The Messir Scientific Approach to Requirements Engineering

Nicolas Guelfi

Received: date / Accepted: date

Abstract A large amount of theories, methods and tools have been produced by the software engineering community but one of the main sources of complexity is their consistent integration that would make them adequate to be used pragmatically and just in time for actual engineering problems. In the last five years, such an integrated approach has been developed, called Messir and experiments have been made to see how it was solving actual software engineering problems. Messir integrates theories, methods and tools from the following software engineering knowledge areas: *Software Requirements; Software Construction; Software Testing; Software Maintenance; Software Engineering Process; Software Engineering Models and Methods; Software Quality; Software Engineering Professional Practice; Computing Foundations; Mathematical Foundations and Engineering Foundations.* Messir components represent in themselves some improvements w.r.t. the state of the art of those theories, methods and tools, mainly by introducing an improved requirements engineering process, language and verification support based on executable requirements specifications. Furthermore, the Messir approach solves also some actual problems related to software engineering education by offering a product line framework for setting up or improving courses in computer science curricula. This indirectly also impacts actual engineering problems by contributing to raise the software engineering capabilities of engineers and scientists that will feed the jobs in industry, research or education.

Keywords Requirements analysis · Model-driven software engineering · Software engineering environments · Software engineering education

1 Introduction

Since the software crisis, as addressed in the NATO conference [5], a huge amount of progress has been made on the research, engineering and teaching dimensions. It is admitted that software engineering (SE) can be defined as

N. Guelfi
University of Luxembourg
Campus Belval
Avenue de l'Université, 2
L-4365 Esch-sur-Alzette
LUXEMBOURG
E-mail: nicolas.guelfi@uni.lu

> The disciplined application of engineering, scientific, and **mathematical principles, methods, and tools** to the economical production of quality software.
> (Watts S. Humphrey in [29])

and it can be still considered as true that:

> Today, the majority of Engineers understand very little of the "science of programming". On the other side, the scientists who study programming understand very little about what it means to be an Engineer ... the two fields have much to learn from each other and that the **marriage of software and Engineering should be consummated**.
> (David Lorge Parnas in [46])

Furthermore the complexity of the SE domain has been highlighted in a provocative and on purpose way with F. Bauer's famous quote:

> Software engineering is the part of computer science which is too difficult for the computer scientist.
> (Friedrich Bauer [5, 71])

In order to be more precise on what SE is, an important standardization effort by ISO and IEEE has been made in the last decade offering a proposal to define the SE body of knowledge [35]. This body of knowledge defines 15 knowledge areas (KA) decomposed into topics and sub-topics (as illustrated in figures **??**, **??**, **??** and **??**). As usual for such complex work, it can be easily shown that the SWEBOK has some defects but represents one of the best joint efforts from research, education and industry actors to improve the SE domain. It is thus used as a basis for this work and will be exploited in the remaining parts of the paper.

It is a fact that the expansion speed of the software development domain at conceptual and technological levels is so high that it is very difficult for researchers to produce SE knowledge solving SE problems such that they can be scaled and transferred in time to be deployed consistently in education and in real industrial projects.

The spectrum of SE approaches can be observed using the two following axis: **science** and **engineering**. Approaches at the extreme end of the science axis will only consider a SE notion if it has a sound (rigorously defined in a complete and consistent way) mathematical basis. On the extreme of the other axis, one will consider only notions that proved to solve efficiently actual problems encountered in real world projects considering the reality of technological and human resources.

A wide spectrum of approaches exists that we could (rapidly, incompletely and subjectively) [1] see as going from so-called *formal or mathematical methods* like all the ones that are based on or extend the fundamental ones including: *B* [1], *Z* [60], *VDM* [7], *Process Algebras* [41], or *Petri nets* [49]; through semi formal methods like the one proposed on the basis of *UML* [33] and its process *RUP* [39] or similar ones empowered by the model driven engineering community (MDE) actively supported by the *OMG*; then through so called "*agile methods* like Scrum [38] or XP [6]; and finally to successful "spontaneous, intuitive, empirical and artisanal" approaches to software development.

[1] The intent here is not to present a precise and well structured list of scientific approaches to software engineering. It is just intended to name the basic approaches that gave rise to many scientific results used in software engineering.

Each approach proposes to address one or more software engineering topics by providing a method that may be defined using scientific theories and for which a software or hardware tool might also be provided.

The problem addressed by Messir , introduced in this paper, is to propose a SE approach that has variable coverage levels of science and engineering, that covers SE topics based on priority levels driven by pragmatics, that integrates theories, methods and tools and, finally that can be used by engineers to improve their SE mastering by being deployed in SE curricula or in life-long learning trainings. It is the combination of all those qualities that makes the problem hard to solve and was the point referred to by F. Bauer, D. Parnas and others in their various quotes and statements. With this work, it is wished to continue the quest on integrated software engineering methods and it can be considered that the result achieved with Messir is a proof of concept that such improvements can be done.

The main contributions contained in the Messir approach presented in this paper are:

1. A flexible requirements description language
2. An axiomatic and operational semantics supporting declarative executable requirements specifications.
3. An improved use case modeling approach
4. An integrated and flexible method for scientific requirements engineering
5. An approach supported by a software engineering environment
6. A specialized knowledge transfer process for improving coverage of targeted software engineering knowledge areas based on standards.

Section 2 presents an observation of software engineering in research, industry and education in order to introduce the core motivations for Messir ; section 3 introduces the general principles on requirements engineering promoted by the Messir approach; sections 4, 5, 6, 7, 8 present in a synthetic way all the requirements engineering models; section 9 explains how non-functional requirements can be covered; section 10 describes the setting up of a software engineering environment in line with Messir ; finally section 11 introduces the Messir SE Project Courses Product line proposed to ensure knowledge transfer.

Number	Knowledge Area Names
1	Software Requirements
2	Software Design
3	Software Construction
4	Software Testing
5	Software Maintenance
6	Software Configuration Management
7	Software Engineering Management
8	Software Engineering Process
9	Software Engineering Models and Methods
10	Software Quality
11	Software Engineering Professional Practice
12	Software Engineering Economics
13	Computing Foundations
14	Mathematical Foundations
15	Engineering Foundations

Fig. 1 SWEBOK Knowledge Areas

Number	Topic Names
1	Modeling
2	Types of Models
3	Analysis of Models
4	Software Engineering Methods

Fig. 2 SWEBOK topics for (9) Software Engineering Models and Methods

Number	SubTopic Names
1	Modeling Principles
2	Properties and Expression of Models
3	Syntax, Semantics, and Pragmatics
4	Preconditions, Postconditions, and Invariants

Fig. 3 SWEBOK topics for (9.1) Modeling

Number	SubTopic Names
1	Heuristic Methods
2	Formal Methods
3	Prototyping Methods
4	Agile Methods

Fig. 4 SWEBOK topics for (9.4) Software Engineering Methods

2 Software Engineering

In this section, it is proposed to introduce and motivate the existing theories, methods and tools available for software engineering that have been selected to be integrated in Messir . This selection is based on observations of the research and industry sectors reinforced by specific contributions previously made[2]. As said in the introduction, the complexity resides in the selection, adaptation and integration of SE artefacts, thus this section presents and motivates the selected artefacts and, the next sections introduce their adaptation and integration to build the Messir approach.

2.1 Research

It is obvious that researches have been conducted in all the software engineering knowledge areas (see fig. **??**) since 1968 (and even before ...), whether theoretical or applied, being empirical or not. We can say that most of the researches intend to develop sound approaches and a majority of researches conducted can be qualified as "scientific" regarding their aim of being objective, precise and rigorous. Some of them, called theoretical researches, are only using mathematical tools.

[2] A selection of those contributions can be found in the following references: for **theories** ([10], [24], [20], [57], [18]), for **methods** ([42], [47], [25], [27], [11], [48], [56], [22], [19], [4],[16],[53], [40]) and for **tools** ([26], [23], [13])

Since the Messir objective is to define an integrated scientific SE approach, it is more pragmatic to list the notions issued from scientific researches that represents milestones and describe their contribution to SE issues. To this aim, Messir considers the following main notions:

- **Theories**: Concerning basic or more advanced theoretical notions applied to SE and exploited in the context of formal methods, Messir focuses mainly on the following ones:
  (a) set theory: basic concept necessary for modeling information space
  (b) mathematical logic: basic concept for declarative characterization of information spaces
  (c) language theory: basic notion for textual modeling
  (d) axiomatic semantics: basic notion for semantic interpretation of declarative descriptions
  (e) operational semantics: basic notion for semantic interpretation of state modifications of abstract computing machine models.
- **Methods**: modeling is a very fundamental scientific method which involves a Cartesian view of reality and is one of the main discovery tools for the scientist. It implies to represent a phenomenon using a notation. Research has produced an important contribution in defining theories, methods and tools for modeling in all areas including computer science. For what concerns software engineering, model driven engineering (MDE) has been intensively developed in the last 30 years mainly supported by the OMG around its model driven architecture initiative (MDA) [44]. In this context, it may be mentioned the following notions related to modeling for SE:
  (a) MOF: the Meta-Object-Facilities [45], providing a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata driven systems.
  (b) UML: the Unified Modeling Language [33], providing categories of modeling concepts adapted to model various types of properties of IT systems.
  (c) OCL: the Object Constraint Language [34], providing a formal language used to describe logical constraint expressions on UML models.
- **Tools**: Supporting SE activities with tools has often been a concern for researchers to support their solutions. In the context of SE tools, the problem is not only to find the correct set of tools supporting the targeted activities but to integrate them to set up the needed SE workbenches or environments. From this perspective, Messir considered the following general tools issued from research and industry:
  1. Eclipse [17]: the open source integrated development environment having a powerful plug-in system for customization.
  2. Xtext [66]: an advanced framework for development of general or domain specific languages.
  3. Sirius [59]: A generic eclipse tool for graphical modeling workbench development.
  4. Sictus Prolog [58]: A generic logic programming tool including constraints solver libraries allowing for implementation of axiomatic and operational semantics of declarative formal specifications.

2.2 Industry

Determining the state of practice of software engineering in industry is not an easy task mainly due to the fact that the main goal of industry is neither to conduct such studies nor to make public their engineering practice. When they conduct such studies they furthermore are not keen to publish them since they evolve in a world highly competitive in which any information can become sensitive and may impact their business. Nevertheless, it is still possible to find direct or indirect ways to have some information on this subject.

Since our goal is to better determine the nature of the software engineering practice in industry at a high conceptual level and not to conduct a detailed, quantitative and qualitative study, it is decided to observe industry project management and industry project practices.

2.2.1 Industry project management

It is shared the idea advocated by the software engineering institute (SEI) that "the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it" [55]. CMMI-DEV is the last and most advanced maturity model proposed as a reference model covering activities products and services development. It is thus interesting to observe industry practice through the capability maturity model in order to better determine the problem to solve to improve software engineering practice. According to available studies [14], [43], [9], [54] and also based on our own empirical experience in handling more than 400 internships in industry for bachelor and master students, being an expert for the court of justice on conformance questions in trials for the software industry, we can state that:

(a) approximately 3/4 of companies are at level 1 or 2 (initial or repeatable).
(b) among the companies that goes for CMMI appraisal, approximately 80% are at or below level 3 (defined) and 20% at level 4 (managed) and 5 (optimizing).

This meant for us that the focus should be made on problems encountered by low maturity level companies thus targeting requirements engineering and more precisely requirements definition, verification and validation.

The success that the so called "agile methods" had, especially in companies having maturity level below 3, has also been studied. From our point of view, agile methods represents a "success story" from the software engineering point of view. This is because it penetrated industry at a high rate and contributed to improve the SE practices in industry, especially promoting the following SE knowledge areas: *Software Construction, Software Testing, Software Engineering Management, Software Engineering Process, Software Quality, Software Engineering Professional Practice*.
Available studies [61],[51] or [65] and [64] show the significant adoption of the approach by the majority of the software development industry actors.
It has been deduced from this observation that the main success factors for improving SE practice, mainly for those level 3 (or less) companies, are:

1. an iterative and incremental process
2. validation is mainly based on usage scenarios (user stories) as test cases
3. a large part of the product quality relies on the programmer who is the one-man band impacting all SE knowledge areas. A large project is a collection of one-man bands that need adequate project management to be orchestrated.
4. documentation and modeling are seen as a burden even though partly tolerated.

2.2.2 Industry project practices

In order to observe more precisely how modeling is used in industry, it can be mentioned the following study [30] made in a context of model driven engineering projects over 17 companies and 250 engineers. The interesting facts that can be extracted from this study are:

(a) On modeling approaches used: 85% make use of UML, 40% use a DSL of their own design, 25% use a DSL provided by a tool, 25% use BPMN, 10% use SysML and MATLAB/Simulink.

(b) On UML Diagrams usage: 87% Class Diagram, 56% Activity Diagram, 38% Use Case Diagram, 33% Sequence Diagram, 23% State Machine Diagram. Other diagrams like: Component Diagram, Flow Diagram, Entity Relationship Diagram, Deployment Diagram, Object Diagram, Composite Structure Diagram are rarely cited.
(c) On the perceived impact on (**P**)roductivity or (**M**)aintainability) of Model Driven Approaches used: **communication**: 73.7 (P), 66.7 (M), Use of models for **understanding** a problem at an abstract level: 73.4 (P), 72.2 (M), **Code generation**: 67.8 (P), 56.9 (M), Use of models to capture and **document** designs: 65.0 (P), 59.9 (M), Use of model-to-model **transformations**: 50.8 (P), 42.6 (M), Use of **domain-specific languages** (DSLs): 47.5 (P), 44.0 (M), Model **simulation** - Executable models: 41.7 (P), 39.4 (M), Use of models in **testing**: 37.8 (P), 35.2 (M).

From those extracted facts, it can be deduced that:

5. modeling notations should be reduced to the smallest possible set with and based on standard concepts.
6. design and usage of domain specific languages is encouraged if based on standard concepts and supported by automated tools.
7. simulation, testing and code generation are mandatory features to make diagrams perceived as having impact on productivity.
8. usage of modeling notations should be flexible enough to allow various and freely chosen precision (i.e. scientific) levels.

2.3 Education

Software engineering education is a very complex task and any SE approach should also take into account the learning support. Presented below is part of the study results we made and which is based on important milestones provided to the SE community which are:

- SWEBOK - Software Engineering Body of Knowledge (submitted [35],[31])
- CS2013 Undergraduate Curriculum (submitted [52])
- SE2014 Undergraduate Curriculum (submitted [2])
- GSwE2009 Graduate Software Engineering 2009 ([50])

In order to determine the actual coverage of the SWEBOK knowledge areas in education, it is presented the evaluation made over a sample of academic curriculums (selected worldwide) in order to determine how they were covering the SWEBOK knowledge areas (KAs). The sample has been structured using ISCED classification [62] and world regions defined for this study. The table given in figure 5 shows the main results giving, for each knowledge areas, the coverage percentages (mean, minimum, maximum and standard deviation).

This observation study has been useful to determine some of the main issues in SE education and their probable causes which largely impacted Messir goals and design. They can be summarized as follow:

1. Most of the curriculums consider software engineering as a discipline in itself and thus include a specific SE course only once in a full bachelor program. This makes the lecture difficult to design and execute since it implies that the teacher has to select a subset of the KAs to cover in a short time budget, and it supposes too that the teacher is expert in all the KAs, which of course, can rarely be the case.
2. most of the KA subtopics are covered thanks to Engineering Projects offered in the curriculum. Those projects are ideally used to cover KAs such as: Software Construction (3), Software Engineering Management (7), Software Engineering Professional Practice (11).

KA	Name	Mean	Min	Max	StdDev
1	Software Requirements	64	33	80	21
2	Software Design	46	17	66	21
3	Software Construction	51	36	67	16
4	Software Testing	28	5	42	16
5	Software Maintenance	11	0	28	14
6	Software Configuration Management	6	0	17	8
7	Software Engineering Management	68	58	75	7
8	Software Engineering Process	35	33	40	3
9	Software Engineering Models and Methods	66	31	100	29
10	Software Quality	29	8	50	20
11	Software Engineering Professional Practice	71	58	84	13
12	Software Engineering Economics	24	8	60	25
13	Computing Foundations	58	11	79	32
14	Mathematical Foundations	56	19	75	26
15	Engineering Foundations	38	18	65	20

Fig. 5 Global KA Coverage (%)

3. Software Requirements (1) is quite well covered and often benefits from a full lecture.
4. Software Testing (4) is less covered than expected especially regarding its importance in industry.
5. the importance, volume and coverage of theoretical computer science at bachelor level (being professional oriented or not) is way over what is described in the SWEBOK. This is mainly due to the profile of academic teachers that are mostly scientists, recruited based on scientific results often having been done based on theoretical work. It represents also a wish and a need to provide a strong scientific and theoretical basis during education.
6. the topics poorly addressed are Software Configuration Management (6), Software Engineering Economics (12) and Software Quality (29) and more surprisingly Software Maintenance (5). This last fact seems in direct opposition with the reality of engineering activities which, for the majority, consists in supporting software evolution and maintenance.

Furthermore other studies [63] show that in the coming years 74% of the jobs in the STEM[3] areas will be in the computer science area and only 30% of the new jobs in computer science could be fulfilled by newly graduated bachelor students.

The conclusion drawn is that it is mandatory to provide the education community with a knowledge transfer support that contributes to improve the capabilities of our engineers and technicians with regards to all the SWEBOK knowledge areas. Messir comes with such artefact described in this paper.

3 Messir Requirements Engineering

In this section it is presented the main elements allowing to understand the definition of the Messir approach for requirements engineering. Due to the limited size of this paper it cannot be expected to have the full details of the approach, this is why the description is highly synthetic and focuses on the main characteristics. This allows a macro presentation of Messir showing the integration achieved.

[3] Science, Technology, Engineering and Mathematics.

3.1 Running Example

Some illustrations are given using a crisis management case study. It is a simplified version of the one proposed in [42] and [37]. In the version used in this paper, the system (called *iCrash*) is intended to support the management of crisis situations. The *iCrash* stakeholders are:

- `Communication Companies`: have the capacity to ensure communication of information between its customers and the *iCrash* system. Objectives are: to be able to deliver any SMS sent by any human to the *iCrash* 's dedicated phone number; to be able to transmit SMS messages from the ABC company that owns the *iCrash* system to any human having an SMS compatible device accessible using a phone number.
- `Humans`: a human is any person who considers himself related to a car crash (a specific type of crisis situation) either as a witness, a victim or an anonymous person. The objectives of a human are: to inform the *iCrash* system about the crisis situation he detected; to be sure that the ABC company has been informed about the situation; to be informed about the situation of the crisis he his related to as a victim or witness.
- `Coordinators`: employees responsible for handling one or several crisis. The objectives of a coordinator are: to securely monitor the existing alerts and crisis; to securely manage alerts and crisis until their termination.
- `Administrator`: the employee of the ABC company responsible for administrating the *iCrash* system. The objectives of an administrator are to add or delete coordinator actors from the system and its environment.
- `Creator`: technician who is installing the system on the targeted deployment infrastructure. The objectives of a `Creator` are: to install the *iCrash* system; to define the values for the initial system's state; to define the values for the initial system's environment; to ensure the integration of the *iCrash* system with its initial environment.
- `Activator`: logical representation of the active part of a system. It represents an implicit stakeholder belonging to the system's environment that interacts with the system autonomously without the need of an external entity. It is usually used for representing time triggered functionalities. The objectives of the *iCrash* `activator` are: to communicate the current time to the system; to notify the administrator that some crises are still pending for a time too long.

3.2 Process

The Messir Requirements Engineering process (Mewop) is **iterative** and **incremental** and depicted in figure 6 which sketches the process activities. The main objective is to describe a set of functional and non-functional properties. The functional properties are modeled using five model types:

- Use case Model
- Environment Model
- Concept Model
- Operation Model
- Test Model

Each model can be described at different scientific levels:

- **Definition Level:** using natural language descriptions (structured textual or graphical) for each of the pre-defined Messir model type.

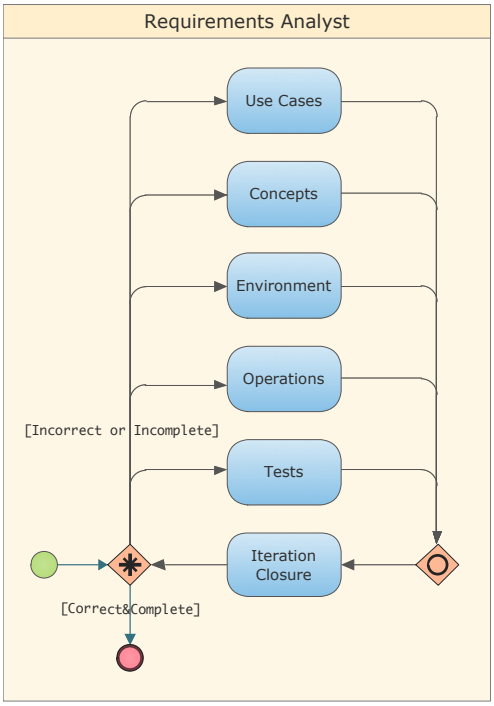

Fig. 6 Messir MeooP Diagram

– **Specification Level:** using object-oriented and constraint oriented specifications.
– **Simulation Level:** using executable code allowing for requirements simulation using the Messir simulator (MesSim) in compliance with the Messir abstract machine MesSam.

Figure 7 sketches model focus by scientific level targeted. For example, it shows that if **Definition level** is targeted then most of the focus will be on use case modeling.

Requirements descriptions are provided using textual languages and their use depends on the targeted scientific levels. Messir is designed to be flexible and used in a large spectrum of life cycles. Thus it is up to the engineer to determine, for the requirements description, the scientific level targeted and the languages used. It can be chosen to use jointly different scientific levels for different functionalities of the system under consideration. Such choice is based on several project constraints and system characteristics. Safety or business critical parts of the system might need more precise and rigorous requirements descriptions, while the project delays, budget and team expertise might justify to provide only natural language descriptions of the requirements or, even, no description at all.

Messir provides three domain specific languages defined at meta-level using grammars supported by the Xtext technology [66]). The languages available are:

– **Documentation language (`msrd`)**: allowing for free natural language descriptions but structured using domain specific templates. Figure 18 provides an illustration of this template language for the system's operation `oeAddCoordinator` description. The generated documentation from this description is shown in figure 19.

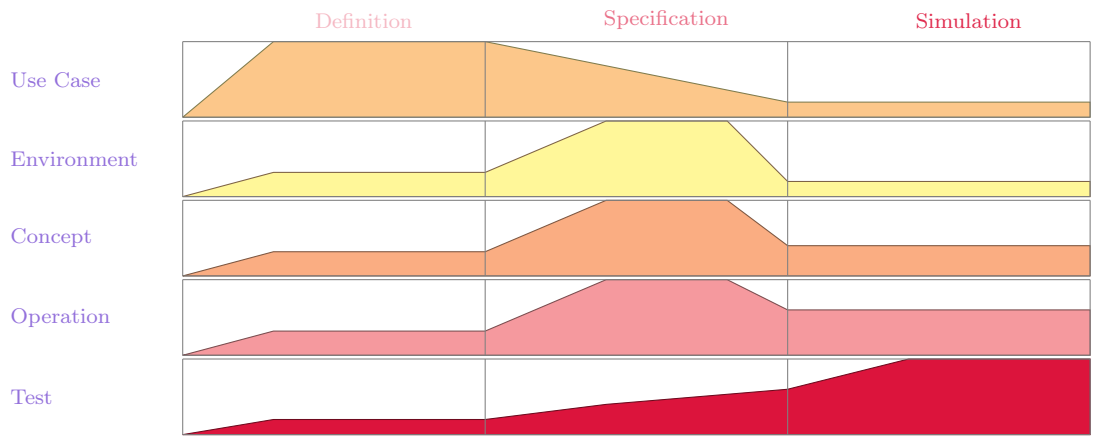

Fig. 7 Messir MeooP - Model Focus by Level

- **Specification language (`msr`)**: allowing for more precise conceptual and logical descriptions given using the Messir language mcLwhich is an adapted executable version of the OCL language [34]. Figure 20 provides an extract illustrating this language for the system's operation `oeAddCoordinator` specification.
- **Axioperational Language (`pl`)**: a Prolog compliant language is proposed to allow for ***axioperational*** (i.e. axiomatic and operational) semantics of the requirements descriptions. Figure 21 provides an extract illustrating this language for the same operation.

Level	msrd	msr	pl
Definition	+++	+	-
Specification	++	+++	-
Simulation	+	+	+++

Fig. 8 Categories of Messir languages by Scientific level.

This approach allows the requirements analyst to tune the scientific level of the requirements description to the exact need and is synthesized in the generated documentation by a scientific level table using the system operations as observation criteria. Figure 9 shows this principle in the context of the *iCrash* case study. A "•" indicates the targeted level, a "✓" indicates a *reached* targeted level and, a " x" indicates a reached level. Thus the targeted and reached levels are clearly indicated and the flexibility of the combinations covers pragmatically all the needs (i.e. reached level equal, lower or greater than targeted level). The table shown in figure 8 indicates the languages usage degree (+/−) by scientific level.

Messir includes a detailed process definition giving guidelines for activities to iteratively and incrementally engineer the requirements description models ([21]). The different models to be provided are introduced in the next sections.

3.3 Requirements Engineering Basic Concepts

Messir considers that the system under development is made of a central system (called *system*) and its *environment*. The system has an observable state and a set of *system operations*.

Actor	Operation	Def.	Spec.	Sim.
actMsrCreator	oeCreateSystemAndEnvironment			x
actCoordinator	oeSetCrisisHandler	•	•	✓
	oeGetCrisisSet	x	•	
	oeGetAlertsSet		•	
	oeValidateAlert	•		
	oeInvalidateAlert	•		
	oeSetCrisisType	•		
	oeSetCrisisStatus	•		
	oeReportOnCrisis		•	
	oeCloseCrisis		✓	x
actAuthenticated	oeLogin	✓		
	oeLogout	✓		
actComCompany	oeAlert	x		•
actActivator	oeSollicitateCrisisHandling			•
	oeSetClock	•		
actAdministrator	oeAddCoordinator	•		
	oeDeleteCoordinator	•		

Fig. 9 Illustration of Messir Scientific Levels Management for *iCrash*

An environment is defined by a set of *actor* instances each of which having an *output interface* defining a set of asynchronous messages that can be sent by the actor to the system and an *input interface* defining a set of messages received by the actor from the system. Interactions are sequences of input and output messages between the environment and the system. An actor sends an output message in order to trigger a system's atomic and instantaneous operation which might: change the system's state or the environment composition and generate message sending to the actors of the environment.

4 Messir Use case Model

Use cases analysis is a very efficient technique for requirements elicitation which has been studied and applied efficiently in industry [36], [3, 15]. The main objective of use case modeling is to specify and illustrates **business use cases** of the actors concerned directly or not by the system under development. Messir introduces a new use-case approach that overcomes some limitations encountered in practice when using tool-supported "standard" approaches. To this purpose, Messir use case analysis has the following characteristics:

- uses a **textual** modeling language for engineering the use case model.
- uses graphical use case diagrams generated from textual models as communication views. Views are based on the standard UML use case diagrams.
- replaces `extends` and `includes` by a `reuse` association to simplify the modular description of use cases.
- replaces sequential ordering of activities (`main success scenario` and `extensions`) by a set of steps and a set of ordering constraints over those steps. This solves a main issue in use case writing which is misuse or inefficient use of main scenario and extensions.
- introduces use case instances to illustrate concrete scenarios that represent meaningful scenarios satisfying the use case model description.
- restrict abstraction to three levels (summary, user-goal and sub-function).
- introduces for each use case the possibility to describe typed parameters.
- introduces for each actor the notions of role (primary/secondary), mode (direct/indirect), involvement (pro-active/active/reactive/passive) and multiplicity.

– allows to provide documentation for each use case including: goal description and protocol/pre/post conditions description if necessary).
– provides **graphical** representations of use cases and use cases instances using Messir use-case diagrams and sequence diagrams generated, maintained automatically w.r.t the textual descriptions. This contributes to make Messir useful for production and not used as a documentation only approach as often encountered.

Figure 10 provides a textual description of a use case of the Messir use-case model using the `msr` language and figure 11 represents the automatically generated diagrammatic view for this use case. Figure 12 shows a textual description extract of a use case instance and figure 13 its graphical representation.

```
Use Case Model {
 use case system summary
  suGlobalCrisisHandling() {
  actor actCoordinator[primary,active]

  reuse ugSecurelyUseSystem[1..*]
  reuse ugMonitor[1..*]
  reuse ugManageCrisis[1..*]

  step a: actCoordinator
      executes ugSecurelyUseSystem
  step b: actCoordinator
      executes ugMonitor
  step c: actCoordinator
      executes ugManageCrisis

  ordering constraint
   "steps (a) (b) and (c) executions are interleaved
   (steps (b) and (c) have their protocol constrained by steps of (a
  ))."
  ordering constraint
    "steps (a) (b) and (c) can be executed multiple times."
```

Fig. 10 Illustration of a textual use case description

5 Messir Environment Model

The objective of the environment model is to describe *actors* with their *output* and *input* interfaces used to interact with the system[4]). The environment model is textually described using the Messir language and graphical view generated using simple class diagram notation.

Figure 14 illustrates such a textual description for a part of the *iCrash* example for the coordinator actor (`actCoordinator`) while figure 15 provides its graphical view. An actor is associated to the system state root[5](the role provides the association name end) and can inherit from another actor (extends). It must be mentioned that the message parameters are typed using the concept model below. So iterations and increments must be made consistently between the environment model and the concept model (the Excalibur tool [12] provides support for

[4]same environment notion as the one introduced in [**?**]
[5]instance of the `ctState` class

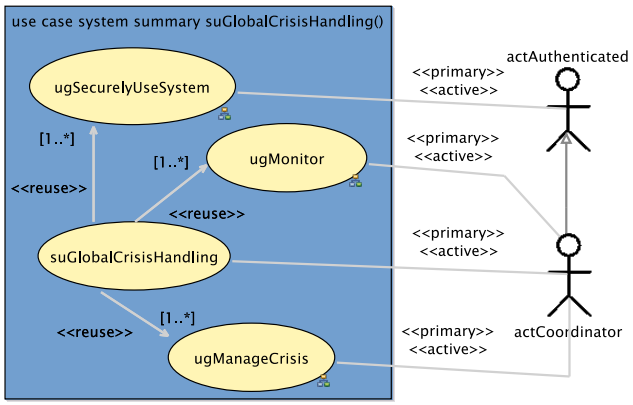

Fig. 11 Illustration of generated graphical view for a use case description

```
//-------------------------------------------------------
    steve
    executed instanceof subfunction
       oeLogin("steve","pwdMessirExcalibur2017"){
        ieMessage('You are logged ! Welcome ...') returned to steve
       }
//-------------------------------------------------------
    steve
    executed instanceof subfunction
       oeGetCrisisSet("pending"){
        ieSendACrisis("crisis with ID 1 details") returned to steve
       }
//-------------------------------------------------------
    steve
    executed instanceof subfunction
       oeSetCrisisHandler("1"){
        ieSmsSend("+3524666445252","The handling of your alert by our
     services is in progress !")
        returned to tango
        ieMessage("You are now considered as handling the crisis !")
        returned to steve
       }
//-------------------------------------------------------
```

Fig. 12 Illustration of a textual use case instance description

this task). Those messages and their parameter's types are partly elicited thanks to the use cases (sub-functions) and the use case instances.

6 Messir Concept Model

The objective is to specify the types structuring the information either used to define the system's state during its life cycle or exchanged with its environment. The Messir Concept Model provides the specification of: **Primary types** used for structuring the information defining the system's state; **Secondary types** for additional types necessary either for the information used in input or output message parameters, for actors' attributes or for the operation specifications

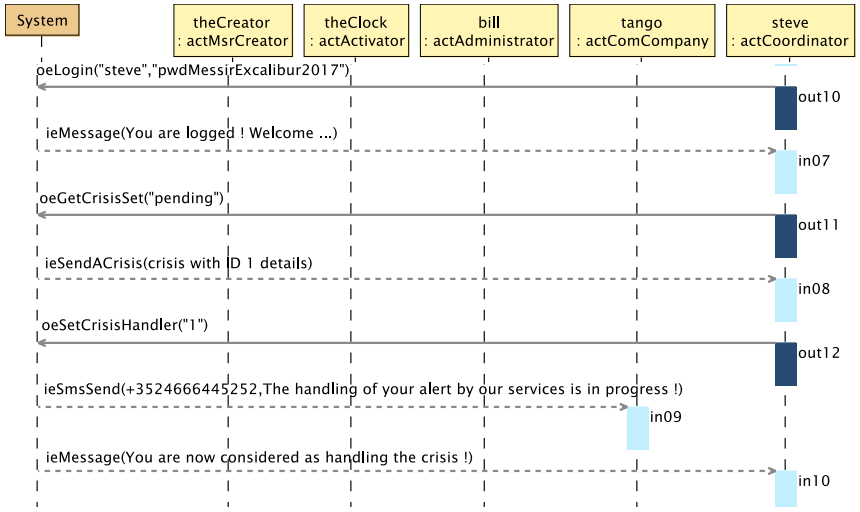

Fig. 13 Illustration of generated graphical view for a **use case instance** description

```
actor actCoordinator
    role rnactCoordinator
    cardinality [0..*]
    extends actAuthenticated{

 operation init():ptBoolean

 output interface outactCoordinator{
  operation oeInvalidateAlert(AdtAlertID:dtAlertID ):ptBoolean
  operation oeCloseCrisis(AdtCrisisID:dtCrisisID ):ptBoolean
  operation oeGetAlertsSet(AetAlertStatus:etAlertStatus ):ptBoolean
  operation oeGetCrisisSet(AetCrisisStatus:etCrisisStatus ):
     ptBoolean
  operation oeSetCrisisHandler(AdtCrisisID:dtCrisisID ):ptBoolean

                                  . . .

 input interface inactCoordinator{
  operation ieSendAnAlert(ActAlert:ctAlert ):ptBoolean
  operation ieSendACrisis(ActCrisis:ctCrisis ):ptBoolean
  }
}
```

Fig. 14 Illustration of a textual **Environment Model** description

given in the operation model.

Primary and secondary types can be defined using the following meta-types: *Class type*, *DataType type* and *Relation type* consistent with the standard data structures:

– a class type is defined using a name, a set of typed attributes and an optional supertype name.There is only one class type named `ctState` dedicated to the system state root. A class attribute type must be of datatype type. Each primary class type is associated to the `ctState` class by an aggregation association in order to provide an access to instances from the `ctState` instance necessary to describe operation semantics (see below).

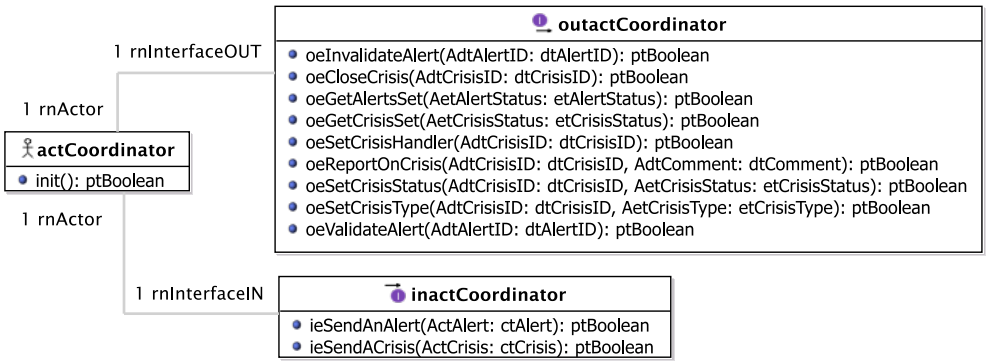

Fig. 15 Illustration of the generated graphical view for a Environment Model description

- a datatype type can be a *structured data type*, an *enumeration* or a *primitive type*. A *structured data type* defines a set of tuples whose elements (named attributes) are typed using any *primary type*. Primitive types are: `Boolean` , `Integer` , `Real` or `String` [6] and correspond to types defining atomic values supported by the abstract machine of the simulator (all the useful primitive type operations are provided in Messir libraries written in Messir at simulation level).
- relation types are aggregation, composition or (common) association.
- functions can be part of a class type definition or a datatype type definition. Those operations semantically correspond to logical predicates and are used to allow for a structured and concise writing of predicates.

Figure 16 illustrates such a textual description for the concept model using a class type primary type `ctAlert`. Its graphical representation is shown in figure 17 which depicts the full concept model for the *iCrash* example.

7 Messir Operation Model

The objective of operations specification is to provide the *properties* that "characterize" all valid *"executions"* for each system operation [7]. An operation specification in Messir is an axiomatic specification of its semantic properties. Those properties are grouped in the following categories: *Pre-protocol*, *Pre-functional*, *Post-functional and Post-Protocol*:

- **Pre-functional properties** are conditions under which the system operation is considered *defined*. Expressed using: the system's state **@pre**, the environment state **@pre** and the operation parameters.
- **Pre-protocol properties** are conditions under which the operation is considered *available*. Expressed using: the system's state **@pre** including the additional variables for protocol specification, the environment state **@pre** and the operation parameters.

[6] a naming convention in Messir proposes to use prefixes to ease understanding while reading textual descriptions (e.g. `pt,dt,ct,et,oe,ie` will stand for primitive type, datatype,class type, enumeration type, output event, input event).

[7] Considering a transition system as semantic model, we need to characterize the transitions $< state_i, env_i > \xrightarrow{oe_i,[ie_1,\ldots,ie_n]} < state_{i+1}, env_{i+1} >$ in which $< state_i, env_i >$ (resp. $state_{i+1}, env_{i+1} >$) represents the system's state and environment state **@pre** (resp. **@post**), oe_i the system operation executed triggered by an output event sent by an actor, ie_i the input events sent to actors that define the specification semantics

```
Concept Model {

 Primary Types{
                                    . . .
  class ctAlert role rnctAlert cardinality [0..*]{
   attribute id:dtAlertID
   attribute status: etAlertStatus
   attribute location:dtGPSLocation
   attribute instant:dtDateAndTime
   attribute comment:dtComment

   operation init(    Aid:dtAlertID ,
         Astatus:etAlertStatus ,
         Alocation:dtGPSLocation ,
         Ainstant:dtDateAndTime ,
         Acomment:dtComment ):ptBoolean
   operation isSentToCoordinator(AactCoordinator:actCoordinator ):
      ptBoolean

  }
```

Fig. 16 Illustration of a textual Concept Model description

- **Post-functional properties** are conditions describing the operation's *functionality*. It characterizes a set of valid post-states for the system and the environment, a multiset of messages sent to actor instances. Expressed using: the system's state **@pre** or **@post**, the environment state **@pre** or **@post** and the operation parameters.
- **Post-protocol properties** are conditions describing the operation's *impact* on the system's interaction *protocol* (i.e. the system status **@post** makes available only the wished operations). They are expressed using the variables for protocol specification **@post**.

To illustrate the operation model, we consider an operation used by the administrator actor to add a coordinator actor. Figure 18 provides its description at **definition level** using the `msrd` documentation grammar of Messir . Figure 19 illustrates the automatically generated presentation of this definition level documentation. Figure 20 provides its description at **specification level** using the `msr` specification grammar of Messir which includes the "OCL -like" mcl constraints specification of the conditions. Finally, figure 21 shows an extract of its implementation using the Prolog language [8]) and using the Messir Prolog layer offered by the MeSSim simulator (e.g. **msrOp, msrNav**).

8 Messir Test Model

The objective of the test model is to define test cases for verification and validation purposes. A test case is a sequence of test steps. Each test step defines: the *test message* characterizing the system operation triggered; the *test constraints* which defines the domain for the parameter values and the *test oracle* to define the test step acceptance conditions.

Messir mainly exploits test cases to:

- verify the requirement descriptions by simulation to detect if the specification contains errors.
- validate the requirements description by asking the system's customer to determine validation test cases.

[8] this can only be done by requirements engineers having a high scientific level.

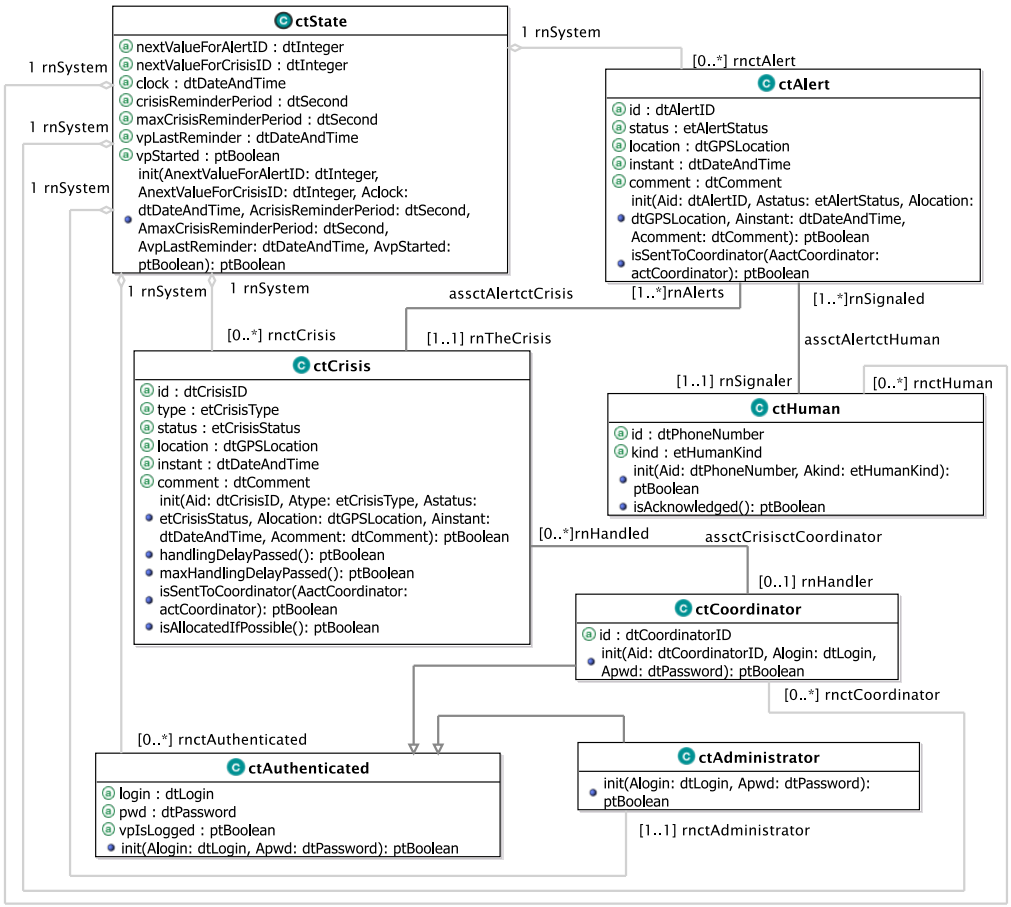

Fig. 17 Illustration of the generated graphical view for a Concept Model description

– verify the delivered *system program* produced after each project implementation iteration by using the test case specification to implement verification test cases to find errors in the implemented program.

Figure 22 illustrates a textual description for a test step of a test case for the *iCrash* system. This test step is given at specification level and tests the correct execution of the operation `oeSetCrisisHandler` triggered by the coordinator actor. The test step is considered successful if the simulator can execute a valid transition on the abstract machine and if the oracle constraint is satisfied. Figure 23 illustrates a graphical representation using UML sequence diagram generated from an actual test case instance either produced by the simulator or manually provided to illustrate the test case expected execution.

9 Non Functional Requirements

Messir bases the requirements description for non-functional requirements (NFR) on the ISO standard on quality requirements [32] (SQUARE) including a quality model organised in

```
@@Operation
icrash.environment.actAdministrator.outactAdministrator.
    oeAddCoordinator

@description
"sent to add a new coordinator in the system's post state and
    environment's post state."

//preProtocol descriptions
@preP
"the system is started"
@preP
"the actor logged previously and did not log out ! (i.e. the
    associated ctAdministrator instance is considered logged)"
@endPreP

//preFunctional descriptions

@postF
"the system's state has a new instance of ctCoordinator initialized
    with the given values."
@postF
"the new actor instance and ctCoordinator instance are related."
```

Fig. 18 Illustration of a generated documentation Definition level Operation description

oeAddCoordinator	
sent to add a new coordinator in the system's post state and environment's post state.	
Parameters	
1	**AdtCoordinatorID: dtCoordinatorID** used to initialize the id field
2	**AdtLogin: dtLogin** used to initialize the login field
3	**AdtPassword: dtPassword** used to initialize the password field
Return type	
ptBoolean	
Pre-Condition (protocol)	
PreP 1	the system is started
PreP 2	the actor logged previously and did not log out ! (i.e. the associated ctCoordinator instance is considered logged)
Pre-Condition (functional)	
PreF 1	it is supposed that there cannot exist a ctCoordinator instance with the same `id` attribute than the one the administrator wants to create.
Post-Condition (functional)	
PostF 1	the environment has a new instance of coordinator actor allowing for input/output message communication with the system.
PostF 2	the system's state has a new instance of ctCoordinator initialized with the given values.
PostF 3	the new actor instance and ctCoordinator instance are related.
PostF 4	the new actor instance and ctCoordinator instance are related according to the authenticated association.
PostF 5	the administrator actor is informed about the satisfaction of its request.
Post-Condition (protocol)	
PostP 1	none

Fig. 19 Illustration of a Definition level Operation description

Quality Criteria (see figure 24) and Quality sub-characteristics (like Time behaviour, Resource and utilization Capacity for the performance efficiency quality criteria).

Currently Messir only allows for inclusion of requirements using natural language based on the ISO standard structure. It is planned to integrate a domain specific language for NFRs that will allow for better handling of NFRs improved with some quantification criteria based on the ISO standard and the formal framework proposed in [20].

```
operation: actAdministrator.outactAdministrator.oeAddCoordinator(
    AdtCoordinatorID:dtCoordinatorID, AdtLogin:dtLogin, AdtPassword:
    dtPassword):ptBoolean

postF{
 let TheSystem: ctState in
 let TheactCoordinator:actCoordinator in
 let ThectCoordinator:ctCoordinator in
 self.rnActor.rnSystem = TheSystem
 and self.rnActor = TheActor
/* PostF01 */
 TheactCoordinator.init()
/* PostF02 */
 and ThectCoordinator.init(AdtCoordinatorID,AdtLogin,AdtPassword)

/* PostF03 */
 and TheactCoordinator@post.rnctCoordinator = ThectCoordinator

/* PostF04 */
 and ThectCoordinator@post.rnactAuthenticated = TheactCoordinator

/* PostF05 */
 and TheActor.rnInterfaceIN^ieCoordinatorAdded()
```

Fig. 20 Illustration of a Specification level Operation description

```
msrop(outactAdministrator,
  oeAddCoordinator,
  [preProtocol,Self,
   AdtCoordinatorID,
   AdtLogin,
   AdtPassword
   ],
  []):-
                                    ...
/* PostF03 */
msrNav([TheactCoordinator],
    [msmAtPost,rnctCoordinator],
    [ThectCoordinator]),
/* PostF05 */
 msrNav([TheActor],
    [rnInterfaceIN,
     ieCoordinatorAdded,[]],
    [[ptBoolean,true]]),
```

Fig. 21 Illustration of a Simulation level Operation description

10 Messir Tool Support

As said in the introduction, software engineering is the pragmatic integration of theories, methods and tools for software production. The Messir approach is supported by a software engineering environment that is built using, as a central part, its specifically designed SE workbench Excalibur ([12]). A software engineering environment set up for a software development project aims at supporting the following main dimensions: **Requirements Analysis**, **Design**, **Construction**, **Management**, **Quality** and **Maintenance**. The Messir approach proposes to build a software engineering environment (SEE) integrating the necessary tools and technologies. An example of such a SEE built for the *iCrash* project is [9]:

[9] a deployment script is provided such that the deployment of such SEE takes around from 2 to 4 hours depending of your internet connexion speed.

```
test step ts12oeSetCrisisHandler order 12{
   variables{
     TheActor : actCoordinator
     AdtCrisisID : dtCrisisID
    }
   constraints{
     TheActor=TheSystem.rnactCoordinator
     ->select(a | a.rnctCoordinator.login.value.eq('steve'))
     ->any2(true)
     //and AdtCrisisID.value= '1'
    }
   test message{
    out:TheActor sends to system actCoordinator.outactCoordinator.
  oeSetCrisisHandler(AdtCrisisID)
   }
   oracle{
    variables{
     AMessage:ptString
     AdtPhoneNumber:dtPhoneNumber
     AdtSMS:dtSMS
     ActAlert:ctAlert

     TheComCompany: actComCompany
     TheCoordinator:actCoordinator
    }
    constraints{
     AMessage = 'You are now considered as handling the crisis !'
     AdtSMS.value = 'The handling of your alert by our services is
  in progress !'
     TheComCompany.inactComCompany.ieSmsSend(AdtPhoneNumber,AdtSMS)
     TheCoordinator.inactCoordinator.ieSendAnAlert(ActAlert)
     TheActor.inactAuthenticated.ieMessage(AMessage)
   }
```

Fig. 22 Illustration of a Specification level Test description

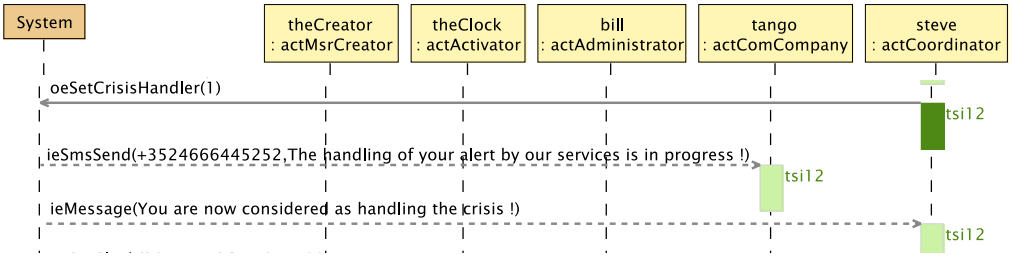

Fig. 23 Illustration of a generated graphical representation of a Test instance description

1	**Functional suitability**
2	**Performance efficiency**
3	**Compatibility**
4	**Usability**
5	**Reliability**
6	**Security**
7	**Maintainability**
8	**Portability**

Fig. 24 SQUARE Main Quality Criteria

- **Requirements Analysis:** the Excalibur SE workbench supporting the Messir requirements textual and graphical modeling, verification using Prolog generated and, completed code report generation (Eclipse, Sirius, Xtext, UML, Prolog, Latex, PDF).
- **Design:** Eclipse UML Designer for production of design graphical models, Messir design document template for Latex (UML, Latex, PDF).
- **Construction:** Java for functionalities, JavaFx for graphical user interfaces, MySQL for data persistence, Java RMI for distributed processing, Apache, Tomcat and qmail for internet services (Eclipse, Java, JavaFx, efxclipse, Apache, TomCat, MySQL, qmail).
- **Management:** Atlassian Confluence for knowledge base collaboration tool, SubVersioN versioning and revision control tool, Atlassian Bamboo and Apache Maven continuous integration server for project builds (Atlassian, Confluence, Bamboo, SVN, Maven).
- **Quality:** Test based Verification & Validation using the Prolog simulator, the SWTbot testing tool for graphical user interface, JUnit unit testing framework and EclEmma Java code coverage. Ensuring syntax validation tools using Eclipse and Xtext frameworks (Eclipse, Xtext, Prolog, Junit, SWTbot).
- **Maintenance:** Atlassian JIRA as issue tracking tool, SubVersioN for versioning and revision control tool, Atlassian Bamboo and Apache Maven continuous integration server for project builds (Atlassian, Jira,Bamboo, Maven).

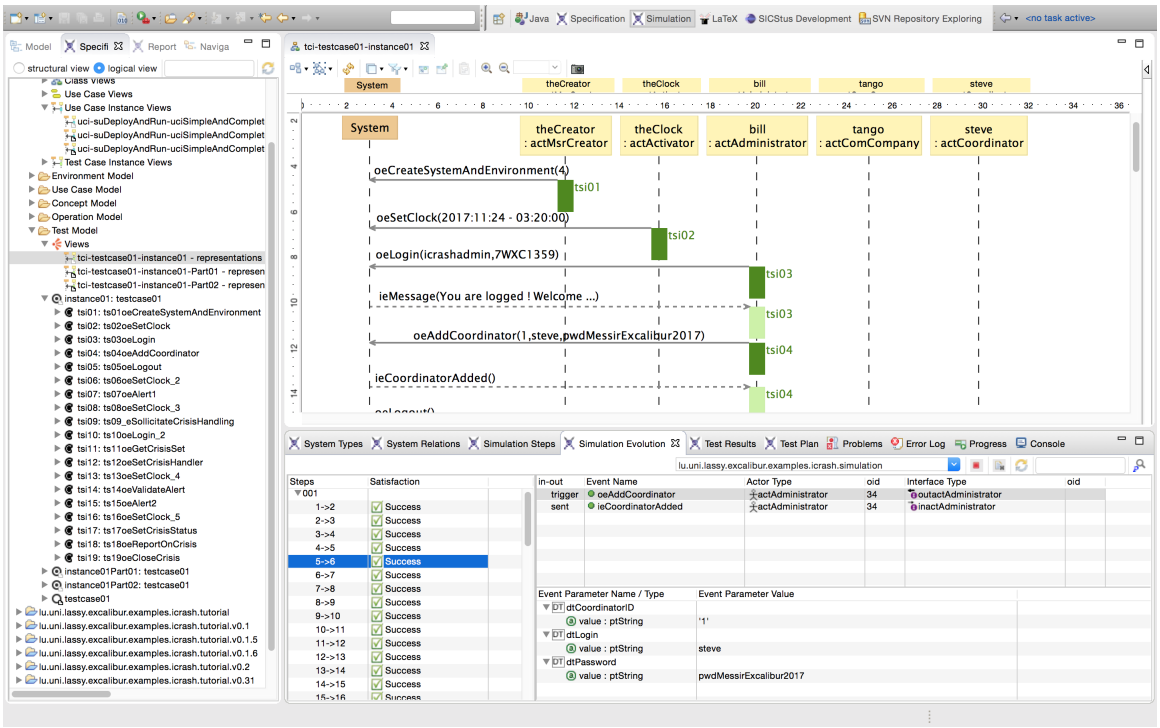

Fig. 25 Messir Requirements V&V using the Excalibur SE Workbench

11 Messir in Software Engineering Education

This section presents a novel knowledge transfer initiative called Messep for "MESsir Software Engineering project courses Product line". The idea is to allow an instructor to derive a full project course including syllabus, project description and project inputs including a software engineering environment as described in the previous section. The possible variation points are: **SWEBOK** (knowledge areas,topics,sub-topics), **Application Domains** [28] (market,categories,sub-Categories) and **IT Technologies**.

Two derivation processes are possible: forward and back&forth. The forward process requests to: bind the variabilities, define deltas (add/modify variation points, variants, constraints) and derive the SE project variant. The back&Forth requests to: select a SE project variant from the product line; define deltas (optional); derive a refined SE project variant (optional).

This allows to engineer a full SE project that can be included in any courses at bachelor or master level (using the International Standard Classification of Education ISCED [62]). At bachelor level, the main courses that could benefit from Messep are: Requirements engineering with use cases; Practical development projects with Java , JavaFx , MySQL ; Introduction to software engineering concepts; Introduction to development methods concepts; Introduction to product quality; Verification and Validation. At master level it could be: Advanced Requirements Engineering; Model Driven Engineering; Domain Specific Languages: concepts and tools; Software Engineering Environments: use and development; Formal Methods; Operational and Axiomatic Semantics; Testing and Model checking; Constraint logic programming.

Features	**Details**
Project Name	*iCrash* v 1.0
ICSED Level	BA 655 (Bachelor/Professional/First degree)
Schedule	10 hours * 14 weeks * 2 periods
Group Size	4 [2-4]
Phases	Per.1 [Pha.1/6w + Pha.2/8w] Per.2 [Pha.1/8w + Pha.2/7w]
Main SWEBOK KAs	KA1/REQ + KA9/MOD + KA3/CONS
Main Market	Applications/Collaborative Applications/Team Collaborative Applications
Main Technologies	

Fig. 26 Reference Card of a SE Project Course Variant

Figure 26 provides a synthesis of the main project course characteristics obtained using the back&forth derivation process. It results in a project for bachelor ICSED level 655 that has two

periods of two phases spread over a full year program, focuses on the SWEBOK knowledge areas KA1 (Requirements), KA9 (Modeling) and KA3 (Construction). Figure 27 provides a detailed coverage of the SWEBOK knowledge areas for the Project course variant derived following a back&forth process which shows that a global coverage of 36% of the SWEBOK is obtained with one course.

Number	**Knowledge Area**	**Coverage** (%)
1	Software Requirements	80
9	Software Engineering Models and Methods	75
7	Software Engineering Management	67
11	Software Engineering Professional Practice	63
2	Software Design	46
3	Software Construction	39
8	Software Engineering Process	33
4	Software Testing	32
5	Software Maintenance	28
14	Mathematical Foundations	19
15	Engineering Foundations	18
10	Software Quality	17
13	Computing Foundations	11
12	Software Engineering Economics	8
6	Software Configuration Management	0

Fig. 27 Example of SWEBOK Coverage for SE Project Course Variant

For what concerns long-life learning, professional trainings for software engineering can be setup using the Messir approach presented in this article (two experiments will be made with industrial partners in the near future). It can be a solution to increase the knowledge level on the SWEBOK knowledge areas. Since the job offers in computing and mainly in software development will increase and be 3 times higher than the degrees awarded, the SE knowledge level might decrease if no life-long learning solutions are developed. The flexible scientific approach proposed by Messir is a limited but real contribution to raising the scientific level of software engineers.

In SE research, Messir can be used to tackle an interesting pool of open research problems mainly in the following areas: Domain Specific Languages, Model Driven Engineering, Specification based testing, Dependability requirements or Simulation and verification of modeling languages and, problem driven development using constraint oriented specifications.

12 Assessment of the proposed approach

In order to verify that the Messir approach really impacts positively requirements engineering, some experiments have been conducted. Those experiments have been made in academia at bachelor and master degrees level. During 3 years around 80 projects have been developed in which the requirements where engineered using the Messir approach. An analysis has been made in order to determine the benefits of using the Messir approach together with the Excalibur tool. The main facts noticed and verified for a large majority of the projects were:

– the use case instances are efficient to ensure a complete and consistent definition of the set of system operations.

- the scientific level choice allows to invest time coherently w.r.t. to the criticality of system operations.
- the unique capability of Messir to have simulation of axiomatic requirements impacts positively the reliability of the system by avoiding to implement invalid requirements.
- the Excalibur tool support increases the productivity for requirements analysis documentation by a factor ranging from 2 to 4.
- the average knowledge level acquired on the software engineering notions covered by the Messir approach is higher by 1 Bloom level [8] compared to project without the Messir method.

It has also been evaluated the Messep process to set up requirements engineering courses at bachelor and master levels at two universities (University of Geneva and Peter the Great Saint-Petersburg Polytechnic University). Thanks to the derivation processes, new courses have been defined in a short period of time (2 days), allowing to set up a new course improving the existing requirements engineering courses. Thanks to the learning material available the learning curve for the tutors has been reduced to a manageable amount which is critical in academic environment in which teachers have low time budget to re-engineers their lectures.

13 Conclusion

Software development is an activity difficult to master both technically and scientifically. This is due to many reasons such as the rapid development rate both of the technologies available, and of the quantity of product developed and requested by the society. The challenge for the software engineering domain is to produce SE theories, method and tools that are efficient for the industry needs, at the right cost, at the right time. Many unitary results exist but the complexity also resides in integrating those results, making them ready to use and, ensuring the necessary knowledge transfer. The Messir approach represents an important milestone in that direction. It succeeds to improve, adapt and integrate many of the SE theories, methods and tools and, offers an approach for efficient knowledge transfer. Among all the contributions brought, its main achievements are: an improved use case modeling approach, a declarative and executable requirement specification technique, a consistent integration of requirements analysis components using model driven engineering techniques, a flexible scientific engineering of requirements analysis, and a pragmatic knowledge transfer process for software engineering education based on the standard body of knowledge for software engineering (SWEBOK).

Current perspectives are to deploy the approach in education (a network of 12 partners is currently on development and new SE project courses are setup with the approach introduced in this paper) and industry (some professional requirement analysis training are on preparation for professional engineers). Concerning the evolutions of the Messir approach it is planned to develop a DSL for modeling non-functional requirements definition and to integrate the modeling of dependability requirements specifications. This will be done keeping the flexible scientific levels, which imply ensuring the soundness of the theories and the consistency of the methods and the tools.

Acknowledgements The author would like to thank all the persons that contributed to this work directly or indirectly since 1995. In particular Alfredo Capozucca and Benoit Ries, the designers and producers of nearly all the software engineering environment components for Messir , who are “highly guilty” for the existence and the quality of Messir .

References

1. Abrial JR, Hoare A (2005) The B-book: assigning programs to meanings. Cambridge University Press
2. ACM/IEEE (2015) Software Engineering 2014 - Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. ACM, New York, NY, USA, URL https://www.acm.org/education/SE2014-20150223_draft.pdf,https://www.acm.org/education/curricula-recommendations
3. Armour F, Miller G (2001) Advanced Use Case Modeling: Software Systems. Addison-Wesley, URL http://www.amazon.com/Advanced-Use-Case-Modeling-Addison-Wesley/dp/0201615924
4. Avgeriou P, Guelfi N, Medvidovic N (2005) Software architecture description and uml. In: UML Modeling Languages and Applications, Springer, pp 23–32
5. Bauer FL (1971) Software engineering. In: 1. Foundations and systems., International Federation for Information Processing: IFIP congress series, pp 530–538
6. Beck K (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
7. Bjørner D (1979) The vienna development method (vdm). In: Mathematical Studies of Information Processing, Springer, pp 326–359
8. Bloom BS, Krathwohl DR (1956) Taxonomy of educational objectives: The classification of educational goals. handbook i: Cognitive domain
9. Bollinger T, McGowan C (2009) A critical look at software capability evaluations: An update. Software, IEEE 26(5):80–83
10. Buchs D, Guelfi N (2000) A formal specification framework for object-oriented distributed systems. Software Engineering, IEEE Transactions on 26(7):635–652
11. Capozucca A, Guelfi N (2010) Modelling dependable collaborative time-constrained business processes. Enterprise Information Systems 4(2):153–214
12. Capozucca A, Ries B (2015) Website of the Excalibur Workbench for Messir. http://hera.uni.lu:8090/confluence/display/EXCALIBUR/Excalibur, accessed: 2015-08-05
13. Capozucca A, Guelfi N, Pelliccione P, Romanovsky A, Zorzo A (2006) Caa-drip: a framework for implementing coordinated atomic actions. In: Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on, IEEE, pp 385–394
14. CMMI Institute (2015) Cmmi appraisal results. https://sas.cmmiinstitute.com/pars/pars.aspx, accessed: 2015-07-05
15. Cockburn A (2001) Writing effective use cases. Addison-Wesley
16. Di Marzo Serugendo G, Fitzgerald J, Romanovsky A, Guelfi N (2007) A metadata-based architectural model for dynamically resilient systems. In: Proceedings of the 2007 ACM symposium on Applied computing, ACM, pp 566–572
17. eclipse (2015) The eclipse fondation. http://www.eclipse.org
18. G Di Marzo Serugendo and N Guelfi and A Romanovsky and A Zorzo (1999) Formal Development and Validation of Java Dependable Distributed Systems. In: Proceedings of ICECCS'99, pp 98–108
19. Gallina B, Guelfi N (2007) A template for requirement elicitation of dependable product lines. In: Requirements Engineering: Foundation for Software Quality, Springer, pp 63–77
20. Guelfi N (2011) A formal framework for dependability and resilience from a software engineering perspective. Central European Journal of Computer Science 1(3):294–328
21. Guelfi N (2016) Messir tutorial. Technical report, University of Luxembour, LU
22. Guelfi N, Ries B (2008) Selection, evaluation and generation of test cases in an industrial setting: a process and a tool. In: Practice and Research Techniques, 2008. TAIC PART'08.

Testing: Academic & Industrial Conference, IEEE, pp 47–51
23. Guelfi N, Sterges P (2002) Jafar: Detailed design of a pattern-based j2ee framework. In: Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA'02)
24. Guelfi N, Biberstein O, Buchs D, Canver E, Gaudel M, von Henke F, Schwier D (1997) Comparison of object-oriented formal methods. Tech. rep., Technical Report of the ESPRIT Long Term Research Project 20072: Design For Validation, University of Newcastle Upon Tyne, Department of Computer Science, 1997. http://lglwww. epfl. ch
25. Guelfi N, Astesiano E, Reggio G (2003) Scientific Engineering for Distributed Java Applications: International Workshop, FIDJI 2002, Luxembourg, Luxembourg, November 28-29, 2002, Revised Papers, vol 2604. Springer
26. Guelfi N, Ries B, Sterges P (2003) Medal: a case tool extension for model-driven software engineering. In: Software: Science, Technology and Engineering, 2003. SwSTE'03. Proceedings. IEEE International Conference on, IEEE, pp 33–42
27. Guelfi N, Muccini H, Pelliccione P, Romanovsky A (2008) SERENE'08: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems. Newcastle upon Tyne, United Kingdom. ACM, New York, NY, USA, URL http://doi.acm.org/10.1145/1479772.1479788
28. Heiman R (2010) Idc's software taxonomy 2010. International Data Corporation, Framingham
29. Humphrey WS (1989) Managing the software process. Reading, Mass: AddisonWesley
30. Hutchinson J, Whittle J, Rouncefield M, Kristoffersen S (2011) Empirical assessment of mde in industry. In: Proceedings of the 33rd International Conference on Software Engineering, ACM, pp 471–480
31. ISO/IEC (2005) Software Engineering – Guide to the Software Engineering Body of Knowledge (SWEBOK). International Organization for Standardization, iSO-IEC TR 19759-2005
32. ISO/IEC (2011) ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. ISO/IEC 13211-1
33. ISO/IEC (2012) Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure. International Organization for Standardization, iSO/IEC 19505-1:2012
34. ISO/IEC (2012) Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 2: Superstructure. International Organization for Standardization, iSO/IEC 19507:2012
35. ISO/IEC (2014) Software Engineering – Guide to the Software Engineering Body of Knowledge (SWEBOK). International Organization for Standardization, iSO-IEC TR 19759-2014
36. Jacobson I, Christerson M, Jonsson P, Overgaard G (1992) Object-oriented software engineering. a use case driven approach. [New York]: ACM Press; Wokingham, Eng; Reading, Mass: Addison-Wesley Pub,| c1992, Revised printing 1
37. Kienzle J, Guelfi N, Mustafiz S (2010) Crisis management systems: a case study for aspect-oriented modeling. In: Transactions on aspect-oriented software development VII, Springer-Verlag, pp 1–22
38. Kim D (2013) The state of scrum: Benchmarks and guidelines. 2013. Scrum Alliance
39. Kruchten P (2004) The rational unified process: an introduction. Addison-Wesley Professional
40. Li Z, Liang P, Avgeriou P, Guelfi N, Ampatzoglou A (2014) An empirical investigation of modularity metrics for indicating architectural technical debt. In: Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures, ACM, pp 119–128
41. Milner R, Parrow J, Walker D (1992) A calculus of mobile processes, i. Information and computation 100(1):1–40

42. Mussbacher G, Al Abed W, Alam O, Ali S, Beugnard A, Bonnet V, Bræk R, Capozucca A, Cheng BH, Fatima U, France R, Georg G, Guelfi N, Istoan P, Jézéquel JM, Kienzle J, Klein J, Lézoray JB, Malakuti S, Moreira A, Phung-Khac A, Troup L (2012) Comparing six modeling approaches. In: Models in Software Engineering, Springer, pp 217–243
43. NDIA Systems Engineering Division (2015) Cmmi status report. http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Past%20Meetings/Division%20Meeting%20-%20June%202011/NDIA%20SED%20June%202011CMMI%20Status%20v1.pdf, accessed: 2015-07-05
44. OMG (2014) Model Driven Architecture (MDA), MDA Guide rev. 2.0 ormsc/2014-06-01, Version 1.2. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf
45. OMG (2014) MOF 2.0 Core Final Adopted Specification. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf
46. Parnas DL (1997) Software engineering: An unconsummated marriage (extended abstract). In: Jazayeri M, Schauer H (eds) Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering, Zurich, Switzerland, September 22-25, 1997, Proceedings, Springer, Lecture Notes in Computer Science, vol 1301, pp 1–3, DOI 10.1145/267895.267897, URL http://doi.acm.org/10.1145/267895.267897
47. Pelliccione P, Guelfi N, Muccini H, Romanovsky A (2007) Software Engineering of Fault Tolerance Systems. World Scientific Publishing Co., Inc.
48. Perrouin G, Klein J, Guelfi N, Jézéquel JM (2008) Reconciling automation and flexibility in product derivation. In: Software Product Line Conference, 2008. SPLC'08. 12th International, IEEE, pp 339–348
49. Petri CA (1962) Kommunikation mit automaten
50. Pyster A, et al (2009) Graduate software engineering 2009 (gswe2009) curriculum guidelines for graduate degree programs in software engineering. Stevens Institute of Technology
51. Rodríguez P, al (2012) Survey on agile and lean usage in finnish software industry. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, ACM, pp 139–148
52. Sahami M, Roach S (2014) Computer science curricula 2013 released. Commun ACM 57(6):5–5, DOI 10.1145/2610445, URL http://doi.acm.org.proxy.bnl.lu/10.1145/2610445
53. Saidane A, Guelfi N (2012) Seter: Towards architecture-model based security engineering. International Journal of Secure Software Engineering (IJSSE) 3(3):23–49
54. Salman RH (2014) Exploring capability maturity models and relevant practices as solutions addressing it service offshoring project issues. PhD thesis, Portland State University
55. SEI CP (2010) Cmmi for development (cmmi-dev). Tech. rep., CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University
56. Sendall S, Perrouin G, Guelfi N, Biberstein O (2003) Supporting model-to-model transformations: The vmt approach. In: Workshop on Model Driven Architecture: Foundations and Applications; Proceedings published in Technical Report TR-CTIT-03-27, University of Twente, 2003
57. Serugendo GDM, Mandrioli D, Buchs D, Guelfi N (2002) Real-time synchronised petri nets. In: Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets, Springer-Verlag, pp 142–162
58. SICS (2015) Sicstus prolog - iso standard compliant prolog development system. https://sicstus.sics.se/
59. Sirius (2015) Sirius - the easiest way to get your own modeling tool. http://www.eclipse.org/sirius/

60. Spivey JM (1992) The z notation: a reference manual. international series in computer science
61. Stavru S (2014) A critical examination of recent industrial surveys on agile method usage. Journal of Systems and Software 94:87–97
62. UNESCO (2012) International Standard Classification of Education (ISCED) 2011. UNESCO Institute for Statistics, DOI 10.15220/978-92-9189-123-8-en, URL http://dx.doi.org/10.15220/978-92-9189-123-8-en
63. US Bureau of Labor Statistics (2015) Employment projections. http://www.bls.gov/emp/ep_table_102.htm, accessed: 2015-07-05
64. VersionOne I (2012) 6th annual state of agile survey. Tech. rep.
65. VersionOne I (2015) 9th annual state of agile survey. Tech. rep.
66. Xtext (2015) Xtext - language development made easy. http://www.eclipse.org/Xtext/=2