential of applying goal-orientation prior to stepping into formal design when compared to early application of formal methods in requirements analysis. The other methods that extended KAOS reported that the employment of KAOS for requirements analysis enhanced their requirements models from the completeness, consistency and correctness perspectives. For example, the employment of KAOS for the design of the biological database in [94] showed that goal-orientation was capable of including the original schemas build with the traditional methods and further suggesting additional alternatives that lead to more comprehensive design. Incorporating goal-orientation in the database design provided support for systematic evaluation of design alternatives and generated enhanced schemas with rich and explicit data semantics [94]. Although, the results obtained for the biological database design are preliminary since they were obtained from applying the goal-orientation approach to some case studies, they match with the results obtained in this research in that incorporating goal-orientation in the development of security concerns enhanced the quality of the resulting requirements models compared to traditional formal methods. The experience of the study participants show that goal-orientation has enabled them to reason more effectively about requirements, identify and resolve potential conflicts and security threats, and build more complete and consistent requirements models compared to their current semi-formal practice and other formal methods applied by some participants in the study.

FADES is different from other methods that extended KAOS in that it is employs the KAOS security extension and becomes specific for security-specific elements of software. Up to our knowledge, FADES is the first approach to security engineering that treats security requirements as first-class citizens and considers them early in development that is driven by the properties specified in requirements. Further, FADES is the first approach to apply goal-orientation to security requirements with the objective of producing an implementation for these requirements while maintaining the security properties throughout the development lifecycle. This means that FADES enhances science in the security engineering domain by participating in the solution of the poor security engineering problem.

The other difference between FADES and the other methods that extended KAOS is that it propagated the paradigm shift from traditional requirements analysis methods to goal-orientation to the security domain. FADES has added to the body of knowledge of security engineering by doing a paradigm shift in that domain to goal-oriented security engineering that make use of all the goal-orientation characteristics. FADES has demonstrated the feasibility of employing goal-orientation for engineering security requirements.

FADES is the first among the rest of the methods that extended KAOS to analyze the KAOS goal graph not only to transform it to a formal language, but to generate operational scenarios in the formal of acceptance test cases as well. The derived acceptance test cases are used to ensure that the derived implementation achieves the requirements elaborated in the KAOS model. Further, the acceptance test cases result could be used by as exit criteria by the customer to accept the product from the development team after verifying that the resulting product achieves what the customer has requested.

The validation mechanism used for FADES in the form of two empirical studies result in a quantitative analysis of the gain of employing goal-orientation prior to design when compared to other formal methods. Further, the two studies collected qualitative results from security engineering practitioners, academic and industry experts about FADES. Though the practitioners' study scale is not statistically large, it focuses on the individual experiences of the participants in order to get more insight about the potential of FADES. The assessment of the experts supported the argument about the scalability of the results obtained in the practitioners' study. 63% of the experts confidently expect the results to scale and none of the experts disagreed to their scalability.

The goal-oriented paradigm, and specifically KAOS, has been introduced in research about ten or more years ago; nevertheless, as Van Lamsweerde reported that there is no quantitative comparative analysis that compares the KAOS framework with other formal methods for requirements engineering. Therefore, the results of the empirical studies validating FADES enrich the body of knowledge in the goal-oriented security engineering domain.

# Chapter 6: Conclusion

With the ever-increasing threats to software security, formal approaches inevitably play a crucial role in developing software with the requisite security. With formality and the affiliated rigor comes additional complexity and constraints to an already difficult process of engineering security into software. How can a reasonable balance between informal reasoning and formal representation of the specifications be achieved? As the process proceeds through design and implementation, how could security properties be preserved and guaranteed in a cost-effective way? This dissertation presents FADES as a requirements-driven software security engineering approach integrating semi-formal requirements methods with formal design methods to produce assured software security in a cost-effective manner. FADES is the first to bridge the gap between goal-oriented requirements and formal design for security-specific elements of software. FADES integrates the semi-formal goal-oriented KAOS method with the B formal method in order to organize requirements in such a way that reasonable completeness and consistency can be achieved for formal design.

The research effort has focused on answering a set of questions:

- How could the gaps between semi-formal security specifications, formal security design and implementation be bridged to produce better security?
- How feasible and effective is the coupling between semi-formal requirements specifications and formal design in producing more complete, consistent and secure software?
- How effective is FADES in preserving security properties throughout the development?
- How does the integration of semi-formal and formal methods in FADES compare to purely applying formal methods from the completeness, consistency and quality perspectives?
- How much is the cost of development affected in FADES as opposed to the cost in other formal methods?
- What are the strengths and limitations of FADES?

The first question has been answered by developing the transformation scheme and the acceptance test cases described in Chapter 3 to bridge the gap between the security specifications and their realization in design and implementation while ensuring compliance between requirements and implementation. The rest of the above research questions have been explored by walking through two paths. The first path is to apply FADES to some industry-related case studies to demonstrate the feasibility and effectiveness of the approach in building secure software in a provable way. The case studies have also navigated the strengths and limitations of the approach. The second path is to qualitatively and quantitatively assess the observed strengths of FADES when compared to relevant formal security engineering approaches with the support of the experience of some academic and industry experts in the domain of formal security. The approach has been compared to other formal methods through an empirical study that involved software engineering practitioners to evaluate and compare FADES to other formal methods. Further, the assessment of some academic and industry experts has been collected to validate the effectiveness of the approach and identify its strengths and limitations.

## 6.1 Main Contributions

FADES has contributed to the Computer Science body of knowledge, and in particular to that of Security Engineering. The approach's contributions are manifested through providing the security software engineering community with a systematic, provable and cost-effective approach to engineer security-specific elements of software in such a way that security properties are preserved throughout the development. These contributions originate from:

- The FADES transformation scheme that bridges the gap between requirements and design.
- The acceptance test cases that ensure compliance between security requirements and the produced implementation raising the confidence in producing what the stakeholders desire.

FADES establishes a reasonable bridge between security requirements and security design while preserving security properties and verifying compliance between requirements and implementation through formal specifications and controlled transformation. FADES provides a set of transformation rules that are comprehensive in mapping every construct in KAOS to an equivalent one in B while not being specific to a given business domain. The transformation from KAOS to B is considered the main contribution of FADES as indicated by most of the academic and industry experts who participated in FADES validation when asked about the strengths of the approach. The transformation allowed for connecting the two worlds of KAOS and B that are already proven technologies and have mature tool support. The integration of KAOS and B enables FADES to be sufficiently rigorous yet flexible enough to produce complete, consistent and clear security requirements model.

Again, the other main contribution of FADES is the automatic derivation of acceptance test cases from the requirements model has the effect of raising the confidence in the compliance between implementation and requirements. This has been validated in the domain experts evaluation of FADES that indicated their belief in the effectiveness of the derived acceptance test cases.

## 6.2 Strengths and Limitations

The application of FADES to some case studies and the empirical validation have highlighted some of the strengths and limitations of the approach. FADES is able to:

- Increase confidence in engineering security applications at the Common Criteria EAL 5, 6, and 7.
- Preserve security properties during development through correctness by construction.
- Provide a comprehensive approach for engineering security across the development life cycle.

- Separate security concerns from the rest of the functional requirements.
- Build a reasonably complete and consistent requirements model.
- Detect and resolve security vulnerabilities early in development during requirements analysis.
- Enforce security constraints of requirements in design and implementation.
- Provide means to automatically transform requirements to an equivalent initial format of design.
- Verify the compliance of the derived implementation to requirements.
- Maintain sufficient traceability information to facilitate future changes.
- Provide strong tool support for wider applicability and practicality.

The quantitative comparison of FADES to other formal methods concluded that for the situations considered, FADES produces a reasonably complete and consistent requirements model with less effort and more convenience. The quantitative comparison has only provided some preliminary results of the FADES capabilities, but it has prepared the ground for a validation framework that could be used when FADES is demonstrated on larger industrial projects. Further, the evaluation of the domain experts has strengthened the results obtained from the quantitative comparison and raised the confidence in the scalability of these results.

The case studies and the experiments hold to quantitatively validate the approach have also revealed some limitations of the approach. First, FADES is primarily based on formal methods for development that come with extra cost due to their complication and difficulty of learning over informal approaches. The tradeoff appears to be in favor of FADES. Note that the extra costs of rigor are applied appropriately after the goal-directed requirements activity is complete. Second, FADES inherits KAOS limitation to provide a formal evidence of requirements completeness and consistency that might affect the derived design quality. There are some recommendations to overcome FADES limitations:

- Apply rigor only to security aspects of software where the extra cost of rigor is justified by allocating the cost to the critical aspects of software.
- Conduct more feedback sessions with the stakeholders. The more feedback sessions the analyst holds with the stakeholders to get their answers on open questions or to verify the completeness and consistency of the requirements model, the better the results obtained at the design and implementation phases [3].

## 6.3 Future Work

FADES needs to be applied in real industrial projects that are large enough to reveal more characteristics to validate the approach further. Gaining experience with FADES on projects of different sizes would reveal areas and configurations where applying FADES obtains the best results.

The Spy Network case study described in section 4.1 demonstrated FADES efficacy in providing sufficient traceability information for performing accurate impact analysis when changes to security specifications are introduced. This research examined corrective changes to security, but there re still three more areas of software change that can be examined more thoroughly -- adaptive (changes in the software environment), perfective (new user requirements), and preventive (prevent future problems).

Van Lamsweerde introduced patterns for specifying security goals in the KAOS security extension [1]. These patterns are used in the preliminary elicitation of security goals driven by application-specific instantiations of generic specification patterns. The security patterns provided by Van Lamsweerde include the common security requirements namely confidentiality, integrity, availability, privacy, authentication and non-repudiation. We need to investigate the ability to build generic test suites for the security patterns that could be instantiated with domain specific knowledge. This future research direction will provide well-structured verification mechanism for testing both the requirements model itself and the compliance of the other development artifacts such as design and implementation with respect to the requirements model. Further, the idea of generic test suites with security patterns brings with it the benefits of reuse that can substantially reduce development costs (time, and money), and produce less error-prone software.

One of the KAOS shortcomings is the lack of formal evidence of requirements completeness and consistency. Future work of testing the requirements model during requirements analysis will allow for the detection of requirements defects such as incomplete or inconsistent requirements. Early detection of requirements defects will save time and effort in resolving these defects before they propagate to later development phases. We suggest investigating the Test Automation Framework (TAF) design by Blackburn [28]. The TAF framework represents the requirements model into a memory structure from which requirements constraints can be related to detect problems. The TAF then generates reports of defects along with test vectors that could be used in unit testing and integration testing of the implementation. Integrating TAF with FADES enhances the ability of FADES to build more complete and consistent requirements model and propagate these qualities to design and implementation allowing for producing better security quality.

FADES has been designed for security-specific elements of software while it can be generalized to other non-functional requirements that are critical like privacy, reliability and safety. More investigation is needed to identify how to model these requirements in KAOS and augment the transformation scheme from KAOS to B with more rules if necessary to handle these requirements.

FADES has been designed to secure software at the application level. However, its applicability could be extended to design and verify new network security protocols. In the case of problems like network security protocols, requirements elicitation is less of a problem than verifying the strength of the protocol design. To show applicability, FADES needs to be demonstrated for feasibility using some case studies.

# References

[1] A. van Lamsweerde, “Elaborating Security Requirements by Construction of Intentional Anti-Models”, *Proc. ICSE’04: 26th Intl. Conf. on Software Engineering*, May 2004.

[2] Wilander, J. & Gustavsson, J., “Security requirements – A field study of current practice”, *Symposium on Requirement Engineering for Information Security (SREIS’ 2005)*, 2005.

[3] A. van Lamsweerde and E. Letier, “Handling Obstacles in Goal-Oriented Requirements Engineering”, *IEEE Transactions on Software Engineering*, Special Issue on Exception Handling, Vol. 26 No. 10, Oct. 2000, 978-1005.

[4] Nakagawa, H., Taguchi,K., Honiden, S, “Formal Specification Generator for KAOS”, *Proceedings of The International Conference on Automated Software Engineering (ASE’07)*, Nov. 2007.

[5] Bowen, J., and Hinchey, M., *Application of Formal Methods*, Prentice Hall, 1995

[6] Stepney, Susan, Cooper, et. al., “An Electronic Purse Specification, Refinement, and Proof”, *Programming Research Group, Oxford University Computing Labarotory*, July 2000.

[7] Abrial, J.-R., *The B Book: Assigning Programs to Meanings*, Prentice-Hall.

[8] Clarke, Edmund, Wing, J., et. al., “Formal Methods: State of the Art and Future Directions”, *ACM Computing Surveys,* Vol. 28, No. 4, 1996.

[9] Wing, Jeannette, “A Symbiotic Relationship Between Formal Methods and Security”, *Proceedings of Computer Security, Dependability and Assurance: From Needs to solutions,* pg. 26-38, 1998.

[10] Sabatier, D., Pierre, L., “The Use of the B Formal Method for the Design and the Validation of the Transaction Mechanism for Smart Card Applications”, *In the 17th Proceedings of Formal Methods in System Design,* pg. 245-272, 2000.

[11] J. Mylopoulos, M. Kolp, and J. Castro, “UML for agent-oriented software development: The Tropos proposal”, *Proc. of the 4th Int. Conf. on the Unified Modeling Language UML’01*, Oct. 2001.

[12] R. G. Dromey, “From requirements to design: Formalizing the key steps”, *SEFM 2003 . International Conference on Software Engineering and Formal Methods*. The University of Queensland School of Information Technology and Electrical Engineering, pp. 2.13, 2003.

[13] M.G. Hinchey, J.L. Rash, C.A. Rouff, “Requirements to Design to Code: Towards a Fully Formal Approach to Automatic Code Generation”, NASA Technical Report, January 2005.

[14] P. Giorgini, F. Massacci, J. Mylopoulous, N. Zannone, “Requirements engineering meets trust management: model, methodology, and reasoning”, *Proc. of iTrust-04, LNCS 2995*, pp. 176– 190, 2004.

[15] McDermott J, Fox C (1999) Using abuse case models for security requirements analysis. In: *Proceedings of the 15th annual computer security applications conference (ACSAC’99)*, Phoenix, Arizona

[16] G. Sindre and A.L. Opdahl, “Eliciting Security Requirements by Misuse Cases, *Proc. TOOLS Pacific’2000 - Techn. of Object-Oriented Languages and Systems*, 120-131.

[17] Firesmith, Donald G., “Security Use Cases”, *Journal of Object Technology (JOT),* issue 5, 2003. http://www.jot.fm/issues/issue_2003_05/column6

[18] L. Liu, E. Yu and J. Mylopoulos, “Security and Privacy Requirements Analysis within a Social Setting”, *Proc. RE’03– Intl. Conf. Requirements Engineering,* Sept. 2003, 151-161.

[19] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, “Nonfunctional requirements in software engineering.”, Kluwer Academic, Boston, 2000.

[20] Liu, L. and Yu, E. “From Requirements to Architectural Design - Using Goals and Scenarios.”, *ICSE- 2001 Workshop: From Software Requirements to Architectures (STRAW 2001)* May 2001, Toronto, Canada. pp. 22-30.

[21] Hoss, Allyson M. and Carver, Doris L., “Ontological Approach to Improving Design Quality”, *in Proceeding of IEEE Aerospace Conference,* pg.12 pp., 2006.

[22] J. Liu, Z. Liu, J. He, and X. Li, “Linking UML models of design and requirement”, *Pro. ASWEC’2004*, Melbourne, Australia, 2004. IEEE Computer Society. 82

[23] X. Li, Z. Liu, and J. He, “Formal and use-case driven requirement analysis in UML”, *COMPSAC01*, pages 215–224, Illinois, USA, October 2001. IEEE Computer Society.

[24] Z. Liu, J. He, X. Li, and Y. Chen, “A relational model for formal object-oriented requirement analysis in UML”, In J. Dongand J. Woodcock, editors, *Proc. ICFEM03, Lecture Notes in Computer Science 2885*. Springer, 2003.

[25] Hung Ledang and Jeanine Souquieres. Formalizing UML behavioral diagrams with B. *In Tenth OOPSLA Workshop on Behavioral Semantics : Back to Basics, Tampa Bay*, Florida, USA, Oct 2001.

[26] Hung Ledang and Jeanine Souquieres. MODELING CLASS OPERATIONS IN B: application to UML behavioral diagrams. In IEEE Computer Society, editor, *16th IEEE International Conference on Automated Software Engineering - ASE’2001*, Loews Coronado Bay, San Diego, USA, Nov 2001.

[27] Hung Ledang and Jeanine Souquieres. Integration of UML Views Using B Notation. *In proceedings of WITUML02*, Spain, 2002.

[28] Blackburn, M., Busser, R., and Nauman, A., “Removing Requirements Defects and Automating Test”, ”, *Software Productivity Consortium,* 2001.

[29] Busser, R., Blackburn, M., and Nauman, A., “Automated Model Analysis and Test Generation for Flight Guidance Mode Logic”, *The 20th Conference on Digital Avionics Systems*, Vol. 2, pg. 9B3/1-9B3/9, 2001.

[30] Blackburn, M., Busser, R., and Nauman, A., “Why Model-Based Test Automation is Different and What You Should Know to Get Started”, *Software Productivity Consortium,* 2004.

[31] Busser, R., et. al., “Reducing Cost of High Integrity Systems through Model-Based Testing”, *23rd Conference on Digital Avionics Systems,* Vol. 2, pg. 6.B.1-61-13, 2004.

[32] Landtsheeer, R., Lamsweerde, A., “Reasoning about Confidentiality at Requirements Engineering Time”, *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* pg. 41-49, 2005.

[33] Lamsweerde, A., “Building Formal Requirements Models for Reliable Software”, *Proceedings of the 6th Ade-Europe International Conference Leuven on Reliable Software Technologies,* pg. 1-20, 2001.

[34] Lamsweerde, A., et. al., “Leaving Inconsistency”, *Proceedings of the ICSE’97 Workshop on “Living with Inconsistency”,* 1997.

[35] Ponsard, C., et. al., “Early Verification and Validation of Mission Critical Systems”, *Proceedings of the 9th International Workshop on Formal Methods for Industrial Critical systems (FMICS),* 2004.

[36] Lamsweerde, A., “Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice”, *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE’04),* 2004.

[37] Letier, E., and Lamsweerde, A., “Deriving Operational Software Specifications from System Goals”, *ACM SIGSOFT Software Engineering Notes,* Vol. 27, Issue 6, pg. 119-128, 2002.

[38] Letier, E., and Lamsweerde, A., “Agent-Based Tactics for Goal-Oriented Requirements Elaboration”, *Proceedings of the 24th International Conference on Software Engineering,* pg. 83-93, 2002.

[39] Letier, E., and Lamsweerde, A., “Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering”, *ACM SIGSOFT Software Engineering Notes,* Vol. 29, Issue 6, pg. 53-62, 2004.

[40] Fontaine, P.J., *Goal-Oriented Elaboration of Security Requirements*, M.S. Thesis, Dept. Computing Science, University of Louvain, June 2001.

[41] Darimont, R., and van Lamsweerde, A., ªFormal Refinement Patterns for Goal-Driven Requirements Elaboration,º *Proc. Fourth ACM SIGSOFT Symp. Foundations of Software Eng*., pp. 179-190, Oct. 1996.

[42] van Lamsweerde A., Darimont, R., and Massonet, P., "Goal- Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned",º *Proc. Second Int'l Symp. Requirements Eng. (RE '95)*, 1995.

[43] Schneider, Steve, *the b-method,* PALGRAVE, 2001.

[44] Cansell, Dominique and M´ery, Dominique, "Foundations of the B Method", *Proceedings of Computing and Informatics*, vol. 22, pg. 1-31, 2003.

[45] Sekerinski, Emil, et. al., "Program Development by Refinement: Case Studies Using the B Method", Springer, 1999.

[46] Bohner, S., Gracanin, D., and Hassan, R., "Tolerating Change in a Secure Environment: A Visual Perspective," *2007 Cyber Security and Information Infrastructure Research Workshop (CSIIRW-07)*, Oak Ridge, TN, pg. 239-249, May, 14-15, 2007.

[47] Hassan, Riham, et. al., "Goal-Oriented, B-Based, Formal Derivation of Security Design Specifications from Security Requirements", *Symposium on Requirements Engineering for Information Security,* Barcelona, Spain, 2008.

[48] Hassan, Riham, Bohner, S., ElKassas, S., "Formal Derivation of Security Design Specifications from Security Requirements", *2008 Cyber Security and Information Infrastructure Research Workshop (CSIIRW-08)*, Oak Ridge, TN, May, 12-14, 2008.

[49] Hassan, Riham, et. al., "Integrating Formal Analysis and Design to Preserve Security Properties", *Proceedings of the HAWAII International Conference on System Sciences,* Hawaii, USA, 2009.

[50] Cimitile, A.; Fasolino, A.R.; Visaggio, G. "A Software Model for Impact Analysis: A Validation Experiment", *Proceedings of the Sixth Working Conference on Reverse Engineering,* Oct. 1999, pg. 212-222.

[51] www.objectiver.com

[52] http://www.atelierb.eu/index_en.html

[53] http://www.b-core.com/

[54] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/depthSearch.htm

[55] http://www.cs.auckland.ac.nz/~ute/220ft/graphalg/node10.html#SECTION00022000000000000000

[56] Clarke, L., Podgurski, A., and Richardson, D., "A Formal Evaluation of Data Flow Path Selection Criteria', *IEEE Transactions on Software Engineering,* Vol. 15, No. 11, Nov. 1989.

[57] E. Letier, "Reasoning About Agents in Goal-Oriented Requirements Engineering", PhD Thesis, Department d'Ingenierie Informatique, Universite Catholique de Louvain, Louvain-la-Neuve, Belgium, 2001.

[58] E. J. Amoroso, "Fundamentals of Computer Security". Prentice-Hall, 1994.

[59] R.A. Kemmerer, "Cybersecurity", Invited Mini-Tutorial, *Proc. ICSE'03: 25th Intl. Conf. on Software Engineering*, Portland, IEEE CS Press, May 2003, 705-715.

[60] CERT, http://www.cert.org/stats/cert_stats.html.

[61] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way,* Addison-Wesley, 2001.

[62] Crook R et al (2002) Security requirements engineering: when anti-requirements hit the fan. In: *Proceedings of the 10th anniversary IEEE international requirements engineering conference RE'02)*, Essen, Germany

[63] Beeck, Michael V., Margaria, Tiziana, Steffen, Bernhard, A Formal Requirements Engineering Method for Specification, Synthesis, and Verification, in *Proceedings of the 8th conference on Software Engineering Environments*, pg. 131-144, 1997.

[64] BRANDOZZI, M. AND D. E. PERRY, “Transforming Goal Oriented Requirement Specifications into Architectural Prescriptions”, *First International Workshop from Software Requirements to Architectures (STRAW'01)*, Toronto, Canada.

[65] Briand, Lionel C., et. al, “Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems”, Fraunhofer Institue for Experimental Software Engineering, 1999.

[66] Ramesh, Balasubramaniam, et. al., “Requirements Traceability: Theory and Practice”, *Annals of Software Engineering*, *3*, p.397-415, 1997.

[67] Lehman M. M. “On understanding Laws, evolution and conversation in the large program lifecycle.”, *Journal of Software & Systems,* vol. 1, pp. 213 - 221, 1980.

[68] Spafford, E., and CRA Board of Directors, “Exploring Grand Challenges in Trustworthy Computing”, Computing Research Association, 2004.

[69] Landtsheer, R., and Van Lamsweerde, A., “Reasoning about Confidentiality at Requirements Engineering Time”, *Proceedings of the 10th European software Engineering Conference,* pg. 41-49, 2005.

[70] Fran¸cois, J., and Ponsard, C., “Deriving Acceptance Tests from Goal Requirements”, CETIC Research Center, Charleroi (Belgium), 2005.

[71] Clarke, L., et. al., “A Formal Evaluation of Data Flow Path Selection Criteria”, *IEEE Transactions on Software Engineering,* Vol. 15, No., 11, 1989.

[72] Feather, M., et. al., “Requirements and Specification Examplars”, *Proceedings of the Automated Software Engineering,* Vol. 4, No. 4, 1997.

[73] Butler, M., et. al., “Rigorous Open Development Environment for Complex Systems”, *Proceedings of the First International Workshop,* 2006.

[74] Smith, S., and Spafford, E., “Grand Challenges in Information Security: Process and Output”, *IEEE Security and Privacy Magazine,* pg. 69-71, Jan/Feb, 2004.

[75] Gerhart, S., et. al., “Experience with Formal Methods in Critical Systems”, *IEEE Software,* Vol. 11, Issue 1, pg. 21-28, 1994.

[76] Letier, E., and Van Lamsweerde, A., “High Assurance Requires Goal Orientation”, *International Workshop Requirements for High Assurance Systems,* 2004.

[77] Nguyen, D., et., al., “A Goal-Oriented Software Testing Methodology”, Lecture Notes in Computer Science, 2008.

[78] GRL homepage. http://www.cs.toronto.edu/km/GRL/.

[79] I* homepage. http://www.cs.toronto.edu/km/istar/.

[80] KAOS homepage. http://www.info.ucl.ac.be/research/projects/AVL/ReqEng.html.

[81] Davis, A., “A Comparison of Techniques for the Specification of External System Behavior”, *Communications of the ACM,* Vol. 31, Issue 9, pg. 1098-1115, 1988.

[82] Davis, A., “Software Requirements: Objects, Functions, and States”, Prentice-Hall, 1993.

[83] Davis A., “Software Requirements: Analysis and Specification”, Prentice-Hall, 1990.

[84] Czarnecki K., Helsen S., “Classification of Model Transformation Approaches”, *Proceedings of the 18th International Conference, OOPSLA’2003, Workshop on Generative Techniques in the context of Model Driven Architecture, Anaheim*, California, USA, October, 2003.

[85] Sendall, S., and Kozaczynski, W., “Model Transformation: The Heart and Soul of Model-Driven Software Development”, *IEEE Software,* Vol. 20, No. 5, pp. 42-45, 2003.

[86] Basin, D., Doser, J., and Lodderstedt, T., “Model Driven Security: From UML Models to Access Control Infrastructures”, *ACM Transactions on Software Engineering and Methodology,* Vol. 15, No. 1, pg. 39-91, 2006.

[87] Yu, E., “Why Agent-Oriented Requirements Engineering”, *In: Proc. of the 3rd Int. Workshop on Requirements Engineering: Foundations for Software Quality,* Barcelona, Catalonia. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1997.

[88] Cheng, B., et. al., “Using Security Patterns to Model and Analyze Security Requirements”, Technical Report MSU-CSE-03-18, Department of Computer Science, Michigan State University, July 2003.

[89] Schumacher, M., et. al., “Security Patterns - Integrating Security and Systems Engineering”, John Wiley & Sons, 2005.

[90] http://www.commoncriteriaportal.org/

[91] Ramesh, B., “Factors Influencing Requirements Traceability Practice”, *Communications of the ACM,* Vol. 41, Issue 12, pg. 37-44, 1998.

[92] Mosley, H. and A. Mayer, “Benchmarking National Labor Market Performance: A radar chart approach”, report prepared for the European Commission, Berlin, WZB discussion paper, 202, 1999.

[93] Van Lamsweerde, A., and Letier, E., “From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering”, *Proceeding of Radical Innovations of Software and Systems Engineering*, LNCS, pg. 325-340, 2004.

[94] L. Jiang, T. Topaloglou, A. Borgida, and J. Mylopoulos, “Incorporating Goal Analysis in Database Design: A Case Study from Biological Data Management.”, In *RE’06*, pages 196–204, 2006.

[95] Van Lamsweerde, A., "From System Goals to Software Architecture." in *Formal Methods for Software Architectures*. M. Bernardo and P. Inverardi, Springer-Verlag: 25-43, 2003.

[96] B. Nuseibeh, J. Kramer and A. Finkelstein, “A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications”, *IEEE Transactions on Software Engineering*, Vol. 20 No. 10, October 1994, 760-773.

[97] Van Lamsweerde, A., “Formal Specification: a Roadmap”, *Proceedings of the Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000.

[98] J. Mylopoulos, L. Chung, and E. Yu, “From Object-Oriented to Goal-Oriented Requirements Analysis”, *Communications of the ACM*, 42(1):31–37, 1999.

[99] Kaindl, H., “A practical approach to combining requirements definition and object-oriented analysis”, *Annals of Software Engineering 3*, pg. 319–343, 1997.

[100] Y. Yu, et. al., “From stakeholder goals to high-variability software design”, Tech. Rep. CSRG-509, Canada, Ontario: University of Toronto, 2005.

[101] William Heaven and Anthony Finkelstein, “A UML profile to support requirements engineering with KAOS”, *IEE Proceedings: Software*, Vol.151, Issue No. 1, pg. 10–27, 2004.

[102] K. Yue, “What Does It Mean to Say that a Specification is Complete?”, *Proc. IWSSD- 4, Fourth International Workshop on Software Specification and Design*, Monterey, 1987.

[103] A. van Lamsweerde, “Requirements Engineering in the Year 00: A Research Perspective”. Invited Keynote Paper, *Proc. ICSE’2000: 22nd International Conference on Software Engineering*, ACM Press, 2000, 5-19.

[104] A. Dardenne, A. van Lamsweerde and S. Fickas, “Goal-Directed Requirements Acquisition”, *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[105] Massacci, Fabio, et. al., "Using a Security Requirements Engineering Methodology in Practice: The Compliance with The Italian Data Protection Legislation", *Computer Standards & Interfaces,* issue 27, pg. 445-455, 2005.

[106] Van Lamsweerde, A., Darimont, R. and Letier, E., "Managing Conflicts in Goal-driven Requirements Engineering", *IEEE Transactions on Software Engineering*, Special Issue on Inconsistency Management in Software Development, Vol. 24, No. 11, pg. 908-926, November 1998.

[107] Sedki, Ahmed, "A Structured Approach to Adoption Agile Practices: The Agile Adoption Framework", Ph.D Disseration, Virginia Tech University, 2007.

# Appendix A: The Spy Network Case Study Details

This appendix is concerned with outlining how obstacles to the security goals of the \ system are derived and resolved.

**Obstacles and resolutions for security goals**

Obstacles to the goal RevelationIntegrity has already been illustrated in section 4.1.3.1 and the obstacles to the rest of the security goals (confidentiality, authentication, availability, access control) are illustrated in this appendix.(confidentiality, authentication, availability, access control).

**A.1 Obstacle to the goal RevelationConfidentiality**

**A.1.1 Generating the obstacle to the goal RevelationConfidentiality**

Revelations are sent in plain text over the network and only spies in the same team are allowed to know their contents. Malicious passive eavesdropper can listen to the network and intercept messages. Interception is only passive, which means that the attacker knows the content of the revelation, but does not remove them and neither the sender nor the receiver knows that someone has intercepted the message.
Negating the goal RevelationConfidentiality gives the following:

**Obstacle** RevelationKnownByNonTeamMember
**InformalDef** A spy knows a revelation and is not in the team
**FormalDef** $\Diamond\ \exists$ sp : Spy, re : Revelation, te1, te2 : Team
Member(sp, te1) $\wedge$ Targeted(re, te2) $\wedge$ te1 $\neq$ te2 $\wedge$ Knows(sp, re.Content)

This yields the following object model increment:

**Relationship** Intercepted
**InformalDef** A spy has stolen the content of a message on the network and owns it although it was not the intended recipient
**Links** Spy {**Role** Intercepts, **Card** 0:N}
Revelation {**Role** Intercepted, **Card** 0:N}
**DomInvar** ($\forall$ sp : Spy, re : Revelation) [Intercepted(sp, re) $\Rightarrow$ Knows(sp, re.Content)]

Regressing through this domain property, the obstacle becomes:

**Obstacle** RevelationOwnedByNonTeamMember
**InformalDef** A spy has intercepted a revelation and is not in the team
**FormalDef** $\Diamond\ \exists$ sp : Spy, re: Revelation, te1, te2 : Team
Member(sp, te1) $\wedge$ Targeted(re, te2) $\wedge$ te1 $\neq$ te2 $\wedge$ Intercepted(sp, re)

If a spy is able to intercept a revelation, he will know its content. It is desired that intercepting spies are not able to understand the revelation.

### A.1.2 Resolving the obstacle to the goal RevelationConfidentiality

A solution to this obstacle is to encrypt the revelation with a secret key that spies share in a team. In this case, Owning a revelation and Knows a revelation are different. The goal restoration technique is being used to generate a stronger goal:

**Goal** *Maintain* [InterceptedRevelationConfidentiality]
**InformalDef** Only team member know a revelation even if it has been intercepted
**InstanceOf** ConfidentialityGoal
**FormalDef** ∀ sp : Spy, re : Revelation, te1, te2 : Team
Intercepted(sp, re) ∧ Targeted(re, te1) ∧ Member(sp, te2) ∧ te1 ≠ te2
⇒ ¬ Knows(sp, re.Content)

This goal is refined into the following subgoals:

**Goal** *Maintain* [TargetedRevelationEncrypted]
**InformalDef** A revelation is encrypted with the team's secret key when targeted
**Refines** InterceptedRevelationConfidentiality
**FormalDef** ∀ re : Revelation, te : Team
Targeted(re, te) ⇒ (∃ sk : SecretKey) Encrypted(re, sk) ∧ Targeted(sk, te)

**Goal** *Maintain* [InterceptedEncryptedRevelation]
**InformalDef** An intercepted encrypted revelation is not known by the intercepter
**Refines** InterceptedRevelationConfidentiality
**FormalDef** ∀ sp : Spy, re : Revelation, te1, te2 : Team, sk : SecretKey
Intercepted(sp, re) ∧ Targeted(re, te1) ∧ Member(sp, te2) ∧ te1 ≠ te2
∧ Encrypted(re, sk) ∧ Targeted(sk, te)
⇒ ¬ Knows(sp, re.Content)

The goal TargetedRevelationEncrypted is assigned to the revelation author.
The goal InterceptedEncryptedRevelation needs to be refined by the two following subgoals:

**Goal** *Maintain* [KeyOwnedByTeam]
**InformalDef** A spy owns a secret key iff he is in the secret key's team
**Refines** InterceptedEncryptedRevelation
**FormalDef** ∀ sp : Spy, sk : SecretKey, te1, te2 : Team
Targeted(sk, te1) ∧ Member(sp, te2) ∧ te1 ≠ te2 ⇒ ¬ Owning(sp, sk)]

**Goal** *Maintain* [InterceptedRevelationWithoutKey]
**InformalDef** An intercepter without the key does not know the revelation
**Refines** InterceptedEncryptedRevelation
**FormalDef** ∀ sp : Spy, re : Revelation, te : Team, sk : SecretKey
Intercepted(sp, re) ∧ Encrypted(re, sk) ∧ Targeted(re, te) ∧ ¬ Owning(sp, sk)
⇒ ¬ Knows(sp, re.Content)

Here there is a chicken-and-egg problem for the goal KeyOwnedByTeam. It is operationalized by passing along the secret keys on a very secure channel so that they cannot be intercepted. A good way to implement this is to pass along the key in a meeting between the two spies. It can be assigned to the Securitas agent and thus becomes an assumption for the system environment.

The following object model increment is needed

**Entity** SecretKey
    **InformalDef** A secret key is a key in a symmetric scheme that is the shared secret

**Relationship** Encrypted
    **InformalDef** A revelation is encrypted with a secret key
    **Links** Revelation {**Role** Encrypted, **Card** 1:N}
        SecretKey {**Role** Encrypts, **Card** 1:1}
    **DomInvar** $\forall$ sp : Spy, re : Revelation, sk : SecretKey
        Owning(sp, re) $\wedge$ Encrypted(re, sk) $\wedge \neg$ Owning(sp, sk) $\Rightarrow \neg$ Knows(sp, re.Content)

The goal InterceptedRevelationWithoutKey is implied by this last domain property. The goal graph can be shown graphically:

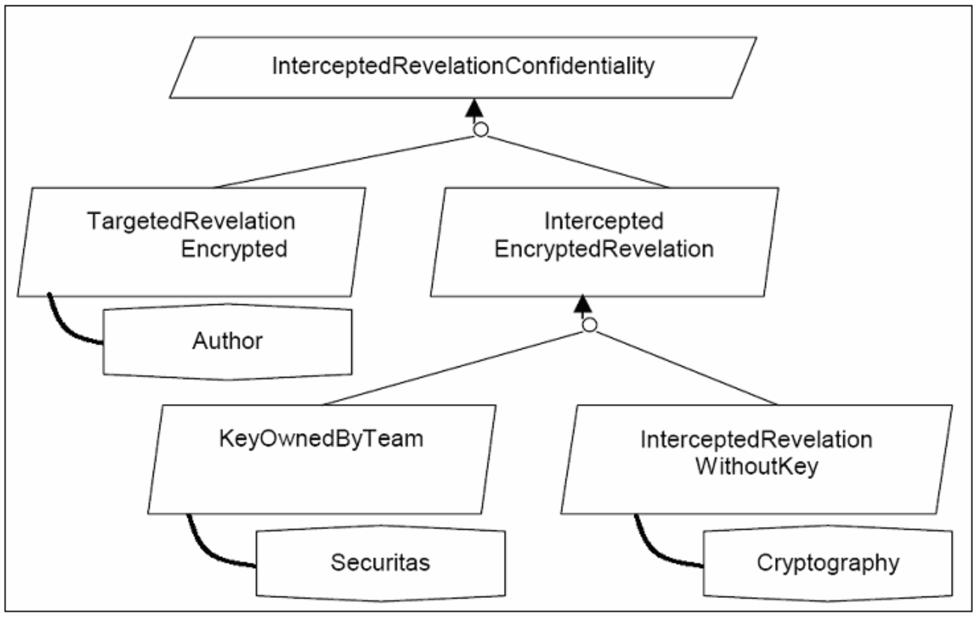

InterceptedRevelationConfidentiality
TargetedRevelation Encrypted
Intercepted EncryptedRevelation
Author
KeyOwnedByTeam
InterceptedRevelation WithoutKey
Securitas
Cryptography

**Figure 50: Refinement for the Intercepted obstacle resolution**

The drawback of the encryption and decryption tasks is that there is more computing time consumed in the transmission of messages. Besides, this does not resolve the case when a spy leaves a team. If the key remains the same, the spy will still be able to decrypt the messages. One solution would be to send a new key to all members of a team when a spy has left them.

## A.2 Obstacle to the goal RevelationAuthentication

### A.2.1 Generating the obstacle to the goal RevelationAuthentication

Negating the goal RevelationAuthentication gives the following:

**Obstacle** RevelationNotAuthenticated
  **InformalDef** A spy owns a revelation and does not know its author
  **FormalDef** $\Diamond \exists$ sp1, sp2 : Spy, re: Revelation
    AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge \neg$ Knows(sp2, AuthorOf[sp1, re])

The following object model increment is required:

**Relationship** Anonymous
  **InformalDef** A revelation does not mention any author name on it
  **Links** Revelation {**Role** Caracterised, **Card** 0:N}
  **DomInvar** $\forall$ sp1, sp2 : Spy, re : Revelation
    AuthorOf(sp1, re) $\wedge$ Anonymous(re) $\wedge$ Owning(sp2, re)
      $\Rightarrow \neg$ Knows(sp2, AuthorOf[sp1, re])

Regressing through this domain property, we get:

**Obstacle** RevelationNotAuthenticated
  **InformalDef** A spy own a revelation and it is anonymous
  **FormalDef** $\Diamond \exists$ sp1, sp2 : Spy, re: Revelation
    AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge$ Anonymous(re)

This could happen if the message is anonymous or if a spy has forged the author of a revelation. A scheme to certify messages with respect to its author is introduced later.

### A.2.2 Resolving the obstacle to the goal RevelationAuthentication

In order to make the signatures relevant, we need to make sure that a spy is accurately identified by his public key. This means that a trusted third party is needed to certify that an association between a public key and a spy identity is accurate. A certification is such an association that has been signed by a more trusted authority.

In order to bootstrap the verification spies, such spies have to get a public key from a certification authority though a secure and authenticated channel. For instance, they could be given it by hand when they sign the contract when joining the spy agency.

**Goal** *Maintain* [RevelationAuthenticationWhenAnonymous]
**InformalDef** A signed revelation is authenticated even if the author name is not mentioned
**InstanceOf** AuthenticationGoal
**FormalDef** $\forall$ sp1, sp2 : Spy, re: Revelation
AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge$ Anonymous(re) $\Rightarrow$ Knows(sp2, AuthorOf[sp1, re])

This goal needs to be refined as follows:

**Goal** *Maintain* [RevelationSignedWhenReceived]
**InformalDef** A spy who owns a revelation also owns its signature and knows whether the author is certified by its public key
**Refines** RevelationAuthenticationWhenAnonymous
**FormalDef** $\forall$ sp1, sp2 : Spy, si : Signature, re: Revelation, puk : PublicKey, prk : PrivateKey
AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge$ Anonymous(re)
$\Rightarrow$ Owning(sp2, si) $\wedge$ Signed(re, si, prk) $\wedge$ Owning(sp2, puk)
$\wedge$ Knows(sp2, Certified[sp1, puk]) $\wedge$ KeyPair(prk, puk)

**Goal** *Maintain* [SignedRevelationVerified]
**InformalDef** A spy who verifies a signature knows whether the purported author is correct
**Refines** RevelationAuthenticationWhenAnonymous
**FormalDef** $\forall$ sp1, sp2 : Spy, si : Signature, re: Revelation, puk : PublicKey, prk : PrivateKey
AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge$ Owning(sp2, si) $\wedge$ Signed(re, si, prk)
$\wedge$ Knows(sp2, Certified[sp1, puk]) $\wedge$ KeyPair(prk, puk) $\wedge$ Owning(sp2, puk)
$\wedge$ Verified(re, si, puk).Genuine
$\Rightarrow$ Knows(sp2, AuthorOf[sp1, re])

The goal RevelationSignedVerified is verified by a domain property of the relationship Verified.

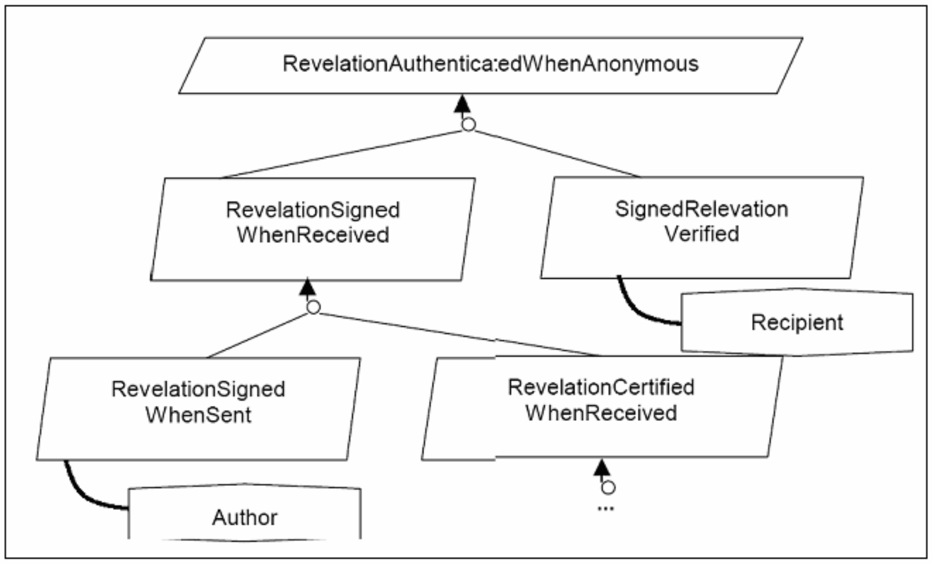

RevelationAuthentica:edWhenAnonymous
RevelationSigned WhenReceived
SignedRelevation Verified
Recipient
RevelationSigned WhenSent
RevelationCertified WhenReceived
Author
...

**Figure 51: Refinement for the Anonymous obstacle resolution**

The goal RevelationSignedWhenReceived needs to be refined as follows:

**Goal** *Maintain* [RevelationSignedWhenSent]
**InformalDef** A revelation is signed when it is sent by the author
**Refines** RevelationSignedWhenReceived
**FormalDef** $\forall$ sp1, sp2 : Spy, si : Signature, re: Revelation, puk : PublicKey, prk : PrivateKey
AuthorOf(sp1, re) $\Rightarrow$ ($\exists$ si : Signature, prk : PrivateKey, puk : PublicKey)
Signed(re, si, prk) $\wedge$ Certified(sp1, puk) $\wedge$ KeyPair(prk, puk)

**Goal** *Maintain* [RevelationCertifiedWhenReceived]
**InformalDef** A signed revelation is verified by the receiver
**Refines** RevelationSignedWhenReceived
**FormalDef** $\forall$ sp1, sp2 : Spy, si : Signature, re: Revelation, puk : PublicKey, prk : PrivateKey
AuthorOf(sp1, re) $\wedge$ Owning(sp2, re) $\wedge$ Signed(re, si, prk) $\wedge$ Verified(re, si, puk).Genuine
$\Rightarrow$ Owning(sp2, si) $\wedge$ Owning(sp2, puk) $\wedge$ Knows(sp2, Certified[sp1, puk])

The goal RevelationSignedWhenSent can be assigned to the revelation author. The goal RevelationCertifiedWhenReceived needs to be refined with a message passing pattern. The details of the refinement is not provided here.

### A.3 Obstacle to the goal RevelationAvailability

Availability goals cannot be stated formally as a global constraint on the system. For every goal, the resources needed for its achievement need to be identified. The availability of the whole system depends on the availability of its components.

The obstacle FloodWithMessage is caused when a spy is busy. This means that if the spy who is assigned the task of sending messages is not available, no spy in the team knows the revelation. A resolution strategy for this obstacle is to assign the forwarding goal to another relay. We need to detect that a relay is unavailable, for instance by measuring the round trip time it takes for him to acknowledge a message. In practice, this relay reallocation cannot be instantaneous. The relay is only reallocated when he turns out to be unreliable in the long term. Another resolution strategy is to introduce redundancy with several relays. When a spy collects a revelation, he sends it all relays hoping that one of them will be available.

### A.4 Obstacle to the goal MailboxAccessControl

#### A.4.1 Generating the obstacle to the goal MailboxAccessControl

A mailbox is accessed only by its subscribed spy. In a system where no access control is provided, spies can access other mailboxes if they can guess what the settings to access the mailbox are. Therefore, introducing passwords in the system is required so that access is granted only to the spy knowing the mailbox password.

**Obstacle** MailboxAccessedByIntruder

**InformalDef** A mailbox is accessed by an intruder

**FormalDef** $\Diamond \exists$ sp1, sp2 : Spy, ma : Mailbox

Accessed(ma, sp1) $\wedge$ Subscribed(ma, sp2) $\wedge$ sp1 $\neq$ sp2

This leads to a simple attack scenario where a spy accesses the mailbox of another spy.

### A.4.2 Resolving the obstacle to the goal MailboxAccessControl

The goal MailboxAccessControl needs to be refined the following way:

**Goal** *Maintain* [MailboxPasswordKnown]

**InformalDef** Mailboxes passwords are known only by their subscribed spy

**Refines** MailboxesAccessControl

**FormalDef** $\forall$ sp : Spy, ma : Mailbox

Subscribed(ma, sp) $\Leftrightarrow$ Knows(sp, ma.Password)

**Goal** *Maintain* [MailboxAccessWithPassword]

**InformalDef** A mailbox is accessed only by a spy who knows its password

**Refines** MailboxesAccessControl

**FormalDef** $\forall$ sp : Spy, ma : Mailbox

Accessed(ma, sp) $\Rightarrow$ Knows(sp, ma.Password)

The goal MaiboxAccessWithPassword can be assigned to the mailbox access controller. The goal mailboxPasswordKnown cannot be assigned to a particular system agent. The password cannot be sent through the system because of an obvious chicken-and-egg problem. It is an assumption that is assigned to agents in the environment.

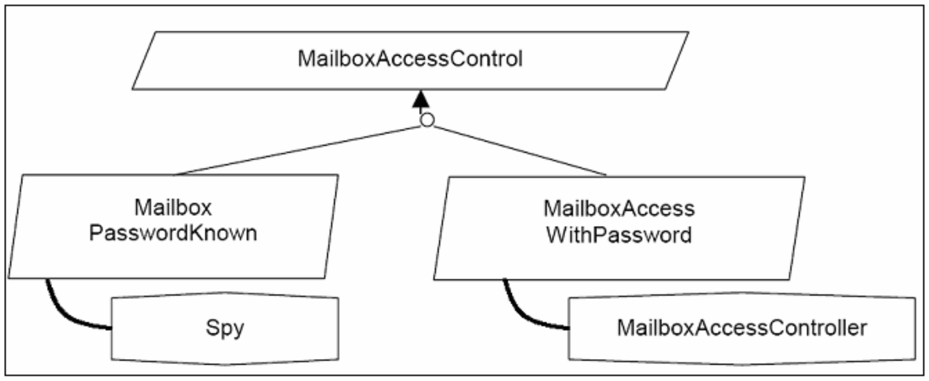


**Figure 52: Refinement of the IllegitimateAccess obstacle resolution**

This assumption generates two organizational rules for the environment:

- All spies know their mailbox password.

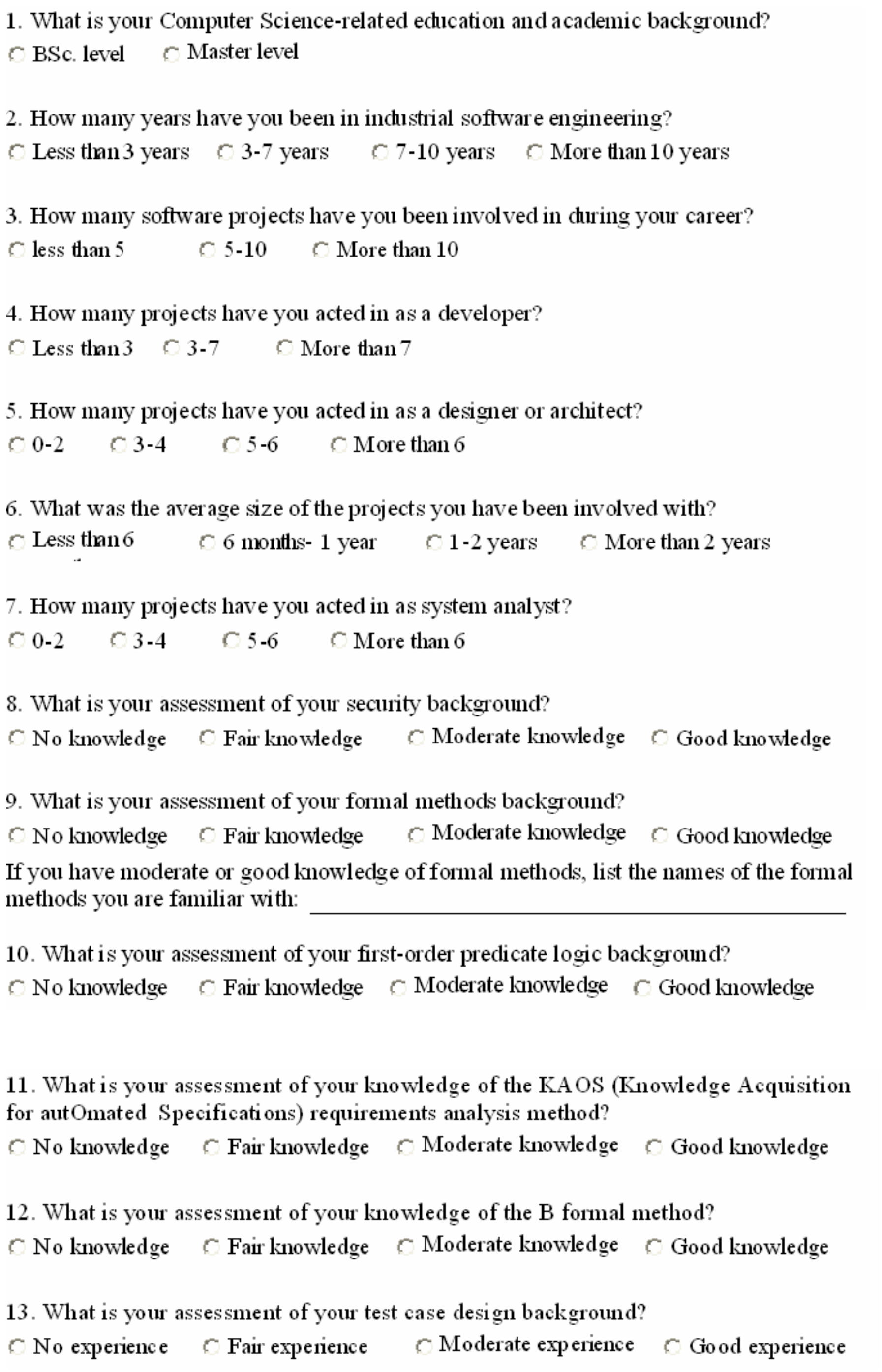
- Only a spy subscribed to a mailbox is authorized to know its password.

These rules need to be enforced by the big boss giving a new password to a spy by hand. Spies are required not to forget their passwords and they should be held responsible when someone else is caught using their password.

In the scenario where a spy is able to access another spy mailbox, there are security issues involved, mainly confidentiality issues.

# Appendix B: Evaluation Experiment

This appendix outlines the two questionnaires given to subjects to select the participants before the experiment and to assess the approaches at the end of the experiment. The subject background questionnaire is as follows:



1. What is your Computer Science-related education and academic background?
○ BSc. level ○ Master level

2. How many years have you been in industrial software engineering?
○ Less than 3 years ○ 3-7 years ○ 7-10 years ○ More than 10 years

3. How many software projects have you been involved in during your career?
○ less than 5 ○ 5-10 ○ More than 10

4. How many projects have you acted in as a developer?
○ Less than 3 ○ 3-7 ○ More than 7

5. How many projects have you acted in as a designer or architect?
○ 0-2 ○ 3-4 ○ 5-6 ○ More than 6

6. What was the average size of the projects you have been involved with?
○ Less than 6 ○ 6 months- 1 year ○ 1-2 years ○ More than 2 years

7. How many projects have you acted in as system analyst?
○ 0-2 ○ 3-4 ○ 5-6 ○ More than 6

8. What is your assessment of your security background?
○ No knowledge ○ Fair knowledge ○ Moderate knowledge ○ Good knowledge

9. What is your assessment of your formal methods background?
○ No knowledge ○ Fair knowledge ○ Moderate knowledge ○ Good knowledge
If you have moderate or good knowledge of formal methods, list the names of the formal methods you are familiar with: ______________________________

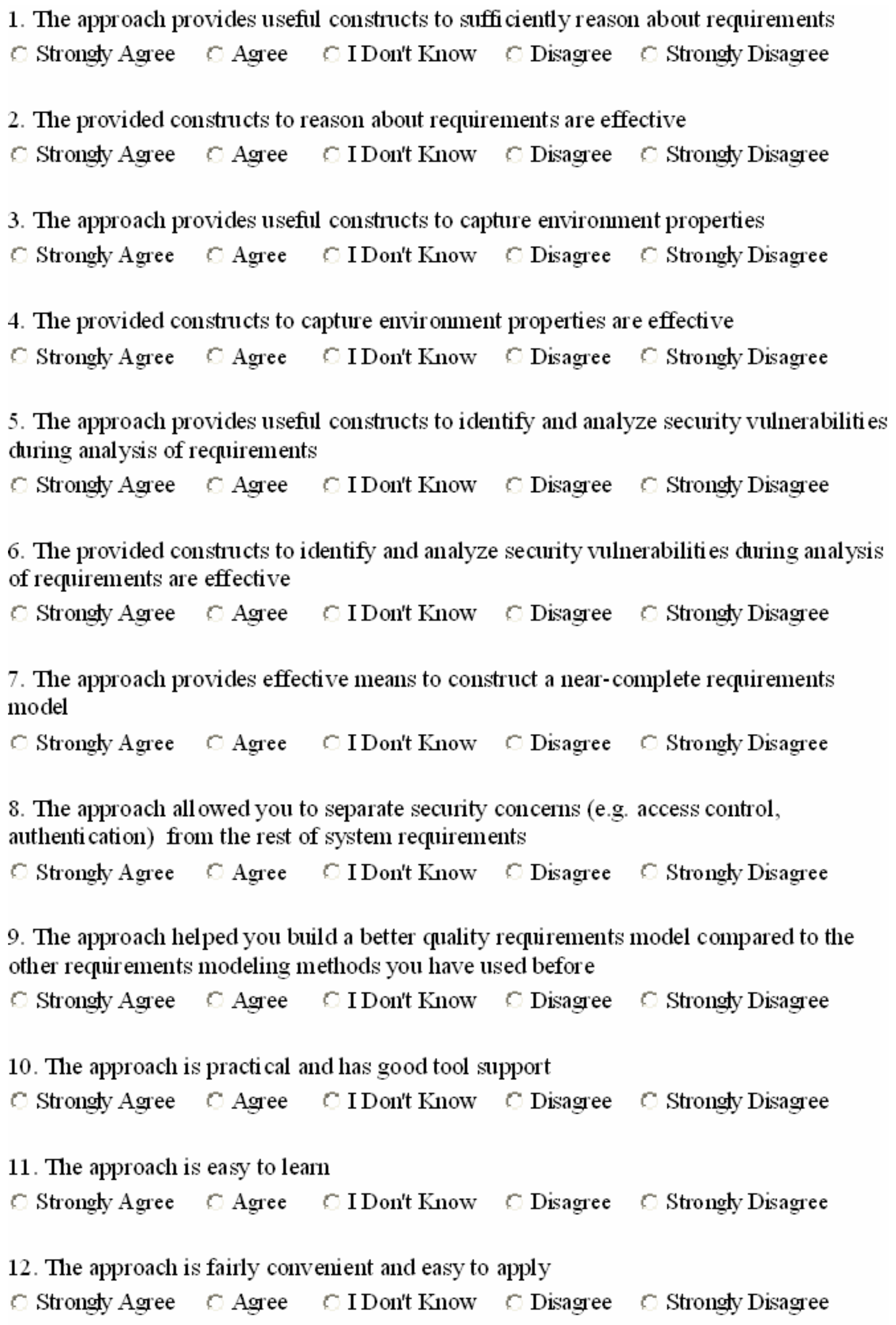
10. What is your assessment of your first-order predicate logic background?
○ No knowledge ○ Fair knowledge ○ Moderate knowledge ○ Good knowledge

11. What is your assessment of your knowledge of the KAOS (Knowledge Acquisition for autOmated Specifications) requirements analysis method?
○ No knowledge ○ Fair knowledge ○ Moderate knowledge ○ Good knowledge

12. What is your assessment of your knowledge of the B formal method?
○ No knowledge ○ Fair knowledge ○ Moderate knowledge ○ Good knowledge

13. What is your assessment of your test case design background?
○ No experience ○ Fair experience ○ Moderate experience ○ Good experience

The questionnaire given to participants at the end of the experiment is as follows:



1. The approach provides useful constructs to sufficiently reason about requirements

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

2. The provided constructs to reason about requirements are effective

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

3. The approach provides useful constructs to capture environment properties

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

4. The provided constructs to capture environment properties are effective

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

5. The approach provides useful constructs to identify and analyze security vulnerabilities during analysis of requirements

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

6. The provided constructs to identify and analyze security vulnerabilities during analysis of requirements are effective

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

7. The approach provides effective means to construct a near-complete requirements model

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

8. The approach allowed you to separate security concerns (e.g. access control, authentication) from the rest of system requirements

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

9. The approach helped you build a better quality requirements model compared to the other requirements modeling methods you have used before

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

10. The approach is practical and has good tool support

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

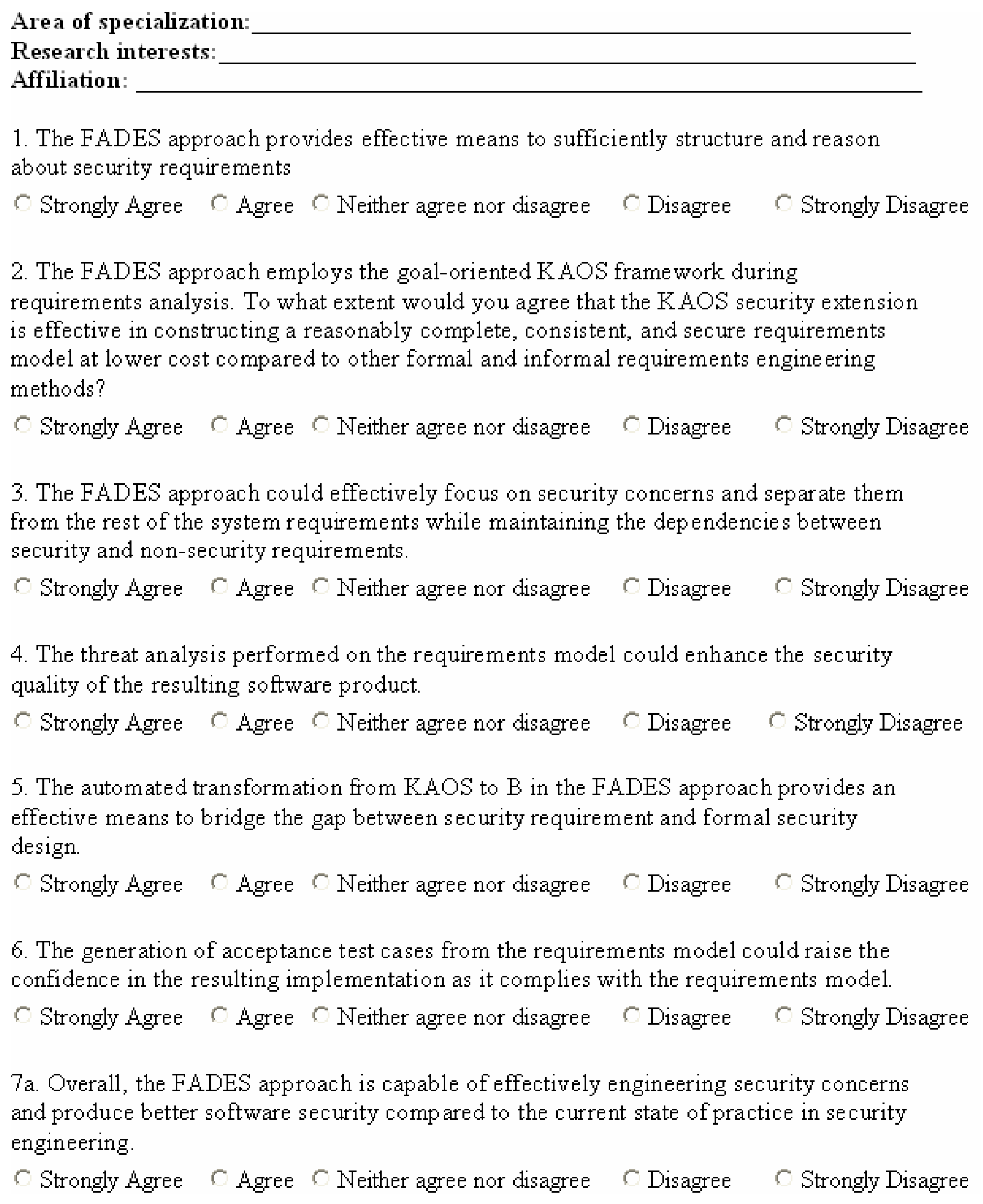
11. The approach is easy to learn

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

12. The approach is fairly convenient and easy to apply

○ Strongly Agree ○ Agree ○ I Don't Know ○ Disagree ○ Strongly Disagree

The questions given to the experts to collect their feedback are as follows:



**Area of specialization:**__________________________________________
**Research interests:**____________________________________________
**Affiliation:** ________________________________________________

1. The FADES approach provides effective means to sufficiently structure and reason about security requirements

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

2. The FADES approach employs the goal-oriented KAOS framework during requirements analysis. To what extent would you agree that the KAOS security extension is effective in constructing a reasonably complete, consistent, and secure requirements model at lower cost compared to other formal and informal requirements engineering methods?

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

3. The FADES approach could effectively focus on security concerns and separate them from the rest of the system requirements while maintaining the dependencies between security and non-security requirements.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

4. The threat analysis performed on the requirements model could enhance the security quality of the resulting software product.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

5. The automated transformation from KAOS to B in the FADES approach provides an effective means to bridge the gap between security requirement and formal security design.

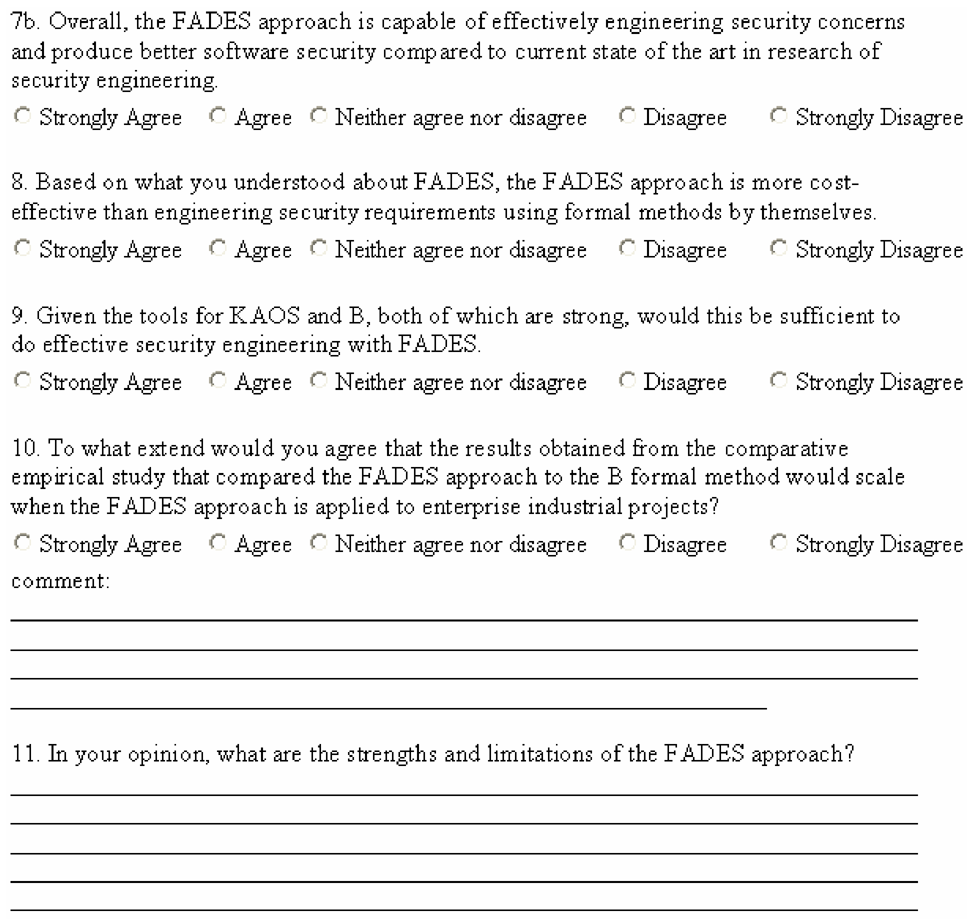
○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

6. The generation of acceptance test cases from the requirements model could raise the confidence in the resulting implementation as it complies with the requirements model.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

7a. Overall, the FADES approach is capable of effectively engineering security concerns and produce better software security compared to the current state of practice in security engineering.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree



7b. Overall, the FADES approach is capable of effectively engineering security concerns and produce better software security compared to current state of the art in research of security engineering.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

8. Based on what you understood about FADES, the FADES approach is more cost-effective than engineering security requirements using formal methods by themselves.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

9. Given the tools for KAOS and B, both of which are strong, would this be sufficient to do effective security engineering with FADES.

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

10. To what extend would you agree that the results obtained from the comparative empirical study that compared the FADES approach to the B formal method would scale when the FADES approach is applied to enterprise industrial projects?

○ Strongly Agree ○ Agree ○ Neither agree nor disagree ○ Disagree ○ Strongly Disagree

comment:

11. In your opinion, what are the strengths and limitations of the FADES approach?