Introducing distributed ledger security into system specifications with the Isabelle RR-cycle

Florian Kammüller

Middlesex University London and
Technische Universität Berlin
f.kammueller@mdx.ac.uk

Abstract. We present an approach to developing secure system specifications for IoT systems with decentralized data using the Refinement-Risk cycle (RR-cycle), a method for security engineering implemented in the proof assistant Isabelle. The RR-cycle enables interleaving attack analysis with system refinement using rigorous machine assisted proof in Isabelle to scrutinize and refine system specifications until security requirements are met. We illustrate this approach by a case study of a privacy critical scenario by refining it with a distributed ledger. The case study is motivated by the IoT project SUCCESS on security and privacy of healthcare IoT applications. We briefly summarize the RR-cycle method before focusing on its application of identifying a privacy attack that leads to a security refinement introducing the distributed ledger.

1 Introduction

In general, rigorous specification is justified despite its cost in the long run for any system as it facilitates and thus economizes maintenance and adaptation. For complex and hybrid systems, like IoT systems, for security and privacy critical applications, like healthcare, the application of formal methods is justified in any case. The approach we advocate in this paper, is to use the classical software engineering approach of a top down rigorous development process starting from an abstract specification based on requirements engineering. This initial specification is then refined step by step to a more concrete one while preserving properties. We use the method called Refinement-Risk cycle (RR-cycle) [10,14] that integrates a state based system view of infrastructures with actors and policies as basis for the representation of IoT systems and is implemented in the interactive proof assistant Isabelle.

We first give a brief summary of the Isabelle RR-cycle [10,14] and the IoT healthcare case study in Section 2 to provide the background for understanding its application. In Section 3 we re-iterate a previous RR-cycle application to the case study [14]. By introducing a distributed ledger we can show that now the RR-cycle terminates with the global privacy property. We discuss, conclude and present related work in Section 4.

2 Summary of RR-cycle and case study

The RR-cycle [10,14] has emerged as a result of the CHIST-ERA project SUCCESS [4] addressing the formal modeling and analysis of security and privacy of IoT in healthcare. The starting point of this cycle is an initial formal specification that may have been produced by a formal requirement elicitation method like in the IoT case study [10,14] but may also be an ad hoc formalization of system requirements. The RR-cycle improves this initial specification by interleaving refinement with attack analysis. The process is directed by a global security property. The RR-cycle is the process that is used to drive security engineering in the Isabelle Insider and Infrastructure framework (IIIf). Attack trees, Kripke structures and modelchecking as well as a formal notion of property preserving refinement are formalized within the framework of the IIIf. This process is graphically depicted in Figure 1. For a simple healthcare scenario we have applied

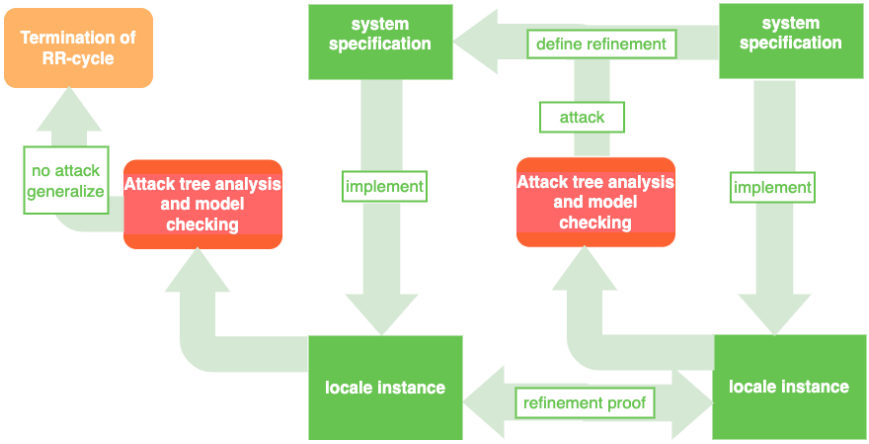

Fig. 1. Refinement-Risk-Cycle iterates design, risk analysis, and refinement

the Fusion/UML process for object-oriented software development to derive a UML system design. One of the major outcomes of this process is the system class model shown in Figure 2. In addition to this static system architecture, the Fusion/UML model also provides a set of operation schemas, use cases, and so-called object collaborations illustrating the method flows within the objects of the system class model. The complete analysis and design documentation is available [13]. The arxiv paper [14] already presented a detailed model and analysis of the first few iterations of the RR-cycle to this case study based on an earlier workshop paper [10] that informally described the stepwise development but without formalizations. A summary of the iterations of the RR-cycle, the attacks exhibiting vulnerabilities, and the countermeasures used for the refinement

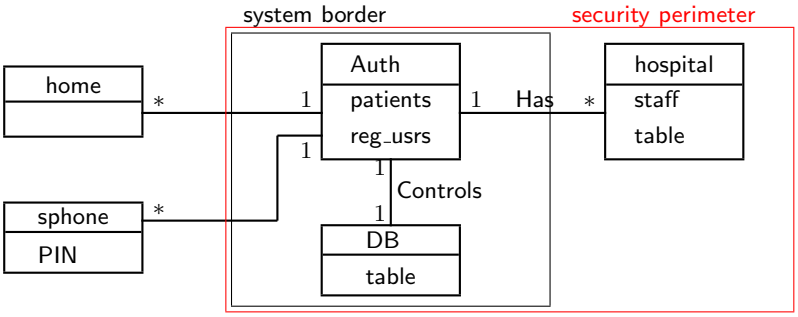

Fig. 2. System class model for IoT healthcare system

in these earlier papers [10,14] is provided in Table 3. However, despite introducing already a blockchain formalization (Iteration 2), the earlier formalization is inadequate. Consequently the RR-cycle could not be successfully terminated – also because the termination condition of the RR-cycle has been identified only later [11]. In the current paper, we thus re-start the RR-cycle after Iteration 2 and introduce a more adequate definition of a distributed ledger. We then show that it guarantees the global security property of privacy of data.

System	**Attack**	**Refinement**	**Where**
Initial Fusion system home-cloud-hospital	Eve can perform action get at cloud	Introduce access control by DLM labels	`RRLoopOne.thy` `hcKripkeOne.thy`
Refinement-Risk-Cycle Iteration 1			
Access control by DLM labels	Eve can perform action eval at cloud; changes label to her own	Introduce privacy preserving functions and enforce their use within system	`RRLoopTwo.thy` `hcKripkeTwo.thy`
Refinement-Risk-Cycle Iteration 2			
Privacy preserving functions type `label_fun`	Eve puts Bob's data labeled as her own	Introduce blockchain	`RRLoopThree.thy` `hcKripkeThree.thy`
Refinement-Risk-Cycle Iteration 3			
Global distributed ledger	Global privacy theorem proved	Done	`LedgerRRLoop.thy` `LedgerhcKripke.thy`
Termination of Refinement-Risk-Cycle			

Fig. 3. Iterated application of Refinement-Risk(RR)-Cycle

3 IoT model in IIIf, attack trees analysis and refinement to distributed ledger

In this section, we first introduce the specification after RR-cycle iteration level 2 (see Table 3) where DLM access control labels [16] and privacy preserving functions have already been introduced into the system. We then exhibit an attack using attack tree analysis and present the refinement where a distributed ledger replaces the initial ad hoc data management.

3.1 IoT healthcare model in IIIf

The IIIf allows representing infrastructures as graphs where the actors and local policies are attached to the nodes. Infrastructures are the *states* of the system. We use the underlying theory of Kripke structures and temporal logics CTL in the IIIf to define a specific state transition relation for our application scenario. By instantiating the generic Kripke structure state transition relation to each of the refinements of the application scenario, we inherit the underlying theory of CTL and attack trees to provide security analysis in Isabelle.

The infrastructure graph is defined as a datastructure `igraph` whose components `gra`, `agra`, `cgra`, and `lgra` represent the parts as: a set of pairs of nodes; the actor identities at each node; the credentials and roles assigned to actors; and the assignment of `dlm` labeled data at each location. The label type `dlm` is a synonym for `actor × actor set` combining the owner and the set of readers into a pair. The constructor `Lgraph` puts these components into an `igraph`.

```
datatype igraph = Lgraph gra: location × location)set
                         agra: location ⇒ identity set
                         cgra: actor ⇒ (string set × string set)
                         lgra:  location ⇒ (data × dlm) set
```

We then instantiate igraphs to *locales* (see Figure 1). The following `ex_graph` is the concrete `igraph` example used in the IoT healthcare case study locale where `ex_creds` and `ex_locs` are example credentials and data (omitted here [12]).

```
ex_graph ≡ Lgraph {(home,cloud), (sphone,cloud), (cloud,hospital)}
                  (λ x. if x = home then {''Patient''} else
                       (if x = hospital then {''Doctor''} else {}))
                  ex_creds ex_locs
```

Infrastructures are generally given by the following datatype that contains an infrastructure graph of type `igraph` and a policy given by a function that assigns local policies over a graph to all locations of the graph.

```
datatype infrastructure = Infrastructure  igraph
                                          [igraph, location] ⇒ policy set
```

For our healthcare example, the instantiated infrastructure contains the above graph `ex_graph` and the local policies defined subsequently.

```
hc_scenario ≡ Infrastructure ex_graph local_policies
```

The function `local_policies` defines the policy for an application: for each location `x` over an infrastructure graph `G` as sets of pairs. The first element of a pair is a function specifying the actors `y` that are entitled to perform the actions specified in the set which is the second element of that pair.

```
local_policies G x ≡
case x of
  home ⇒ {(λ y. True, {put,get,move,eval})}
| sphone ⇒ {((λ y. has G (y,''PIN'')), {put,get,move,eval})}
```

```
| cloud ⇒ {(λ y. True, {put,get,move,eval})}
| hospital ⇒ {((λ y. (∃ n. (n  @G hospital) ∧
              Actor n = y ∧ has G (y, ''skey'')), {put,get,move,eval})}
| _ ⇒  {})
```

In general, policies specify the expected behaviour of actors of an infrastructure. They are given by pairs of predicates (conditions) and sets of (enabled) actions. Policies are controlled by the `enables` predicate: an actor `h` is enabled to perform an action `a` in infrastructure `I`, at location `l` if there exists a pair `(p,e)` in the local policy of `l` (`delta I l` projects to the local policy) such that the action `a` is a member of the action set `e` and the policy predicate `p` holds for actor `h`.

```
enables I l h a ≡ ∃ (p,e) ∈ delta I l. a ∈ e ∧ p h
```

Infrastructure State Transition The generic state transition relation uses the syntactic infix notation `I` $\rightarrow_n$ `I'` to denote that infrastructures `I` and `I'` are in this relation. To give an impression of this definition, we show here just one of several rules that defines the state transition for the action `put` because this rule will be crucial in the following attack analysis. The rule `put` assumes an actor `h` residing at a location `l` in the infrastructure graph `G` and being enabled the `put` action. If infrastructure state `I` fulfils those preconditions, the next state `I'` can be constructed from the current state by adding the data item `d` with label `(h, hs)` at location `l`. The addition is given by updating (using `:=`) the component `lgra` by adding this new labeled data item.

```
put: G = graphI I ⟹ h @G l ⟹ enables I l (Actor h) put ⟹
     I' = Infrastructure
            (Lgraph (gra G)(agra G)(cgra G)(lgra G)
                    ((lgra G)(l := (lgra G l) ∪ {((Actor h, hs), d)})))
            (delta I)
    ⟹ I →n I'
```

3.2 Attack tree analysis

The goal of the security analysis is also described by a policy: the global policy. The global policy is 'only the patient and the doctor can access the data in the cloud'. In the first two iterations of the RR-cycle, this global policy has twice been attacked (see Table 3) by first simply taking the data away (action `get`) and second, after introducing dlm-access control labels, by overwriting the access control label of a data item using the `eval` action. This attack can be prevented by enforcing the use of privacy preserving functions (for details see the source code [12] and [14]). However, even now there is still an attack possible.

We consider next the following global policy expressing data privacy.

```
global_policy I a ≡ ∀ l ∈ nodes(graphI I). ∀ l' ∈ nodes(graphI I).
         ∀ d:: data. ∀ lb:: dlm. ∀ lb':: dlm.
         (lb, d) ∈ lgra(graphI I l) ⟶ (lb', d) ∈ lgra(graphI I l')
         ⟶ lb = lb'
```

It expresses privacy by saying that different occurrences of the same data in the system must have the same labels. Trying to prove this global policy, we fail. The reason for this is exhibited by the following attack which we find by attack tree analysis (for details see [14,12]). Using the CTL semantics of attack trees in IIIf [9], allows translating the attack into an EF property showing that there exists a path (E) on which eventually (F) Eve can put data on the cloud.

```
hc_KripkeF ⊢  EF I. enables I cloud (Actor ''Eve'') put
```

That is, Eve could learn the data by other means than using the privacy preserving functions introduced in the second iteration of the RR-cycle and use the action `put` to enter that data as new data to the system but labelled as her own data. As a countermeasure, we need a concept to guarantee consistency of data labeling across the system: we introduce a distributed ledger also known as a blockchain.

3.3 Introducing distributed ledger via refinement

We use the inherent uniqueness of the function type in Isabelle to provide a definition of a distributed ledger in a concise way as a type synonym.

```
type_synonym ledger = data ⇒ (dlm × location set)option
```

A ledger is now a type of "partial" functions that maps a data item to a pair of the data's label and the set of locations where the data item is registered. Since all functions in HOL are total, we use a standard Isabelle way of representing partial functions using the type constructor `option`. This type constructor lifts a type α to the type α `option` which consists of the unique constant `None` and the range of elements `Some x` for all `x` $\in \alpha$.

Since the type `ledger` is a function type, an element of type ledger maps a data item `d` to at most one range element `Some(l,L)` (or to `None` if this data element is not in use in the system). Thus, every data item `d` has at most one valid data label `l` of type `dlm` and one unique list of current infrastructure locations `L` where this data item is located.

These observations about the definition of the type ledger can be exploited formally in Isabelle and proved accordingly (∃! is unique existence).

```
lemma ledger_def_prop: ∀ lg:: ledger. ∀ d:: data.
                         lg d = None ∨ (∃! lL. lg d = Some(lL))
```

In order to refine the IoT healthcare application introducing the ledger as a component of the infrastructure state, we re-define the type `igraph`.

```
datatype igraph = Lgraph gra: location × location)set
                         agra: location ⇒ identity set
                         cgra: actor ⇒ (string set × string set)
                         lgra:  ledger
```

The previous database component `lgra` is now replaced by a distributed ledger: the consistency is guaranteed by its global control but the decentralization is given by assigning the data to various locations as recorded in the ledger in a set of locations `L`.

To formally verify that the introduction of the ledger is a refinement, we follow the RR-cycle process and define a refinement map.

```
definition ref_map :: [LedgerRRLoopFour.infrastructure,
                       [RRLoopThree.igraph, location] ⇒ policy set]
                       ⇒ RRLoopThree.infrastructure"
where ref_map I lp = RRLoopThree.Infrastructure
                            (RRLoopThree.Lgraph
                                (gra (graphI I))(agra (graphI I))
                                (cgra (graphI I))
                                (ledger_to_loc (ledgra (graphI I))))
                              lp
```

The `ref_map` takes elements of type `LedgerRRLoopFour.infrastructure` – the refined infrastructure type – using their components to construct abstract infrastructures of type `RRLoopThree.infrastructure`. The names `RRLoopThree` and `LedgerRRLoopFour` are Isabelle theory names and also designate the iterations of the RR-cycle (see Table 3 for an overview). These names are used for name spacing and thus disambiguating the constructors in Isabelle. The second input `lp` is a local policy which is a parameter to the application refinement defined in the application's locale instance. The Isabelle RR-cycle provides the necessary meta-theory to prove this refinement.

Note, the last component in the infrastructure **ledger_to_loc** is the most crucial part of the refinement as it transforms an element of type `ledger` of the refined infrastructure into a function assigning sets of labelled data to locations in the abstract infrastructure type. This constructor is defined as follows.

```
definition ledger_to_loc :: ledger ⇒ location
                         ⇒ (RRLoopThree.dlm × RRLoopThree.data) set
where ledger_to_loc ld l ≡ (if (∃ d. l ∈ snd(the(ld d))) then
                              {(lb,d). l ∈ snd(the(ld d))} else {})
```

The state transition relation in the refined infrastructure specification uses the ledger to implement privacy control by adding corresponding conditions to each inductive rule. For example, the rule for `put` (see Section 3.1), now checks whether the ledger designated the actor's identity `h` as the owner of the data item `d` to authorise that it may be copied to location `l`. This copying is recorded in the ledger by adding (using function update `:=`) the location `l` to the location set `L` of `d` (using the `insert` set operation).

```
put: G = graphI I ⟹ h @_G l ⟹ enables I l (Actor h) put ⟹
     ledgra G d = Some ((h, hs), L) ⟹
     I' = Infrastructure
            (Lgraph (gra G)(agra G)(cgra G)(lgra G)
                    ((ledgra G)(d := Some ((h, hs), insert l L))))
```

```
          (delta I)
⟹ I →n I'
```

Considering now the refined infrastructure model for the IoT healthcare scenario, we reconsider the privacy attack detected in Iteration 2 of the RR-cycle. We can now prove that the previously desired global policy expressing data privacy (Section 3.2) is globally valid: data are labelled uniquely.

```
theorem ledger_guarantees_privacy:
  hc_KripkeF ⊢ AG {x. ∀ d:: data. ∀ d':: data.
                    d = d' ⟶ ledgra(graphI x d) = ledgra(graphI x d')}
```

The initial `hc_KripkeF` is the concrete Kripke structure representing the IoT healthcare example instantiated as a locale. The `ledger_guarantees_privacy` theorem is first proved at the locale level but can be generalized to hold for any model using the ledger as a final step of the RR-cycle. The RR-cycle is thus terminated successfully.

3.4 Discussion

Even though Eve can still put data on the cloud, the refinement introducing the distributed ledger prevents the privacy attack. Because of the ledger, Eve cannot put existing data (that of Bob) with her own label any more. The new `put` semantics only allows extending the ledger by adding new locations and this only for the data owner.

Effective implementations of the ledger need mapping out the globally consistent type `data` to a decentralized datastructure, for example a blockchain with Consensus to preserve the decentralized control specified here. This implementation could be defined within Isabelle and proved to be a refinement of the ledger specification. If this implementation is constructive, code can be extracted from Isabelle into programming languages, like ML, Haskell or Scala.

In comparison to the earlier experiments with applying the RR-cycle to the IoT healthcare case study [14], we could now terminate the RR-cycle process and prove a global (AG) privacy theorem. Moreover, the new type ledger also simplifies the proofs. Compared to the older (partially unfinished) proofs at Iteration 3 and 4 (see [14]) the proofs are shorter. In addition, we do not need to add an explicit precondition to the new put rule that the data item has not been there ([Section IV.C][14]). The new type ledger implicitly guarantees that.

4 Conclusions and related work

In this paper, we summarize some previous preliminary reports [10,14] on using the Isabelle RR-cycle to develop a privacy enhancing architecture for an IoT healthcare scenario. The main contribution is to show how to integrate a distributed ledger into the specification to guarantee global consistency of access control data labeling correcting earlier attempts [14]. Thereby, the RR-cycle can be successfully terminated exhibiting a global privacy property.

Formal system specification refinement has been investigated for some time initially for system refinement in the specification language Z [8] but a dedicated security refinement has not been formalized for some time [15]. The idea to refine a system specification for security has been already addressed in B [2,17]. The former combines the refinement of B with system security policies given in Organisation based Access Control (OrBAC) and presents a generic example of a system development. While B is supported by its own tool Atelier B, it does not provide a formalization in a theorem prover unlike our integration which supports dedicated security concepts like attack trees and enables useful meta-theory over the integration. The paper [17] looks at attacks within the B framework but it aims at designing a monitor that catches actions forbidden by the policy not on using these attacks to refine the system specification. Dynamic risk assessment using attack formalism, like attack graphs, has recently found great attention, e.g. [7]. However, usually, the focus of the process lies on attack generation and response planning while we address the design of secure systems. Rather than incident response, we intend to use early analysis of system specification to provide a development of secure systems. This includes physical infrastructure, like IoT system architecture, as well as organisational policies with actors.

The Workshop Formal Methods on Blockchain (FMBC) [5] has been running yearly from 2019 till 2022 and has produced a number of formalizations also in interactive proof assistants like Coq or Isabelle. From this rich set of related work it is worth mentioning an approach to formalize the FA1.2 Ledger standard in Coq [6]. The goal of this paper has been similar to ours to provide a precise specification of a ledger. The authors look however at a specific technical standard for Tezos as well as using Coq. Moreover, they are not aiming at providing a high level general specification.

There are a number of formalizations of blockchains in interactive proof assistants, for example, Mi-cho-coq [3] and Concert [1] amongst others. Mostly, these formalizations focus on smart contracts to analyze security attacks on cryptocurrencies like Bitcoin. They effectively try to grasp the semantics of the smart contract languages and the effect they have on manipulating cryptocurrency transfers in their models, how to analyze them and extract code into blockchain languages like Tezos. We focus on the global consistency property of a distributed ledger as a security measure to support privacy in IoT applications like healthcare. Smart contracts and cryptocurrencies are a secondary concern.

Other fully automated verification techniques like Modelchecking sometimes also use refinement checking but the refinement of the RR-cycle requires data refinement in addition to trace refinement thus necessitating more expressive logics like Isabelle's HOL.

The presented work aims at showing how the RR-cycle helps identifying and improving on design errors in early phases of the IoT system design. It draws together a number of previous preliminary attempts and finalizes them by applying the closed form of the RR-cycle definition.

The developed specification presents a concise formal definition of a distributed ledger. Although abstract it is well suited to ensure that distributed

data can be handled in a privacy-preserving fashion. The formal specification in Isabelle can also be used to derive more concrete specifications – using refinement – and finally to extract code into programming languages like Haskell or Scala. Future work may address scalability: RR-cycle refinement could be applied to map the current abstract specification to add more detail of actual IoT hardware and Instruction Set Architectures (ISAs). We specify privacy by using data labeling and verifying it by ensuring label consistency across the system. It is interesting to explore how this technique relates to specific notions of data privacy, like k-anonymity or differential privacy.

References

1. D. Annenkov, J. B. Nielsen, and B. Spitters. Concert: a smart contract certification framework in Coq. Certified Programs and Proofs, CPP, ACM, 2020.
2. N. Benaïssa, D. Cansell, and D. Méry. Integration of security policy into system modeling. *B 2007: Formal Specification and Development in B*, pages 232–247, Springer 2007.
3. B. Bernardo, R. Cauderlier, Z. Hu, B. Pesin, and J. Tesson. Mi-cho-coq, a framework for certifying tezos smart contracts. *CoRR*, (abs/1909.08671), 2019.
4. CHIST-ERA. Success: Secure accessibility for the internet of things, 2016. http://www.chistera.eu/projects/success.
5. FMBC. International workshop on formal methods for blockchain, 2019. `https://fmbc.gitlab.io`.
6. J. Gabbay, A. Jakobsson, and K. Sojakova. Money grows on (proof-)trees: The formal fa1.2 ledger standard. *3rd International Workshop on Formal Methods for Blockchains (FMBC 2021)*, pages 2:1–2:14. OASics 2021.
7. G. Gonzalez-Granadillo, S. Dubus, A. Motzek, J. Garcia-Alfaro, E. Alvarez, M. Merialdo, S. Papillon, H. Debar. Dynamic risk management response system to handle cyber threats. *Future Generation Computer Systems*, 83:535–552, 2018.
8. J. He, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. *ESOP*, *LNCS* **214**: 187–196. Springer, 1986.
9. F. Kammüller. Attack trees in isabelle. *20th International Conference on Information and Communications Security, ICICS2018*, *LNCS* **11149**, Springer, 2018.
10. F. Kammüller. Combining secure system design with risk assessment for iot healthcare systems. *Security, Privacy, and Trust in the IoT, SPTIoT'19*. IEEE, 2019.
11. F. Kammüller. Dependability engineering in isabelle, 2021. arxiv preprint, `http://arxiv.org/abs/2112.04374`.
12. F. Kammüller. Isabelle Insider and Infrastructure framework with Kripke structures, CTL, attack trees, security refinement, and IoT example. `https://github.com/flokam/IsabelleAT`, 2022.
13. F. Kammüller, O. Ogunyanwo, and C. Probst. Using fusion/uml for iot architectures for healthcare applications. *arXiv*, https://arxiv.org/abs/1901.02426, 2018.
14. F. Kammüller. A formal development cycle for security engineering in isabelle, 2020. arxiv preprint, `http://arxiv.org/abs/2001.08983`.
15. C. Morgan. The shadow knows: Refinement and security in sequential programs. *Science of Computer Programming*, 74:629–653, 06 2009.
16. A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 1999.
17. N. Stouls and M.-L. Potet. Security policy enforcement through refinement process. *B 2007: Formal Specification and Development in B*, pages 216–231, Springer 2007.