

## SSURF project

SAFETY AND SECURITY UNDER FOCAL

### Identification du projet

*Acronyme* : SSURF

*Thème principal* : Justification de la confiance

*Thèmes secondaires* : Sécurité des systèmes d’information, Sûreté des systèmes informatisés

*Type de projet* : Projet de recherche sur un domaine d’expertise d’une durée de 3 ans

## A Full scientific and technical description of the project

### A.1 Goals

Security of Information Systems is now a well-established field of computer science. The methods used to consider this problem are evolving, as are the ones applied in safety areas, where ad-hoc and empirical approaches are being replaced by formal methods: assurance in system security asks for the use of true formal methods along the process of software development, starting at the specification level. The SSURF project consists in:

- characterizing and studying the required features that an Integrated Development Environment (IDE) must provide in order not only to obtain software systems in conformance with high Evaluation Assurance Levels (EAL-5, 6 and 7), but also to ease the evaluation process according to various standards (e.g. IEC61508, CC, ...), and
- in developing a formal generic framework describing various security properties, e.g. access control policies, together with their implementations using such an IDE.

While the first part aims to elaborate features for an IDE of general purpose, the second one aims to provide a theoretical framework in which security properties and associated implementations can be specified, defined, compared and proved. This theoretical framework will be implemented within the considered IDE, providing a non-trivial illustration of its use. This work will be based on the Focal environment [37, 32, 18] which already provides tools to write specifications, programs, properties and proofs at the same level but which must be tuned to fit security evaluation requirements.

Access control software must be based on a security policy model as software flaws often come from a lack of precision or some incoherences in the policy model. Hence one of the aims of the SSURF project is to introduce an abstract framework allowing to define access control policies, in a very concise way, offering to refine specifications through several levels and ending by different possible implementations. Such a framework must allow to formally reason about security policies and also to compare them. In order to trust in the implementation of such a framework, we need to use an IDE allowing to explicitly deal with the complete security life cycle of a system.

Hence one of the aims of the SSURF project is also to provide a rationale for the conception of such an IDE. To achieve this goal, many questions have to be addressed: methodological guide of development, evaluation/assessment of software, traceability requirements, proof and test techniques, translation into constrained low-level code, static analysis tools over low-level code, vulnerabilities detection and impact analysis, safety analysis of the model used. The SSURF project aims to define an IDE allowing to take into account all these aspects.

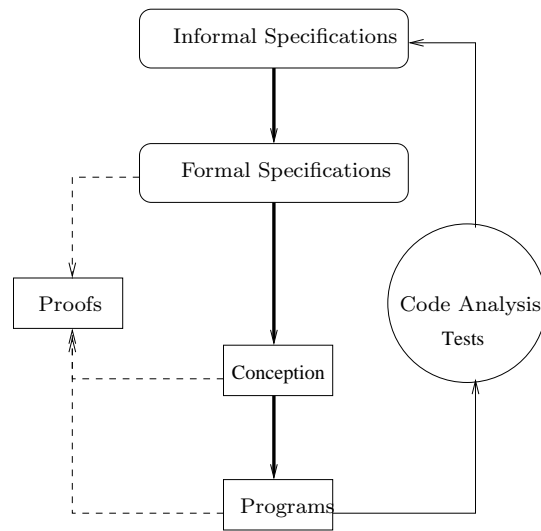


Figure 1: Life cycle

## A.2 Context and Survey

### A.2.1 Motivations

The aim of the SSURF project is to provide a methodology based on a formal IDE capable to develop high integrity software which address all faults and vulnerabilities handling techniques like:

- *Formal proofs of security policies.* This permits to verify the soundness of the specification under consideration; no security lack at specification level
- *Fault avoidance.* This permits to develop a software taking into account potential failures coming from the development process as well as from the hardware supporting the software
- *Fault detection.* This permits to ensure that the developed software has reached its security level by validation testing, vulnerability and safety analysis.

### A.2.2 A brief presentation of Focal

The **Focal** project<sup>1</sup> attempts to build an Integrated Development Environment (IDE) based on firm theoretical results, with clear semantics and which permits an efficient implementation. It provides means for the developers to formally express their specification and to go step by step (in an incremental approach) to design and implementation while proving that such an implementation meets its specification or design requirements. **Focal** offers powerful functional and object-oriented features, such as inheritance and refinement mechanisms, which are particularly well-suited to develop a library for secure applications since they allow to make several refinements of a specification up to the production of an executable code. Thus, **Focal** avoids separation between the formal modeling work and the code development.

In **Focal**, species are the nodes of the hierarchy of structures that makes up the library. A species can be seen as a set of methods, which are identified by their names. Each method can

---

<sup>1</sup><http://focal.inria.fr>

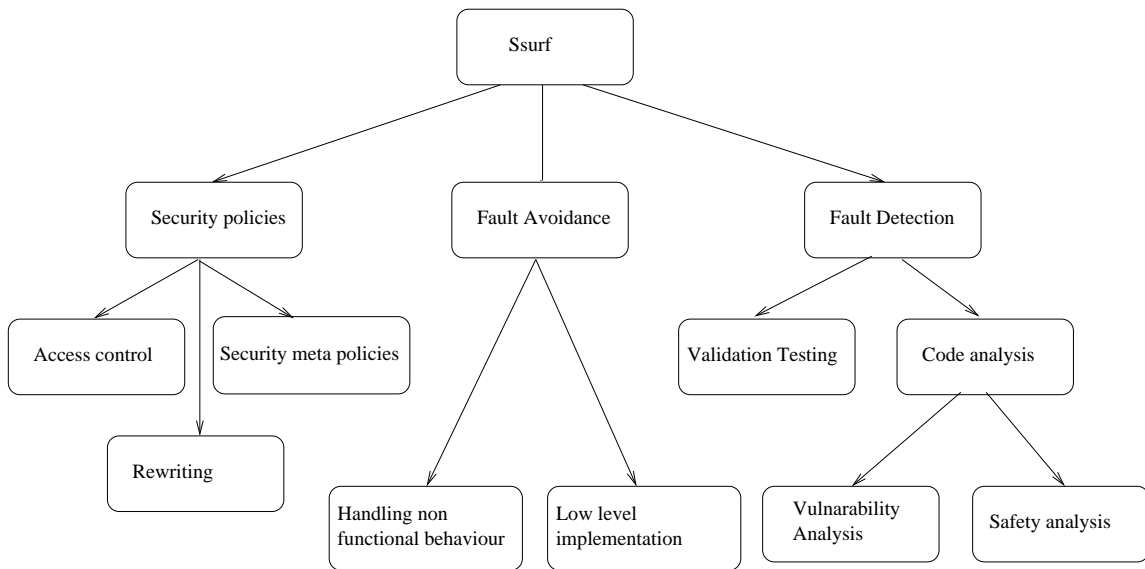


Figure 2: The SSURF project

be either declared (primitive constants, operations and properties) or defined (implementation of operations and proofs of theorems), and belongs to one of the three following types:

(i). The carrier is the method representing the underlying set of the structure defined by the species. The carrier is represented by the keyword `self`, so that we identify the set with the structure, as it is usual. Each species must have one unique carrier, but as all the other methods, it can be either declared or defined. A declared carrier is simply an abstract data type, while a defined one is bound to a concrete type.

(ii). Programming methods represent the constants and the operators of the structure. Declared methods are called signatures. The language of the definitions is similar to the functional core of Ocaml, with the addition of a construction to call a method from a given structure.

(iii). Logical methods are methods which represent the properties of programming methods. In this context, the declaration of such a method is simply the statement of the property, while the definition is a proof of this statement. In the first case, we speak of *properties* (that are still to be proved later in the development), while in the second case we speak of *theorems*. The language used for the statements is composed of the basic logical connectors and universal and existential quantification over a `Focal` type<sup>2</sup>. Currently, a proof in `Focal` consists in either a Coq script interpreted in the context of the species where it is expressed<sup>3</sup> or in a more or less detailed proof expressed with Cutter, a declarative language on which the theorem prover Zenon is based<sup>4</sup>.

The main (object-oriented) features of `Focal` are illustrated on figure 3. Using frameworks involving inheritance requires to perform analysis in order to check coherence properties (inheritance lookup, resolution of multiple-inheritance conflicts, dependency analysis, type-checking ...). In `Focal`, classical object-oriented features have been restricted in order to avoid unsound constructions,

<sup>2</sup>So statements are first-order formulas, containing names of methods bound within the species, directly, by inheritance or parametrization

<sup>3</sup>Some syntactic constructions have been added to tell the `Focal` compiler what are the dependencies of such a proof, so that the compiler can set up the appropriate environment when translating a `Focal` species into Coq.

<sup>4</sup>Zenon allows to easily, and often automatically, obtain formal proofs that are checkable by Coq.

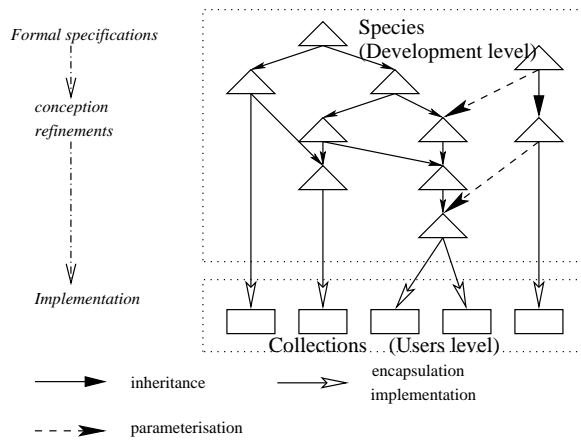


Figure 3: Focal 's features

such as open recursion, which can lead to inconsistencies when used carelessly.

A species can inherit the declarations and definitions of one or several already defined species and is free to define or even redefine an inherited method (but must not change its type). The type of a species is a record, which components are the types of all methods (inherited or introduced in it) of the species. The interface of a species is obtained by abstracting the carrier type in the species type.

A collection is built upon a completely defined species (in which every method is defined). In other words, in a collection, every operation has an implementation, and every theorem is formally proved. In addition, a collection is “frozen”: its “type” is an interface.

Besides inheritance, the other important feature of Focal resides in the possibility to parameterize a species by an interface. An actual parameter may only be a collection, which interface contains the interface of the formal given parameter. This last feature provides a very simple notion of subtyping by inclusion.

Last, note that the efficiency of Focal is demonstrated by its computer algebra library, mostly developed by R. Rioboo [34], which implements mathematical structures up to multivariate polynomial rings and includes complex algorithms with performance comparable to the best computer algebra systems in existence. As we will see, such library is very useful when formalising the algebra of security models (using implementations of lattices and boolean algebras). We can also mention the use of Focal in the previous Action ACI Security Edemoui, where it was very successfully used to specify the security policy of an airport[13].

### A.3 Organisation of the project

In this subsection, we present each theme of the project following the methodological process described above.

#### A.3.1 Formalisation of security policies

One part of this project consists in formalising security properties, especially properties of access control policies. Access control is a mechanism by which a system grants or revokes the rights for active entities, the subjects, to access some passive entities, the objects, or to perform some action.

Such a mechanism is defined through two independent descriptions: what are the entities (subjects, objects, actions, contexts, roles, ...) and what is the applied security policy (confidentiality, integrity, ...).

Access control software must be based on a security policy model as software flaws often come from a lack of precision or some incoherences in the policy model. Hence our aim is to introduce an abstract framework allowing to define access control policies, in a very concise way, offering to refine specifications through several levels and ending by different possible implementations. Such a framework must allow to formally reason about security policies and also to compare them, a point which is rarely approached. In this part, several aspects have to be considered.

## Formal specifications of access control policies and their implementations

*Theme coordinator: UPMC*

Formalising access control policies and their implementations leads to address two questions.

First, building a formal library of access control policies requires to formalise in a precise and unambiguous way the access control policies we want to implement (and to prove). While there exists an extensive literature on access control (HRU, Bell and LaPadula, Chinese Wall, RBAC, OrBAC, TBAC, TMAC, CBAC, ...), describing such policies and explaining how a particular access control works, most of the papers are rather informal and/or deal with a particular access control mechanism through examples without any formalisation (or generalisation) of the concepts involved in the policy, nor real help on how to implement it in a different context.

Furthermore, even for papers containing specifications, definitions and properties expressed in a mathematical way, such formalisations are not always done at the level of detail required to obtain a “computer-assisted formalisation”. When attempting to check proofs done by hand with a proof assistant, many of them have been discarded. Often, the errors are introduced by points considered as evident details or by forgotten cases. Even if having a mathematical model drawn by hand is a very serious way to increase confidence, this is not enough: nothing is said about how these abstract notions can be implemented. Last, even if proofs done by hand are correct, nothing formally ensures that the implementation meets the “formal” specification done “on the paper”. To conclude, such a work also allows to fill the gap between (in)formal papers and (formal) implementations and can be considered as the first step of a formal implementation.

The second point we have to address is concerned with the definition of a generic, abstract framework allowing to deal with various types of access control policies. Indeed, many access control policies share some common definitions, properties and so proofs, and it seems worth to build an abstract generic framework in order to ease and speed implementations by reusing. Ideally, suiting an implementation to a given context should only request the instantiation of some parameters. In previous works [21, 25, 26, 30], we have used the Coq proof assistant and the Focal IDE to provide a formal development of both the Bell and LaPadula model and the “algebra of security” introduced by J. Mac Lean [29]. Such a development has been used to implement a safe reference monitor in a relational database [5], but is not completely satisfactory as it is difficult to reuse to implement a Chinese Wall security policy, without starting from scratch.

To do so we need a formal framework in which both what is an access control policy and what is the interpretation of a security policy by another one (and what properties such an interpretation must satisfy) can be specified and defined. Unfortunately, the framework of the “algebra of security” is not expressive enough to achieve this goal and we need to develop a more appropriate framework for specification and implementation of access control policies

## Security Meta-policies

The purpose of the theme is to define formally and to study a non standard access control policy model as defined in [4, 3]. The specificity of this model is that the access control policy is *dynamic* and *itself obeys a (static) security policy*, which means that some subjects of the system have the capability (the **permission**) to modify access rules between some other subjects and objects.

Such a dynamic is motivated by the need to ensure security properties on a distributed system using *local* access control policy on each node of the system. Distributed systems have to be tolerant to changes occurring to the other nodes during execution, including faults, new distributed software installed on a node... For the local access control policy to be able to deal with such events, it must allow dynamic changes to permission as well as *creations* or suppression of subjects and objects.

The idea of [4, 3] is to define a fixed security *meta-policy*, installed on every node of the system, which allows some access permissions to be modified. What is currently conjectured is that if the meta-permissions are well chosen, then some information flow properties can be ensured locally and globally. The typical example of such properties is that no information flow can leak from a subject  $s$  to an object  $o$ . One interesting aspect here is that  $s$  and  $o$  could be on different nodes of the distributed system.

In such a system, reachability properties are undecidable in general (not yet published results). In particular if an agent can create new objects which may have meta-permissions, reachability is undecidable.

The purposes of our work are the following:

- Give a formal definition of such a framework using or adapting the formal abstract framework described in the above paragraph.
- Prove non reachability properties on some realistic examples using **Focal** ,
- Define restrictions on the meta-policy for which reachability properties are decidable, and prove the decidability in **Focal** .

## Security and Rewriting

Theme coordinator: LORIA

Term rewriting provides well-established techniques and tools that are being successfully applied to reason about many aspects of computer security, notably in verification of security protocols [9, 38]. In order to use similar term rewriting techniques for the investigation of access control related problems, the security policies should be modeled as term rewriting systems and their properties analyzed accordingly.

Provided that the flexibility of the policy specification is a key issue, such formalizations must allow capturing the dynamic aspects important for policy enforcement, e.g. diverse attributes of subjects and resources and to use these attributes to state which actions the first can perform over the second.

Our goal is to formalize access control policies representing the environment where policies are enforced as algebraic terms, forming a base of facts which is updated as the target application runs. Therefore, a security policy in this model is a term rewriting system, defined over the signature used to represent the environment, that defines a normalization process that leads to authorization decisions taking into account the current state of the target application, and the current access request.

Some of the expected results of such a formalization include the analysis that can be performed over these systems with respect to the combined use of positive and negative authorizations, which brings two main problems: incompleteness, when no decision is specified for a certain request, and inconsistency, when for an access there are both negative and positive authorizations.

Completeness is achieved by assuming that one of either the open or closed policy operates as a default. Conflict resolution is a more complex issue where different decision criteria can be adopted, each applicable in specific situations, corresponding to different policies that can be implemented. In this direction, we may investigate how to solve conflicts using rewriting strategies.

Another issue that will be covered is the composition of access control policies. We must check the impact to the system when joining different policies and whether the security properties hold under composition. This can be done by relating the existing results on modular properties of term rewriting, like confluence, for example.

## Security components

*Theme coordinator: UPMC*

A more general aim of the SSURF project about formalisation consists in defining a hierarchical typology of security functions with the formalisation of their associated security properties, both functional and non-functional (e.g. filtering, to be further refined into firewall, contextual firewall, and so on...). There are several rationale to justify this task:

- this will constitute a set of test cases to assess the expressiveness of the Focal language regarding security properties, as well as the inheritance of such properties through composition mechanisms,
- the identification of generic elementary security functions will provide a framework for a standardised description of real security equipments (a commercial firewall being the association of filtering, transformation, and so on),
- formal profiles, mimicking the so-called Protection Profiles (PP) of the Common Criteria, will be proposed, allowing refinement with proved traceability into High Level Description (HLD), up to Low Level Description (LLD) or even prototype implementation
- by composition, complex systems (perhaps even an entire network) will be modelisable, allowing formal analysis of their properties. For this last application, it will also possible, reusing security policy generic formalisation, to assess wether or not a given system complies with a security policy.

### A.3.2 Handling non fonctionnal properties in Focal

*Theme coordinator: UPMC*

Formal development tools generating source code are focus on soundness of the produced source code in comparison with the related formal specification. From a safety point of view, the produced source code is not always as robust as it should be, due to many causes, the main ones being that:

- the model does not handle the dysfunctional behaviors of systems interfacing with the software
- types in the formal framework have not exactly the same meaning than their implementation. For example: in a formal tool, a sum type is characterized by different items, yet at the physical level, each item of the same sum type is represented by a machine word that is bigger than the item itself.

- developer has only correctness proof obligation of functional behavior to discharge.

Therefore, failure situations may result into execution states which were not addressed by the formal analysis.

This work will identify how to formally characterize failures and to automatically generate associated proof obligations in the Focal project addressing such failure occurrences.

### A.3.3 Implementation: from high-level Focal code to constrained low-level code

*Theme coordinator: BERTIN Technology*

Currently, the Focal compiler produces Ocaml program. Our aim is to produce code for low level systems such as OS. Hence, Focal must be extended with a generator of C code, in conformance with CC related rules such as those defined in [11]. Designing a compiler from Focal to C is a rather difficult task but cannot be avoided for industrial projects. Indeed, the compilation process will need to solve some advanced compilation problems:

- preserving, as far as possible, the structure of Focal code through the species, collections, parameterisation, multiple inheritance and other specific features of this language,
- compiling fonctionnal code to imperative code
- managing to preserve the garbage collecting feature: the target language of programs developed within Focal is OCaml, providing a modern garbage collector, so changing the target language may create a new problem of memory management.

Furthermore, the C generated code must follow guidelines defined in [11] which forbids some constructions that could be unsafe (OO multiple inheriting is prohibited, multiple end-points in one function deep successive pointers indirections, ...) Of course, all the restriction rules do not have the same impact on security, and they can be classified into a security relevance hierarchy.

### A.3.4 Tests and Validation

*Theme coordinator: CEDRIC*

Even if the Focal environment ensures a high level of confidence as a result of its methodology based on specification and proof, testing is still required for the following reasons:

- Testing allows to find bugs.
- Formal verification ensure consistency between the implementation and its specification; however the specification may not reflect the true requirements.
- Some basic types in Focal may have a formal representation that differs from their implementation (e.g. `int` is translated in the Ocaml type `int` and the Coq type `Z`, so the machine integers are used in the code but proofs are done with inductively defined integers). So we have *some* confidence in the code but we must test code to verify if the properties are verified, in particular around the bounds.
- Some of the properties may not be proven, for example low level properties about the addition of machine integers (we trust it because of external formalizations) or very general mathematical properties. In that case, these properties are assumed to be true (the keyword `assumed` is used instead of giving a sketch of proof). Such assumed properties automatically become test objectives.

- Some Ocaml untrusted code may be imported in a Focal certified code and should be tested.
- When Zenon does not succeed in proving a property automatically, two issues are possible: the property is not true or Zenon needs to be helped by giving some intermediate lemmas the user has to find. So before embarking on a proof, it can be useful to test a not yet proven property in order to discover a counter-example or to have more confidence. Such testing tools have been integrated in the proof assistants Isabelle [2] or Agda [20].
- Each Evaluation Assurance level, including EAL-5, 6 and 7, requires a validation phase with testing.

We propose to define a framework and a few tools allowing to take into account test cases proposed by the user and also to generate test cases from Focal components.

Focal is a functional language with object oriented features. Little effort has been done regarding testing functional programs (wrt the litterature around imperative or reactive languages). Currently QuickCheck [10] is a success tool for testing Haskell programs. It has been adapted to other languages e.g. Erlang, Scheme or Ocaml [27]. Widera [39] has defined in a functional setting the notions of control and flow graphs.

We propose to test a property, thus our technique will range over the category of specification based testing. It means that we want to *execute* this property on some data carefully chosen, called test cases. This requires that all the functions appearing in the property must be defined as well as the representation. A first task will be to syntactically characterize the executable properties.

A property usually contains some hypothesis and a conclusion. If not, we can transform it into a set of such elementary properties. Roughly speaking, testing a property consists in choosing values for the variables of the property and then evaluating it. More precisely we have to select values that satisfy the hypothesis and then evaluate the postcondition. If the result is true, then the test case is a success otherwise it is a failure. So the property serves as an oracle.

Values that satisfy the preconditions are not so easy to find. A first method can be to determine such values randomly but they are likely not to satisfy the preconditions. We can repeat the random draw until convenient values are produced but it will be an expensive process. Another method consists in exploring very carefully the hypothesis and more precisely the definition of the involved functions in order to produce constraints upon the values of the variables. Then it remains to instantiate the constraints in order to generate test cases ready to be submitted. This method is a *white box* method testing whereas the first method is a *black box* testing method.

The objectives of the corresponding task is to develop both methods. The first one has been studied for simple species (without parameters) and simple properties whose syntax grammar covers 90% of the properties found in the computer algebra library [8]. Both methods will require investigating the derivation of automatic data generators from datatype definitions in a canonical way.

### A.3.5 Security and Safety Analysis

#### Vulnerability Analysis

*Theme coordinator: Safe River*

Safety and security software must comply with assurance requirements, stated in standards. Both for safety oriented recommendations and standards (IEC 61508, DO178B, EN50128, IEC880) or security oriented (Common Criteria), it is mandatory to perform validation and verification tasks and activities.

Validation and verification of safety are performed on the basis of :

- Failure mode effects and criticality analysis, performed at the system level,
- Software errors and effects analysis,
- Code analysis,
- Code review and auditing,
- Robustness and safety requirements testing. Test scenarios and testing activities must be "independent" for high integrity levels.

Validation and verification of security properties are based on :

- the fact that the TSF is coherent and complete, in the sense that it describes exactly the policy model,
- the traceability proofs throughout the whole life cycle, and especially on the proof that the implementation is compliant with the specification,
- testing activities,
- vulnerability analysis.

**Vulnerability Analysis** In Common Criteria methodology, vulnerability analysis [AVA\_VLA] must be performed by the developer to ascertain the presence of vulnerabilities and to prove either that they cannot be exploited. The rigour of vulnerability analysis increases in coherence with the assurance level. Basic activities of vulnerability analysis are the following :

- the developer must identify and document the ways an attacker could violate the security policy,
- the developer must identify vulnerabilities,
- the developer must document the vulnerabilities which have been detected
- the developer must determine whether the vulnerabilities can be exploited or not, taking into account the threat level.

The objective of our work is to study and specify useful tools in order to help in AVA\_VLA activities and part of AVA\_SOF activities.

**Identification of vulnerabilities, based on a deductive approach (Attacks trees)** This analysis intends to identify the means and ways that an attacker may use in order to violate the security policy. Inputs of the analysis are the specification and the design documents (High Level Design, Low Level Design) if available. The model which describes the security properties (it should be the formal model in Focal) must be analysed in order to identify attack modes which can be done: attacks against password management functions, attacks when using cryptographic services and functions, when using sensitive resources,... Attack scenarios may be described using a fault tree formalism (Attack trees) in order to identify critical modules and resources which could be targeted by attackers. This analysis will be used in order to document the ways an attacker could

violate the security policy. It is similar to Sw fault tree analysis (SwFTA). The root of the attack tree is an event. Events, representing the attacks an attacker will use, are organised in a "and/or" tree. The complete fault tree permits to describe the attackers' strategy. The objective of the study is twofold :

- exploitation of the formal method in order to determine the root events (attacks classes),
- Fault tree analysis methodology.

**Inductive approach : Vulnerabilities detection and impact analysis** This activity is based on a source code analysis. Source code analysers are useful for detection of software bugs which can be exploited as flaws or which can be propagated as flaws, until they have impacts on the behaviour of the system. A very famous example is the overflow, which may be the starting point of many flaws and vulnerabilities, with impacts both on safety and security.

Analysis tools detect particular classes of errors, for instance :

- execution errors (static analysis tools)
- hazardous statements (syntactical analysis tools such as RATS, ITS4, flawfinder, coverity,...)
- hazardous design in specific domains such as password management, cryptography.

Tools are useful to detect and localize potential flaws, but the auditor has to determine whether flaws and vulnerabilities may be exploited, and to check whether a function is used properly or in a hazardous way.

The study will be organized in three subtasks :

- Errors and Vulnerabilities modeling and formalisation. The classification and the formalisation of coding errors or software problems which can lead to vulnerabilities are mandatory in order to identify detection techniques which can be at least partially automatized. We shall use the model which has been proposed by NIST (cwe-classification tree). This model provides a taxonomy for software security flaws and vulnerabilities. The objective of our study is to refine this model, in order to state which flaws or which security problems can be detected by code analysis, and to state which kind of analysis is appropriate.
- Identification of the conditions in which vulnerabilities may be exploited. Flaws and security problems may lead to vulnerabilities, but it is the responsibility of the auditor to evaluate whether they will do actually. The concern of this subtask is to determine and formalise conditions which are necessary to exploit a flaw or to propagate an error (API abuse, time and race conditions, for instance) and protections which can mitigate a flaw or an error effect. The formalisation should permit to determine whether it is possible to have automatized detection techniques of the exploitation and propagation conditions.
- Impact analysis Methodology. We shall study whether it is possible to define a systematic Sw Vulnerabilities Analysis, which will permit to document vulnerabilities analysis and to determine whether an attacker can exploit a detected vulnerability.

Section A.3.2 deals with improvement of software robustness during the specification and software design. This current phase is focus on how to perform safety analysis in order to *measure* robustness of the produced software. Safety standards recommend to perform this measure by using methods like SwFMECA (Software Failure Mode Effect and Criticality Analysis) [31] or SwFTA (Software Fault Tree Analysis).

**Sw Failure Mode Effect and Criticality Analysis** is a method for identifying effects of failure modes through the software. Objective of an SwFMECA is to identify all modes of failure within a system/software design, and to find potential catastrophic and/or critical effects related to these failures. SwFMECA should be initiated as soon as a preliminary dataflow model is available at the higher software level and extended to the lower levels as more information becomes available. Even if the software is developed using a formal method, failure modes are still existing in software, like failures in hardware supporting the software (RAM, CPU, ...).

**Sw Fault Tree Analysis** is a method for identifying and documenting the combinations of lower-level software/hardware events that allow a top-level event (or root node) to occur. Events, representing failure modes, are organised in a And/Or tree. Starting with a root node representing a hazard, the SwFTA permits to describe the known ways and the combinations (called minimal cuts) in which the software can reach that particular unsafe state.

**Functional model.** These safety analyses are based on a functional model of the system/software to analyse. The minimal requirement for this model is to be able to identify for each data the last software component producing the data and the components consuming it. We can see that dataflow model is a perfect candidate as a fonctionnal model.

We propose during this phase to identify how to transpose these software safety analyses into the Focal project. This will permit to identify the needed features to carry out safety analysis on Focal models. First step will be to define failure modes dedicated to the Focal theoretical foundations and the Focal development process. Then, we will provide a way to elaborate a functional model from Focal species. Next steps will consist in applying a usual safety analysis on the model.

These analysrs should be achievable at each level of the development cycle.

### A.3.6 Methodology of development within Focal

**Objectives.** One of the main characteristics of critical software is that it is subject to the approval of a safety authority before its commissioning. Since more than 10 years, safety authorities have defined sets of requirements describing what should be an acceptable software and its related life cycle process for their own domain. These requirements are described in safety standards like Common Criteria, IEC 61508 for safety related systems, Cenelec 50128 for railways, Do178B for aeronautics, ... That's why, each tool that aims to be used in safety system development should be incorporated into a life cycle process.

**Focal life cycle.** We propose during this phase to elaborate an optimized life cycle *around* the Focal tool. This life cycle will address all phase of the development from the specification to the validation testing as well as the support activities like modifications management, non regression testing, traceability, safety analysis, . . .

We start by choosing the appropriate life cycle model for Focal development (classical V model, waterfall model, spiral model . . .). Then, a *generic* life cycle will be build from the chosen model, defining precisely each inputs, outputs, processing and stopping criteria for all the life cycle phases. This will aim to specify new features for the Focal tool in order to meet requirements needed by a complete and consistent life cycle.

This generic life cycle will be mapped to each domain specific standard.

**Documentation generation from Focal specification.** A mandatory feature for a specification tool is its capability to generate a usable documentation for the customer, the verifier and the assessor. Documentation tool should be able to trace all requirements and safety data from the specification to the implementation level.

## B Appendice

### B.1 UPMC

**Philippe Ayrault** [100%], System and Software Safety Expert since more than 10 years, was in charge of a safety team performing verication, validation or assessment of system or software in various domains like railway, energy, aeronautic or nuclear. He's approved by the french transport Department as a Technical Independant Body for public transport safety assessment as well as by the CERTIFER Agency. He participated in writing of the ECSS Q80-3 Software Product Assurance Standard for the European Space Agency. At present, he's carrying out a PhD thesis under T. Hardin and R. Rioboo's supervision, in partial time.

**Thérèse Hardin** [30%], is Professor at Pierre and Marie Curie University. She heads the SPI team (Semantics, Proofs and Implementations) of the LIP6 lab. T. Hardin research activities were first concerned by the study of the rewriting system of Categorical Combinators of Curien[22], then by the study of explicit substitution calculi[12]. These works led her to give a presentation of higher-order unification using only first-order features, in collaboration with G. Dowek and C. Kirchner[16]. This collaboration was pursued by the introduction of a new presentation of first-order logic, the Sequent Calculus Modulo, and by the study of automatic theorem proving by resolution modulo[15]. In collaboration with R. Rioboo, she launched the research project on the design and the development of Focal and she is currently the scientific coordinator of the research activities conducted within Focal framework[7, 6, 14, 18, 23]. She supervised S. Boulmé Ph. D. (co-supervisor R. Rioboo) on the specification of Focal in Coq, V. Prévosto Ph. D. (co-supervisor D. Doligez) on the Focal compiler, S. Fechter Ph. D. (co-supervisor C. Dubois) on Focal operational semantics. The research interests of T. Hardin also concern the use of formal methods to help security and safety engineering and evaluation of critical systems. Focal design was partly done according to her acquired experience via some research works together with companies involved in these subjects[1]. Research activities on security policies are conducted by SPI members, M. Jaume, C. Morisset, J. Blond. She currently supervises P. Ayrault Ph. D., on the definition of methods easing the evaluation of

the conformance of critical systems to standards (Common Criteria, IEC61508, EN 50127-8-9, etc. ). And Eric Jaeger is starting a doctoral work on the formalisation of security components like firewalls under her supervision.

**Eric Jaeger** [100%], engineer with experience in defence telecommunication and information systems, is working as a security expert at the Direction Centrale de la Sécurité des Systèmes d'Information, the French CIS security authority. His job position involves research in the field of the application of formal methods to security. He is also a part-time PhD student under the supervision of T. Hardin and M. Jaume.

**Mathieu Jaume** [100%], is assistant professor at the University Pierre and Marie Curie in Paris. Its former research activities mostly concerned formal methods in various contexts: semantics of logic programming languages, teaching, proof management [17, 33]. His research interests now concentrate on formalisation of security properties such as access control policies [21, 25, 26] and their description within the Focal IDE.

**Charles Morisset** [100%], is a PhD student under M. Jaume's supervision, at the University Pierre and Marie Curie in Paris, France. Its research interests include formalisation and modelisation of access control security policies and formal methods.

**Renaud Rioboo** [34, 24, 36, 35] [20%], is assistant professor at Université Pierre and Marie Curie. He defended his habilitation in december 2002. Renaud Rioboo is working on the Focal project, he was one of the main designers of the language and wrote the standard Focal library which implements concepts from effective mathematics. He is now working on the compiler and on tools for the Focal language. In particular he is working on the automatic documentation tools and XML related tools for Focal . Renaud Rioboo partipates to the supervision of all Ph. D. students working for Focal .

## B.2 CEDRIC

**Matthieu Carlier** [100%] , is a Phd student. He began his thesis in october 2005 with a grant from the ministry of education under the supervision of Catherine Dubois. The subject of his thesis concerns the introduction of testing techniques into Focal.

**Pierre Courtieu** [20%] , is assistant professor in the Conservatoire des Arts et Métiers in Paris. Its research activity, developed at the CEDRIC, include proof systems (Coq) and proof automation, program verification, and security properties.

**Catherine Dubois** [19, 18, 28] [20%] is Professor at the Institut d'Informatique d'Entreprise (ENSIIE very soon) since 2000. She has a PhD from Conservatoire National des Arts et Métiers.

She is a member of the laboratory CEDRIC and heads the CPR team. Her main research interests concern typed programming languages and formal proof, more generally the use of formal methods to develop safe applications. For example she has formalized within Coq a large subset of ML: e.g. semantics, type inference. She has also experimented B in the field of geometric modelling. Since several years, she has been involved in the development of the Focal environment, in particular about the definition of the formal semantics (she was the co-supervisor of S. Fechter's Phd with T. Hardin) and the introduction of testing techniques into Focal (she is the supervisor of M. Carlier's thesis).

She has participated to the projects Edemoui and Modulogic (ACI Scurit Informatique) and is currently involved in the projects REVE (projet ARA SSIA) and A3PAT (projet blanc).

### B.3 LORIA

**Horatio Cirstea** [40%], is Assistant Professor at the Nancy 2 University and member of the Loria laboratory in the Protheo team. His main research interests include the theoretical foundations and practical applications of rewriting. He introduced together with Claude Kirchner a new formalism called the rewriting calculus and has been studying in collaboration with several scientists the properties, the extensions and the applications of this calculus. He has also used the rewrite based languages developed in the PROTHEO team (ELAN and TOM) for the description and verification of various transition systems (like, for example, cryptographic protocols). After a master in computer science at the University Politehnica of Bucharest and an one year project at the Oxford University on the use of CSP on the analysis of network protocols, he obtained his PhD at the University Henri Poincaré in 2000. He continued with an industrial post-doctoral stay at Platform Computing and obtained an assistant professor position in 2002. He's been participating to the MANIFICO (non-intrusive meta-compilation of matching with constraints) RNTL project, to the MODULOGIC ACI project and to the REWERSE (Reasoning on the Web with Rules and Semantics) Network of Excellence (NoE) within the 6th Framework Programme.

**Anderson Santana de Oliveira** [50%], is PhD student at Henri Poincaré (Nancy 1) since 2004 under the supervision of Claude Kirchner and H  l  ne Kirchner, in the PROTHEO project. He has accomplished a B.S. in Computer Science in 2001 in the Federal University of Rio Grande do Norte (UFRN), in Brazil, and obtained a M.S. in Computer Science in the same university in 2004, which comprised a four month research internship in the PROTHEO team. He has been teaching assistant during one semester on compiler construction at UFRN. In his thesis he is interested in composition and modularity of formal systems, specially term rewriting systems and security policies.

### B.4 Bertin Technologies

Bertin Technologies is a French technology services and engineering company with a strategic activity in software security. As such, the company has identified the key need for more formal software engineering tools incorporating the different aspects of software security: code analysis, security-oriented code reverse engineering, inclusion of security requirements deriving for the Common Criteria for software security. Bertin Technologies is heavily involved in the French initiative on the development of secure operating systems.

**Julien Blond** is a Ph. D. student under CIFRE convention between Bertin and UPMC, supervised by R. Rioboo and M. Jaume. His research subject concerns the use of formal methods within the SINAPSE access control implementation. Using a Coq proof developed by Emmanuel Gureghian and Mathieu Jaume, he has developed an OCaml RDBMS front-end granting a multi-level access control implementation.

**Emmanuel Gureghian** is Bertin System Security Officer and manages the Information System Security Department. He has designed the architecture of the SINAPSE secure OS and is in charge of the supervision of the ISS R&D of Bertin including the SERIOUS ITEA project, the TSC MEDEA proposition, the PFC synaptic.

## B.5 Safe River

SafeRiver has been created by two experts in Safety and Security software. Their industrial projet is to propose a set of methods and tools for performing validation, verification and assessment of existing pieces of software, based on code analysis and audits. These tools are the basis of consultancy services, commercialized by SafeRiver. SafeRiver founders have been involved in the validation and verification of critical systems (underground transportation, automotive, railway, aeronautic) from 1994. They also have a strong experience in the development of secure software: (PKI applications, secure e-government applications, health applications). Two key issues motivated SafeRiver project: on one hand, more and more critical systems embed existing software, so that risk assessment and proof of assurance must be done without mastering the whole life cycle documentation. On the other hand safety systems which were tampered are now interacting with various systems. Thus, it is crucial to deal both with errors and security flaws, and to take into account vulnerabilities in risk assessment methodologies. SafeRiver has obtained a label "Projet en emergence" in 2005.

**Véronique Delebarre** [30%] (PHD thesis in computer dependability evaluation) worked at Verilog, a software engineering company, where she was responsible of consultancy services from 1988 to 1993. She developped validation and verification packaged services, based on model checking (SDL, Estelle, Lustre were the sublying languages) for safety critical systems. In 1994, she joined CESIR a start up, and became the manager of this young company which has developped a cryptography tool box, and PKI applications (authentication proxy and PKI components). CESIR was managed by Véronique Delebarre from 1995 to 2000 and has been acquired by Communication and Systems (CS) at the end of 2000. From 2001 to 2005, she worked at CS in the development of various secure systems, especially in the area of electronic signature (for the french notaries) and e-government applications.

**Eric Juppeaux** [100%] is currently co-founder and R&D manager of SafeRiver. He is graduate of master of science in mathematics and obtained his DEA (SPP) in 1996. He joined CESIR in 1996. In 1997 he developped a "Constants verification tool", which has been used by RATP in order to check exhaustively the safety properties and constraints that must be satisfied by the embedded constants (METEOR project). This tool is based on a "constraint description language" and a verification engine. Also, Eric Juppeaux has been the project manager of the "Security Tool Box" which has been developped by CESIR and afterwards by CS. This Security Tool box, composed of cryptographic components and protocols, has been the basis for developping PKI products (authentication and signature in particular) which have been deployed for french notaries.

## References

- [1] P. Ayrault, M. Guesdon, and T. Hardin. Mthodologie de dveloppement d'un outil d'valuation de la sret du logiciel, en langage ocaml. In *JFLA2000*, pages 159–171, 78153, Le Chesnay Cedex, France, 2000. INRIA.
- [2] S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In J. Cuellar and Z. Liu, editors, *Software Engineering and Formal Methods (SEFM 2004)*, pages 230–239. IEEE Computer Society, 2004.
- [3] M. Blanc, P. Clemente, P. Courtieu, S. Franche, L. Oudot, C. Toinard, and L. Vessiller. Hardening large-scale networks security through a meta-policy framework. In *Third Workshop*

on the Internet, *Telecommunications and Signal Processing (WITSP.04)*, Adelaide, Australia, dec 2004.

- [4] M. Blanc, P. Courtieu, G. Hains, L. Oudot, and C. Toinard. A novel approach for distributed updates of MAC policies using a meta-protection framework. In *Supplementary Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE 2004)*, Saint Malo, France, nov 2004.
- [5] J. Blond and C. Morisset. Formalisation et implantation d'une politique de sécurité d'une base de données. In INRIA, editor, *17ème Journées Francophones des Langages Applicatifs, JFLA'2006*, pages 71–86, 2006.
- [6] S. Boulmé, T. Hardin, V. Mniissier-Morain, and R. Rioboo. On the way to certify computer algebra systems. In *Calcuemus 99: Systems for Integrated Computation and Deduction*, volume 23, Trento, Italy, 1999. Elsevier.
- [7] S. Boulmé, T. Hardin, and R. Rioboo. Modules, objets et calcul formel. In *JFLA99*, pages 171–188, 78153, Le Chesnay Cedex, France, 1999.
- [8] M. Carlier. Introduction de techniques de test dans focal. Master's thesis, University Paris 6, 2005.
- [9] H. Cirstea. Specifying authentication protocols using rewriting and strategies. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2001.
- [10] K. Claessen and J. Hughes. Specification based testing with QuickCheck. In *The Fun of Programming*, Cornerstones of Computing, pages 17–40. Palgrave, 2003.
- [11] SINAPSE Consortium. Sinapse plan d'assurance qualité logiciel, 2005.
- [12] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.
- [13] D. Delahaye, J-F. Etienne, and V. Vigié-Donzeau Gouge. Certifying airport securitu regulations usinf the focal environment. In *Formal Methods 06 (FM06)*, 2006.
- [14] Damien Doligez, Thérèse Hardin, and Virgile Prevosto. Algebraic structures and dependent records. In Butler Rick, editor, *Proceedings of TPHOL'03*, NASA, Hampton, USA, 2002.
- [15] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.
- [16] G. Dowek, Th. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.
- [17] C. Dubois, J. Grandguillot, and M. Jaume. Réutilisation de preuves formelles : Une étude pour le système FoC. In INRIA, editor, *14ème Journées Francophones des Langages Applicatifs, JFLA'2003*, pages 63–75, 2003.
- [18] C. Dubois, T. Hardin, and V. Vigié Donzeau Gouge. Building certified components within Focal. In *Symposium on Trends in Functional Programming, TFP'04*, 2004.

- [19] C. Dubois and J.M. Mota. A formally verified geometric modelling core. In *International Conference on Software Engineering Research and Practice*, 2006.
- [20] P. Dybjer, . Haiyan, and M. Takeyama. Combining testing and proving in dependent type theory. In D. Basin and B. Wolff, editors, *Proceedings of Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 188–203. Springer-Verlag, 2003.
- [21] E. Gureghian, Th. Hardin, and M. Jaume. A full formalisation of the Bell and Lapadula security model. Technical Report 2003-007, Univ. Paris 6, LIP6, 2003.
- [22] T. Hardin. Confluence results for the pure strong categorical logic ccl.  $\lambda$ -calculi as subsystems of ccl. *Journal of Theoretical Computer Science*, 65:291–342, 1989.
- [23] T. Hardin and R. Rioboo. Les objets des mathématiques. *L’Objet, Hermès Sciences*, 10(4):83–119, 2004.
- [24] Thérèse Hardin and Renaud Rioboo. Les objets des mathématiques. *RSTI - L’objet*, Octobre 2004.
- [25] M. Jaume and C. Morisset. Formalisation and implementation of access control models. In *Information Assurance and Security (IAS’05) International Conference on Information Technology, ITCC*, pages 703–708. IEEE CS Press, 2005.
- [26] M. Jaume and C. Morisset. A formal approach to implement access control model. *Journal of Information assurance and security*, 2:59–70, To appear. 2006.
- [27] F. Keth. Ocamltest, 2005.
- [28] R. Laleau, S. Vignes, Y. Ledru, M. Lemoine, D. Bert, V. Vigié Donzeau-Gouge, C. Dubois, and F. Peureux. Adopting a situational requirements engineering approach for the analysis of civil aviation security standards. *Software Process: Improvements and Practice, special issue SREP*, 2006.
- [29] McLean. The algebra of security. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–7. IEEE Computer Society Press, 1988.
- [30] C. Morisset. Formalisation et implantation d’un modèle de contrôle d’accès dans l’atelier Focal. Master’s thesis, Université Paris 6, 2004.
- [31] Department of Defense. *Procedures for performing a Failure Mode, Effects, and Criticality Analysis*. Number MIL-STD-1629A. 1998/11/24.
- [32] V. Prevosto and D. Doligez. Algorithms and proof inheritance in the Foc language. *Journal of Automated Reasoning*, 29(3-4):337–363, dec 2002.
- [33] V. Prevosto and M. Jaume. Making proofs in a hierarchy of mathematical structures. In *11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Calculemus 2003*, pages 89–100. Aracne, 2003.
- [34] R. Rioboo. *Programmer le Calcul Formel, des Algorithmes à la Sémantique*. Mémoire d’habilitation, Université Pierre et Marie Curie, Paris, France, 2002.

- [35] R. Rioboo. Certifying computer algebras systems: Data safety. EACA, Issac 2004 Statelite Conference, Santander Spain, July 2004. Invited Talk.
- [36] R. Rioboo. Concrete mathematics with the FoCaL environment. Formal Methods 2005, Calculemus Workshop, Newcastle, July 2005. Invited Talk.
- [37] R. Rioboo, D. Doligez, V. Prevosto, M. Jaume, M. Maarek, C. Dubois, S. Fechter, V. Ménissier-Morain, O. Pons, D. Delahaye, V. Viguié, and T. Hardin. *Focal, version 0.2 Tutorial and reference manual*. LIP6 – INRIA – CNAM, sept 2004. Distribution available at: <http://focal.inria.fr>.
- [38] M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental abr conformance algorithm. *J. Autom. Reasoning*, 30(2):53–177, 2003.
- [39] M. Widera. Flow graphs for testing sequential erlang programs. In *ERLANG '04: Proceedings of the 2004 ACM SIGPLAN workshop on Erlang*, pages 48–53, New York, NY, USA, 2004. ACM Press.