INSTITUT DE RECHERCHE EN INFORMATIQUE DE NANTES

Formal Methods Integration for Software Development: Some Locks and Outlines

Christian Attiogbé

Christian.Attiogbe@irin.univ-nantes.fr
IRIN, Université de Nantes
B.P. 92208, F-44322 Nantes Cedex 3

— Formal Specifications and Methods —

RESEARCH REPORT

N° 00.8

July 2000

IRIN, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3

Christian Attiogbé

Formal Methods Integration: Some Locks and Outlines - RR00.8 (December 1998, Revised:December 1999, January 2000, July 2000)

26 p.

Formal Methods Integration for Software Development: Some Locks and Outlines

Christian Attiogbé

Abstract

We present some locks related to the integration of software development methods. Here integration means putting together paradigms used by different methods in order to have an homogeneous environment suited for formal development. In a first part of the paper, after introducing formal methods in order to fix the vocabulary used in the sequel, we deal with the main difficulties for methods integration. This can be can be summarized by the following questions which are examined in the paper : What is method integration ? What methods can be integrated ? Does the integration always possible ? What are the practical locks for methods integration ? How to integrate methods ? What are the theoretical and technical solutions ?

In a second part, we present some outlines for methods integration through some elements to answer the above questions. They are : semantic domains as integration basis, domain compatibility, syntactical and semantic compatibility, refinement compatibility, specification bridges, integrated homogeneous reasoning context. Idea to build or generate integrated environments are sketched.

Formal methods[Integration]

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.4 [**Software Enfineering**]: Software/Program Verification; D.2.6 [**Programming Environments**]: Integrated environments; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Methods Integration, Specification, Development

Additional Key Words and Phrases: Formal specification, Formal Methods Integration, Compatibility, Integration basis, Specification bridge

Table des matières

1 Introduction **6**

2 Overview of formal techniques and methods **6**
- 2.1 Nature of systems, facets and paradigms . . . 6
- 2.2 Semi-formal methods . . . 6
- 2.3 Formal languages and Formal methods . . . 7
- 2.4 A layered view of languages and methods . . . 7
- 2.5 About methods classification . . . 7
- 2.6 On semantics models . . . 9
  - 2.6.1 General Semantic model . . . 9
  - 2.6.2 Other Semantic models . . . 9

3 Methods : characteristics, efficiency **9**
- 3.1 Characterizing methods : an overview . . . 9
- 3.2 Efficiency : What methods for what systems ? . . . 9
  - 3.2.1 What methods ? . . . 10
  - 3.2.2 What systems ? . . . 10

4 Some locks to formal methods integration **10**
- 4.1 About compatibility between methods . . . 11
  - 4.1.1 Heterogeneous Syntax . . . 11
  - 4.1.2 Heterogeneous semantics and formal reasoning . . . 12
  - 4.1.3 Heterogeneous paradigms and formal reasoning . . . 12
  - 4.1.4 Tools level . . . 13
- 4.2 Practical integration cases examination . . . 13
  - 4.2.1 Case 1 : Data with Data Methods . . . 13
  - 4.2.2 Case 2 : Data with Process . . . 14
  - 4.2.3 Case 3 : Process with Process . . . 14
- 4.3 Concluding remarks on locks . . . 15

5 Integrating methods : some outlines **15**
- 5.1 Formal basis for integration . . . 15
- 5.2 Domain compatibility and Integration Domain . . . 16
- 5.3 Syntactic compatibility domain . . . 17
- 5.4 Semantic compatibility domain . . . 17
- 5.5 Methods compatibility . . . 17

6 Specification Bridges for integrating methods **17**
- 6.1 The concept . . . 17
- 6.2 Syntactic level . . . 18
- 6.3 Semantical level . . . 18
- 6.4 Formal reasoning . . . 19
- 6.5 Generalization for integration approach . . . 19

7 Appraisal of the work **20**
- 7.1 Related works . . . 20
- 7.2 Lessons for integrating methods . . . 22

8 Concluding remarks and future works **22**

1 Introduction

Methods integration is a promising approach for complex software development. Methods integration concerns the combined use in a unique environment of different methods which when considered independently are not sufficient to tackle the multiple facets (data, reaction, behaviour and concurrency, etc) of real life systems but when considered together can give interesting results. A lot of real life systems have several facets. For long time, development techniques (including languages, methods, environments) focus on one of the facets and are strengthened according to this facet (Structured Analysis, Abstract Data Types, Process Algebra, etc). This results on a partial treatment of software. Nowadays a lot of works focus on the integration of several approaches in a unique method or environment in order to provide appropriate techniques and tools for software development. Historically, JSD [Jac86] which integrates Structured Analysis and CSP, has shown the feasibility of the integration of methods. LOTOS [ISO89], [GS90] follows this approach of integration by combining Act One and CCS with some features of CSP. RAISE [Gro95a] which integrates VDM and Petri Nets had also used such approach of integration.
Immediate questions related to methods integration are : *can we integrate any method with any other one ? What are the main problems to be solved for methods integration ? What are the limits for methods integration ? How to formally reason about system within integrated environment ?*
In this work we examine some of these questions. We look at some means to check if given methods *can* be integrated and *how* they can be integrated. The major results of this work should be experimented by (the generation of) integrated environment. According to methods characteristics, it should be possible to generate (i) obligations for integration basis, (ii) parts of environment suited for formal reasoning and more generally formal development. By the way, integrated environment can be flexible and parameterizable. Recent works on integrated approaches exist. Most of them mainly dealt with formalisms combination : [EO94], [Gal96], [Fis98], [CPR99], [Pai97], [Smi97], [Att98], [Fis97], [MD98] ; Formal reasoning tools integration are treated in [Hun93].

The paper is organized as follows : the section 2 gives a general overview of formal techniques and methods in order to fix the vocabulary and related concepts used in the paper. In section 4 we give some methods characteristics used in the paper for the integration purpose. Section 4 discusses some locks for methods integration and the examination of some paradigms integration cases. In section 5 we propose some solutions to deal with the locks. Section 6 gives some details on specification bridges and section 7 discusses related works and lessons for methods integration.

2 Overview of formal techniques and methods

2.1 Nature of systems, facets and paradigms

Software systems are naturally different from one to another. Apart from their application domains, they can be : sequential, parallel or concurrent, information systems (databases), autonomous, reactive, real-time, protocols-oriented, centralized, distributed, etc.

Accordingly, one distinguishes some *facets* to characterize systems : *data, functions, processes, protocols, real-time, etc*. *Paradigms* are related to these facets and qualify the mechanisms used by formalisms or methods to deal with systems. Therefore, the main paradigms used for methods and formalisms are : *data, operation, process(concurrency), real-time, reactive*.
The preliminary analysis of a software requirements help to determine these facets and the paradigms required. In the software life-cycle, formal methods and techniques find their places after the analysis step.

2.2 Semi-formal methods

A lot of methods have been proposed for mastering software systems development life cycle. Most of the time they do not cover all the steps of the life cycle, but one or several steps. Some examples of these methods are : SADT, SA-RT, SSADM, JSD-JSP [Jac86], Merise, Axial, OOA, OMT, UML.

They help in analyzing the requirements of the system to be developed. The result of the analysis is a semi-formal specification ; that means they include both formal notations, informal notations and graphical notations, and cannot be given formal semantics. The problem is that a specification which is not formal does not have a precise semantics and can lead to confusion and ambiguities. Therefore, semi-formal specifications can not be used as is for formal reasoning. They are not a good starting point for formal development. These lacks motivate some research works on the formalization of semi-formal methods.

2.3 Formal languages and Formal methods

Some introductions on formal methods can be found in [Win89], [CGR95], [Jon86], [cM96].

A *formal language* is a language with precisely defined syntax and semantics.
Some examples are : Logic formalisms, set-theory languages, Z [Spi92], Algebraic specification languages [BGM89], [BKL+91] (CLEAR [BKL+91], ASL [Wir86], Act One [EFH83], PLUSS [BKL+91], LARCH [GG89], CASL [Gro], etc). They are used to write specifications which describe what a system is (a model) or what a system does (a behaviour).

A *formal system* is : a formal language and a proof system (or so called formal reasoning system).

A *formal method* is : a formal language and a formal reasoning system or proof system or refinement system.

Some examples are : CSP [Hoa85], CCS [Mil80], OBJ [FGJM85], LP [GG89], VDM [Jon90], RAISE [Gro95a], Cleanroom [Dye92], B [Abr96].

Formal reasoning

It consists in applying a formal system to a specification. Refinement of specification is a formal reasoning. The verification of system properties is a formal reasoning. The validation by verification is a formal reasoning. Proving theorems is a formal reasoning. Analysing a state machine against given properties (called model checking) is a formal reasoning.

Formal development

Development covers the stages between informal requirements and executable codes. It includes formal reasoning steps.

Formal development is : systematic transformation of specifications into executable codes using predefined rules (rewriting, stepwise refinement, synthesis). There are a lot of approaches in formal development :

- *operational approach* : system development is based on the description of a model which has the properties of the system to be developed.
- *axiomatic approach* : development is based on an axiomatization that describes the behaviour of the system.
- *algebraic approach* : development is based on algebraic description of the properties of the operations of the system without the description of states.
- *hybrid approach* which combines operational with one of the other.

The steps in formal development are combined or followed by analysis done by checking the properties of the system. This is done by two main approaches, theorem proving (proving properties directly on the specification of the system) and model checking (checking in an exhaustive way the state graph of the system against some properties).

2.4 A layered view of languages and methods

The figure 1 summarizes the idea. The point is that formal development methods are all based on the use of a specification language and formal reasoning tools and are used to build systems. Formal reasoning methods are based on theories or formal languages or formal specification languages and are used to analyze models and systems. Formal specification languages are all based on theories or formal languages and are used to build models.

2.5 About methods classification

Here, the term *method* stands for both formalisms and methods. According to the paradigms they follow, methods are said to be *data-oriented, operation-oriented, process-oriented.* A first approach of classification is

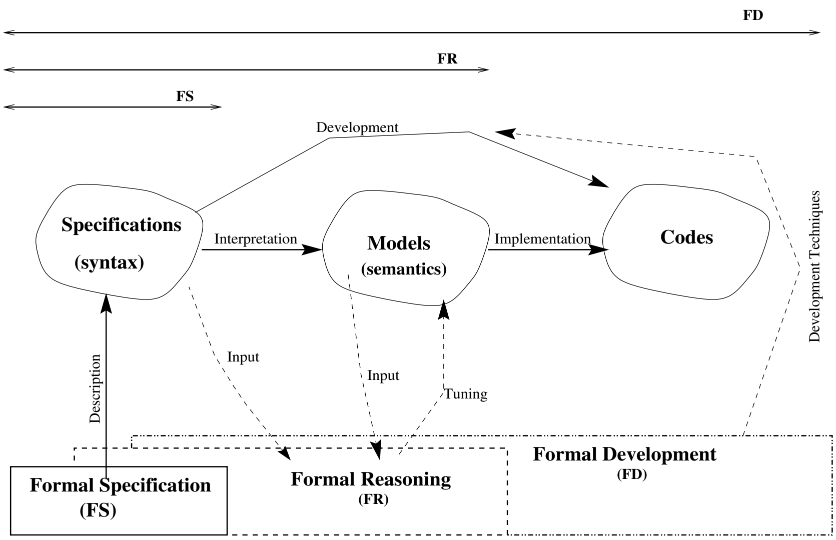

FIG. 1 – A Layered View of languages and methods

related to these paradigms. This first approach is widely admitted [CW95], although perfectible. Other criteria can be used to classify methods for examples : semantic models used, formal reasoning issues, application domains covered (real-time, reactive, etc), expressiveness.

State-oriented (or Model-oriented)

Data-oriented and *process-oriented* methods are classified in this approach. The principle is that the specification language used allows to describe a model of the system to be developed. This model has the properties of the system and reasoning on the system is done through the model.

Examples of formalisms in this approach are :

- Z[Spi92], VDM [Jon90], B [Abr96], Finite State Automata [AVA95] for sequential systems,
- Finite State Automata, Petri Nets [Rei85], CCS [Mil80], CSP [Hoa85], Unity [CM88] for concurrent and distributed systems. (CCS and CSP are also process-oriented)

Property-oriented (axiomatic and algebraic)

They are also called operation-oriented. Here specifications provide an axiomatization which defines the properties of the system to be developed.

Examples of formalisms used here are :

- Algebraic specification languages : ASL [Wir86], LPG [BE95], ACT ONE [EFH83], CLEAR [BKL+91], OBJ [FGJM85], PLUSS [BKL+91], Larch/LP [GG89], CASL [Gro].
- Process algebra : LOTOS [GS90]

Hybrid approaches

Methods that integrate other state-oriented methods or process-oriented or property-oriented methods are called hybrid methods. The hybrid approach can be considered as the basis for methods integration.

2.6 On semantics models

Semantic models give *meaning* to syntactic constructions described by formal specifications written in some formal languages. The more basic example of semantic model is an interpretation given to a logic formula considered as a specification. More generally, a state machine can be an interpretation for a formal specification ; an algebra can be the interpretation for a specification.

2.6.1 General Semantic model

Semantic model for sequential systems

Sequential systems have been well studied and formal models are well known for different levels considered :
- Turing Machines [Spi96, AVA95] and RAM [J.98] at a very concrete level : algorithms or programs,
- Logic Models, Finite State Machines, Algebra, Labelled Transition Systems at a more abstract level : specifications.

Semantic models for concurrent systems

The Petri Nets Theory generalizes the idea of state machines to deal with concurrency and furnishes means to model concurrent systems. The Labelled Transition Systems (LTS), the synchronized product of Finite State Machines (FSM, Arnold-Nivat) are other widely used models for concurrency. Through CSP and CCS appear other models based on the composition of sequential processes with related notions of traces, bisimulation [Mil89] and the introduction of Failure-Divergence [Ros98].

Therefore the main models for concurrency are : Petri Nets, FSM, LTS, Product of FSM, Traces, Failure-Divergence, Bisimulation.

2.6.2 Other Semantic models

Milner had extended the work on process algebra (CCS, SCCS) and had proposed the $\pi-$calculus [RJD92], this remains in the scope of process algebra models (CCS, CSP, ACP).

Event structures [NPW81] constitutes another approach where systems are represented using a set of events together with relations on the events denoting action occurrences.

3 Methods : characteristics, efficiency

3.1 Characterizing methods : an overview

A lot of characteristics can be used to distinguish methods. Here we choose to use in one hand syntactic characteristics and semantic characteristics in order to underline the main problems for methods integration and in the other hand formal reasoning in integrated environment. By syntactic characteristics we want to explore the means provided by the method to describe formal specifications or constructions. By semantic characteristics we explore the means provided to give meaning to the specifications. The other characteristic is formal reasoning techniques used by the methods.

The table 1 introduces the *heterogeneity* of formalisms and theoretical fundaments behind methods. This heterogeneity is the basis of locks on methods integration.

3.2 Efficiency : What methods for what systems ?

The table 1 gives a rapid overview of the variety of methods one can use for system development. The question is, which one is suitable for a given system ?

Note that formalisms are treated here as methods, considering that the term *method* can cover the large meaning of formalisms and techniques.

Formalisms	Theoretical basis	Semantic Models
Z	1st order logic, Set Theory	State Models
B	1st order logic, Set Theory, AMN	State Models, Predicate transformers, WP
TLA	Temporal logics, Set Theory	State Models, Temporal logics
CCS, CSP	processes, events, channels	Traces, Bisimulation, Failure-divergence
π-calculus	processes	Transition systems, Bisimulation
Pluss, LP	1st order logic, Universal Algebra	Algebra
Meije/Esterel	Processes	Synchronous Calculi
Signal	processes, Equational Logic	Traces, Algebra
HOL	Equational logics	Higher Order Logics, *
Coq	Equational Logics, Types	Constructive Types Theory
PVS	Equational Logics	Higher Order Logics, *

TAB. 1 – Formal methods, fundamentals and semantics

3.2.1 What methods ?

According to the systems one has to develop it should be clear to envision a family of formalisms, techniques and tools. This is quite clear for some formal methods specialists. However, most of the time and unfortunately, developers try to use their preferred methods or techniques to deal with any system.

For non-specialists, it is not so clear to address a family of appropriate formalisms, appropriate techniques and tools, to the systems they have to develop. Therefore it is the task of specialists to guide the analysis of the system and to guide the development (including the specification) in the right manner.

The right method is that appropriate to the nature of the system (requirements) one has to develop.
This means that the choice of a method depends on the requirements.

3.2.2 What systems ?

Real life systems do not separate between particular paradigms (or facets). Most of the time, a given system includes aspects dealing with several paradigms. In the sequel, the term *facets* is used to designate aspects related to paradigms, and *multifaceted systems* are systems which have several facets.

Therefore, it is neither easy nor realistic to undertake the formal development (including analysis, specification, design, verification, simulation, implementation, tests), of a system using only one formalism or paradigm. The fact is that the nature of complex software systems obliges the developer to use the appropriate formalisms, techniques and tools, in the right steps of the development process. That raises the needs of integrating several approaches for the same system : *methods integration* and more generally multiformalism specification.

As systems are multifaceted, *methods* should be integrated to efficiently guide rigorous development. However, methods integration raises particular problems which should be solved before applying it in real cases.

The right system is that integrating all the facets of the requirements. This means that the system expected depends on the capabilities of the *right* methods used.

4 Some locks to formal methods integration

Formal development by methods integration approach is not straightforward. Indeed, one faces mainly the problem of *heterogeneous syntax, heterogeneous semantic and reasoning techniques.* These problems have to be well stated and solved partially or totally in integrated environments.

4.1 About compatibility between methods

The main problem can be summarized as the existence of an abstract theory to reason about the relationships between the other theories underlyin the combined methods.

When integrating two or several methods, a problem is that, each of them comes with its own characteristics, syntax and semantics. The characteristics of the method can be underlined by the facet(s) of system this method can be used for and the formal reasoning techniques used. The syntax is related to the way the specifications are built in this method. It provides a syntactic support as a formal language. Most of the time, logics and set theory are the main parts of the syntactic support. The semantic is related to the models introduced by the constructions written with the syntactic support of the method. This aspect is related to the theoretical foundations of the method. Semantical models encountered are : logical models, algebra, traces, transition systems, etc.

Definition 1. *(Ambiguity)*
There is semantic ambiguity *if a given construction (appearing in different formalisms) can be given different semantic according to the formalism considered.*

There is syntactic ambiguity *if a given construction (used in two different formalisms) have different usage in each formalism, thus different semantic.*

Semantic and syntactic ambiguity are joined.

Definition 2. *(Preliminary)*
For a given method M_i, we associate a language $L_i = L(G_i)$ with G_i the grammar corresponding to L_i.

Definition 3. *(Compatibility)*
Two (or more) formalisms are compatible if they have no ambiguty.

Syntactic Level Compatibility : two methods M_1, M_2 are syntactic-level compatible, if their languages L_1 and L_2 are compatible.

Semantic Level Compatibility : two methods M_1, M_2 are semantic-level compatible, if their semantic models Sem_1 and Sem_2 are compatible.

Therefore, when integrating methods, the side by side compatibility between these aspects considered for each method should be a main concern.

We can now list some points considered as possible locks for methods integration. The characteristics (data, process or behaviour) of the methods can be transversal with regard to these points.

- Considering the theoretical fundamentals of methods, it appears that some are incompatible. That means there are contradictory choices in their fundamentals.
- Methods basis can be so far from each other in their approaches so that they can not be useful or efficient when used together.
- We can have no commonalities between theoretical fundamentals, but no divergences (syntax and semantics).
- We can have no commonalities between theoretical fundamentals and divergences (syntax and semantics).
- More generally, how compatible are their characteristics according to formal reasoning and formal development.

4.1.1 Heterogeneous Syntax

There are practically some possibilities : we can distinguish a main formalism and a secondary formalism or several secondary formalisms. We can also have the case where the different formalisms are of the same level.

- When a formalism is given as the main one and the other a secondary, the constructions of the secondary formalism can be translated into the first ones or the main one is augmented by the constructions of the secondary for example by extending the typing scope of some operators. Possible lock is : *compatibility* between the constructions of the two formalisms.
- In the case where the formalisms are all treated of the same level, a third formalism is needed to integrated the two others. This third formalism should be a target language for a translation from the two others or one language is completly embedded in the other. Here we note also *compatibility problems* : is it possible to have the third language for the two considered languages ? Is it possible to translate completely into the third (without lost) the constructions of the different languages ?

Compatibility between the constructions of the languages

The problem here can be stated more precisely by simplifying it and considering formal languages defined by grammars. Consider two languages L_1 and L_2 so that $L_1 = L(G_1)$ is the language generated by the grammar G_1, and $L_2 = L(G_2)$ is the language generated by L_2, where G_1 and G_2 are the grammars associated to the languages. The table 2 gives a summary.

Constraints on L_1, L_2	Observations	Possibility for integration
$L_1 \cap L_2 = \varnothing$	No ambiguities	augmentation, $L_1 \cup L_2$
$L_1 \cap L_2 \neq \varnothing$	Possible Ambiguities	Solving compatibility on $L_1 \cap L_2$
$L_1 \cap L_2 = L_1 \; or \; L_2$	subsets, syntactic inclusion	No integration problem unless parts of the subset have different semantic
$\exists\, T \mid L_1 = T(L_2)$	T can be syntactic (or semantic) transformation	*Bridge* Definition (syntactic and/or semantic)
$L_1 \cap L_2 = \varnothing$, $\exists\, sL_1 \subseteq L_1$, $\exists\, sL_2 \subseteq L_2$, $\exists\, T \mid T(sL_1) = sL_2$	T can be syntactic (or semantic) transformation	*Bridge* Definition (syntactic and/or semantic)

TAB. 2 – Synthesis on syntactic heterogeneity

$L_1 = T(L_2)$ stands for the case where one ($L1$) of the two languages is obtained by transforming (syntactically or semantically) the other (L_2). Therefore, w.r.t. to this transformation, it is possible to define a *bridge* between the two languages.

The other cases are when some subsets of a language are the same (perhaps by renaming) as subsets of the other language. For example an operator written differently in the two languages. A simple example can be the logical operator written `or` in one language and written $\wedge$ in the other.

4.1.2 Heterogeneous semantics and formal reasoning

Two points of view can be undertake. One is language point of view. The other is a logic point of view.
As treated in the previous section on syntactic level, it seems clear that when integrating several semantic models one will face the problem of *semantic compatibilities* : The semantics and more generally the formal basis of the methods can be the same, completely different but consistent, completely different but not consistent, different but sharing some commonalities. The table 3 gives a summary.

Let $sL = L_1 \cap L_2$; Sem_{L_i} means the semantics w.r.t.. to the language L_i. The table 3 summarizes the idea.

In the logic point of view, moving between logics, for example the transfer problem [dR99] is a lock.

4.1.3 Heterogeneous paradigms and formal reasoning

Mainly, one needs to integrate paradigms : *data* into *behavior-oriented* formalisms or conversely to integrate *behavioural aspects* into a *data-oriented* formalisms. These cases of integration should treat the problem of complex systems but raise the problems listed before : heterogeneous syntax, semantics and reasoning tools.
The combination or integration of paradigms can then be considered as the high level locks for methods integration. The subsequent problem is that of formal reasoning. Consider for example refinement, it will not be always possible to refine, in the same manner, specifications from different paradigms. This lock can also be noticed at properties proving level.

A hint is that, the compatibility should be extended to formal reasoning and development : refinement, verification, simulation, synthesis. The state of current works shows that for some purposes, temporal logics are useful and

Constraints on $L1$, $L2$	Semantic constraints	Guides for integration
$sL = \varnothing$ No ambiguities	Semantics can be incompatible according to the models used.	OK if compatible semantics. To be studied if incompatible models.
$sL \neq \varnothing$ Possible Ambiguities	$Sem_{L_1}(sL) = Sem_{L_2}(sL)$	OK. They are compatible.
$sL \neq \varnothing$ Possible Ambiguities	$Sem_{L_1}(sL) \neq Sem_{L_2}(sL)$	They are incompatible. Ambiguities
$sL = L_1 \text{ or } L_2$		No integration problem if $Sem_{L_1}(sL) = Sem_{L_2}(sL)$. Otherwise find a solution
$L_1 = T(L_2)$		Compatible. Need of *Semantic bridge* Definition.

TAB. 3 – On semantic compatibility

are integrated with different approaches. Higher order logic tools also work well for some purposes. It is important to note that policies to reason on behaviour (safety, liveness, equivalence, fairness) are different from policies to reason on data and functions (axiomatization, refinement).

Then, there is a need of an appropriate logic relating the logics behind different paradigms and allowing reasoning on these logics.

4.1.4 Tools level

This level is related to the previous section since tools follow reasoning policies. Therefore the compatibility problem is also raised in the level of tools : how does tools for different methods/techniques can interact ?

Then compatibility should be extended to appropriate tools for the manipulation of different methods considered.

In a practical level, formal development within an integrated environment should provide means to deal with the different paradigms evolved by the requirements considered.

Some solutions are raised in the sequel for these main locks for methods integration.

4.2 Practical integration cases examination

In this section, we consider an almost exhaustive examination of integration cases. The aim is to have a good appreciation of problems which arise and in which cases.

For this purpose, we follow the paradigms underlying the formal methods : *Data (by state), Data (by Operation) and Process (including Transition Systems).* Accordingly, model-oriented methods, property-oriented methods are in the **Data** category whereas process-oriented methods are in **Process** category. Transition systems can be considered in the Process category. We do not include **Time** since this characteristic is transversal to all other, and the principle of the study should remain the same with time characteristic.

All other considerations can be reorganized around Data, Operation and Process. For each case it suffixes to consider two methods or formalisms for integration and the study is sound for more than two languages.

4.2.1 Case 1 : Data with Data Methods

We give some examples to illustrate each case considered and then we look at the possible syntactic and semantic problems.

State *with* Operation	:	Z or B *with* Algebraic formalism
State *with* State	:	Z *with* B
Operation *with* Operation	:	Algebraic *with* other Algebraic formalism

Notes on syntactic aspects

In this case of integration, a formalism or method can be chosen as the main one and the other a secondary. In this case the secondary language constructions can be translated into the first ones or the main is augmented by the constructions of the secondary. Possible locks are : *compatibility* between the constructions of the two languages. An example is the translation from an algebraic specification to a B specification (Ken Robinson, [**?**]).

An other choice is that none of the two languages can be considered as the main. In this case, a third language is needed to integrate the two others. This third language should be a target language for a translation from the two others. Here we note also *compatibility problems* : is it possible to have the third language for the two considered languages ? Is it possible to translate completely (without lost) the constructions of the different languages ?

Notes on semantic and formal reasoning aspects

Typically, in the semantic level, when integrating several data-oriented formalisms, one have to operate the translation of semantic structures from one to the other and to keep only one language level of reasoning. Most of the time, an extension of one of the formalisms is achieved.

4.2.2 Case 2 : Data with Process

This case can be seen as the most interesting integration case. A lot of work have been done in this area since several years : LOTOS [ISO89], Raise [Gro95b, Gro95a], Dynamic Data Types [EO94], Z-CCS [Gal96], [GS97],[TA97], [Fis98], Z-CSP [RWW94, Fis97, Smi97],[Smi97],[Fis97], [MD98], B-CSP [But99, But].

State *with* Process	:	Z or B *with* CCS or CSP or Petri Nets
	:	Z or B with Transition systems
Operation *with* Process	:	Alg. formalism *with* CSP or CCS or Petri Nets
	:	Alg. formalism *with* Transition Systems

Notes on syntactic aspects

A method is used as the main method and the other as the secondary method. For example a process algebra can be the main method and the data oriented-method used as the secondary method. This is efficient for value passing to process algebra.

The data-oriented method can be used as the main method and the process method used to describe possible behaviour for using operations in the data level. For example a data type is described and then its operation names are used as the elements of processes alphabet. Here heterogeneous syntax is not so tedious and there are not very difficult problems of compatibilities. The remaining problem is that of expressiveness.

Notes on semantics and formal reasoning aspects

Compatibility problem have to be studied like in table 3. Briefly, the main problem to solve is embedding one formalism to the other or extending a formalism by considering a paradigm not initially included within it.

4.2.3 Case 3 : Process with Process

The process-oriented approach is more stable than the other. Indeed, CSP (Hoare), CCS(Milner) and ACP(Bergstra) are the main process algebra and all other are built on their basis. Therefore, there are some common fundamentals. The differences between process algebra are most of the time on the level of expressiveness, model of communication (synchronous or asynchronous), time treatment, etc.

LOTOS is an example of formalism combining two process-oriented formalisms (CCS and CSP).

Process *with* Process	:	CSP *with* CCS or RdP
	:	CCS, CSP *with* Transition systems

Notes on syntactic and semantic aspects

From the syntactic point of view, process algebra are not so different. The compatibility problem is less accurated. Some operators from one formalism can be added to the other formalism.

Most of the time, the traces semantics is used. The other models are : Failure-divergence (CSP), Bisimulation (CCS) ; algebraic laws (CSP) or semantic rules (CCS) are used to describe behaviour of operations.

4.3 Concluding remarks on locks

To conclude this section on locks, we can notice that the integration of paradigms *data, behaviour, functions, time* is the basis of the locks.
Integrating some of these paradigms raises the concrete problems of : heterogeneous syntax, heterogeneous semantics and the related problem of formal reasoning and compatibility between objects (syntactic or semantic) manipulated.

5 Integrating methods : some outlines

We develop in this section some idea for building *multiparadigm-integrated method.*

According to the locks described, we need to precise *compatibility between methods* which is an important notion in the work.

Definition 4. *(Compatible Methods)*
Two methods M_1 and M_2 are compatible if they are three levels compatible :syntactically compatible, semantically compatible and Formal-Reasoning compatible.

It will not be possible to integrate non-*compatible* methods or the cost for doing that will be prohibitive. The very simple way is to base integration on *compatible methods*, this implies syntactic, semantic and formal reasoning levels. *Integration basis* and compatibility can help in integrating methods. We suggest an approach of integration based on the consideration of *semantic basis* to integrate methods.

5.1 Formal basis for integration

When integration is possible, according to compatibility, another difficulty is to find a *convenient homogeneous basis* in order to reason on objects initially attached to formalisms or methods ; in the sequel we consider for this purpose : *integration basis* and *integration domain*. It is an aswer to the question to find a convenient abstract level to reason on different logics.

Definition 5. *(Semantic Integration Basis) A semantic integration basis for given methods is a semantic model that enables formal reasoning on (parts of) the given methods : it can be seen as a common compatibility level or a more abstract logic, within which reasoning can be done soundly on the given methods.*

Therefore a semantic integration basis can be included in an other one and can share characteristics with others.

An example of semantic integration basis is first order logic ; indeed first order logic enables reasoning on several formalisms which use logic for objects description. Consider the case where two object descriptions (from different languages perhaps following different paradigms) have been translated as predicates in first order logic ; Then these resulting predicates can be combined by the classical operators of first order logic.

If such integration basis exists, it will be always possible to translate objects semantics (from one formalism and paradigm) into the basis, to integrate them and to reason in the context of the basis (and to possibly translate results in target formalisms).
Higher-order logics are good candidates for integration basis. More generally, we can consider for illustration and experimentation purposes : HOL ([Gor93] M. Gordon), Isabelle [Pau93b],[Pau93c], Type theory [], Category theory [].

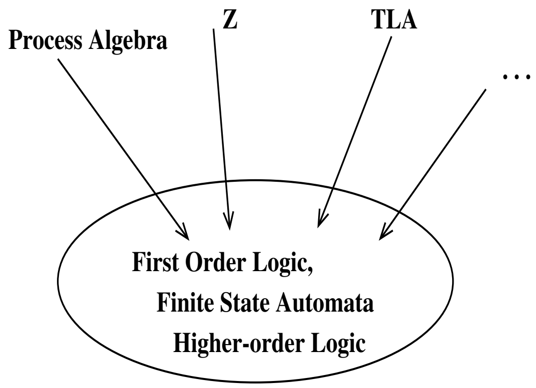

FIG. 2 – A simple integration basis

On this basis, a new integrated method is available for the user and the advantage is that it can cover several paradigms but with a homogeneous context.

Therefore, we need the definition of *bridges* between methods in syntactical and semantic level ; then we introduce : *specification bridges*.

Roughly, the principle of a bridge is to translate (embed) the model underlying a technique/method to a given model or a language which can be used in the integrated level.

A direction of future works is to consider classes of methods and to consider a minimal and common formal basis for integrating methods. By the way, we can suggest guides for method integration : level of confidence to a given integration approach, recommendations for some integration approaches, prohibition of some other integration approaches.

5.2 Domain compatibility and Integration Domain

Here and in the sequel the idea of compatibility, gives an achievement to the idea of classes of methods sharing the same logic or sharing the same paradigms.

Definition 6. *(Compatibility Domain)*
A Compatibility Domain is defined as a category of methods characteristics such a way that, any two methods considered within this category, are compatible (w.r.t. a logic or a paradigm).

Here are some *basic compatibility domains :*
- *semantic models* : logics, labelled transitions systems, traces semantics, temporal logics, weakest preconditions, Kripke model, algebra ;
- *syntactic descriptions* : grammars, logics, set-theory formalism ;
- *application domains* : sequential, distributed, real-time, reactive, concurrent.

Definition 7. *(Integration Domain)*
An Integration Domain is an integration basis with respect to domain compatibility and with possibilities of formal reasoning and formal development.

The practical idea is that the *integration domains* for method integration should be more general than the basic domains listed above : they should provide *homogeneous framework* for formal development and formal reasoning on integrated methods.

Accordingly, relationships between domains should be studied : does a method can belong to more than one domain ? In case of positive answer, how to treat the transitivity ?

Domains	methods in the domain
D1	M1, M2, M4
D2	M3, M5
D3	M1, M6

M1 is compatible with M4 because they are in the same domain D1, M6 is compatible with M1, does M6 and M4 are compatible ?

5.3 Syntactic compatibility domain

Proposition 1. *Compatible syntactic domains*
All languages which use the same logic to describe their constructions are in the same domain.

5.4 Semantic compatibility domain

The purpose is the semantic compatibility according to semantic models.

Proposition 2. *(Compatible semantic domains) All methods using the same semantic models are in the same semantic domain.*

Therefore considering two methods in a given compatible domain, they are compatible an can be integrated.

The study of the different integration cases in the previous section shows that, the integration of process paradigm and data paradigm, by using one as the main and the other as the secondary, seems to be the interesting case.

5.5 Methods compatibility

The compatibility relation can be extended to development : data-refinement, operation-refinement. Refinement and synthesis are the main techniques used for development. Then we introduce *refinement-compatibility* relation between methods.

Definition 8. *(Formal Reasoning(FR)-compatibility)*
Two or more techniques/methods are FR-compatible *if they use the same Formal Reasoning technique (refinement, theorem-proving, etc).*

Proposition 3. *(Refinement-Compatibility)*
Two or more methods are refinement-compatible if they enable one to refine specifications by the same refinement techniques.

The *B method* [Abr96] and *Action Systems* [BKs83] are refinement-compatible. Indeed, considering operations, these two methods use weakest preconditions to model operations and use the same refinement techniques.

6 Specification Bridges for integrating methods

In this section, we consider one of the steps toward methods integration. Indeed in the case of the integration of existing methods, a preliminary step is their languages integration which is practically achieving bridge between specifications. At semantic level, the bridge is between models underlying the specifications and at method level the bridge enables effective integration. The idea is that, when methods are compatible, we can define bridge between them in order to follow reasoning within one of them or in a more abstract third formalism.

6.1 The concept

Associated to formal methods, we have specification languages. The concern is to establish a bridge between two formal specification languages in the purpose of their integration. Therefore, from a specification in one language we can get a corresponding specification in the other language. At least three levels can be distinguished : syntactic, semantic and formal reasoning. The figure 3 summarizes this idea.

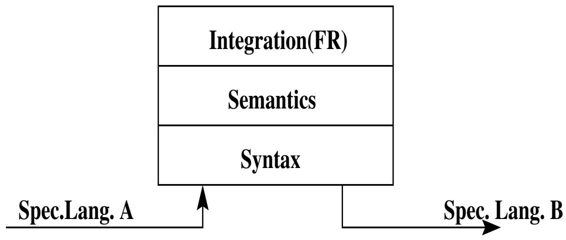

FIG. 3 – A Specification Bridge

6.2 Syntactic level

A specification in a language can be :
– translated into a specification in the other language,
– encapsulated by a specification language in the other language

An example is : going from a Z specification to a specification in a fancy language FL which encapsulates the constructions of Z.

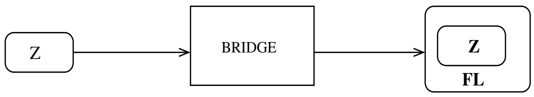

FIG. 4 – Z and FL

6.3 Semantical level

A logic (or an appropriate langage) is used to express the semantics of involved formalisms. For example HOL to express the semantics of Z or HOL to express CCS or CSP semantics. This is not a new idea ; it is the *embedding* of formalisms [BGG+92]. There are two main techniques of embedding : *shallow embedding* and *deep embedding*. The first technique deals with the translation of objects of a formalism to semantically equivalent object in the target formalism. In the case of deep embedding, the complete semantics of a formalism is translated in the other formalism. The choice of one of the techniques depends on the pursued goal.

The bridge has the general form described in figure 5.

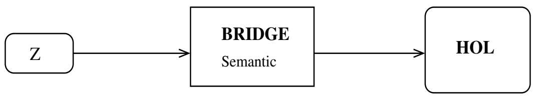

FIG. 5 – Z to HOL

Some studies exist on these topics.
For HOL-Z, a lot of references exist. In [KW96], the authors represent the semantic of Z within the framework Isabelle/HOL. In [San98], the author investigates the relation between the semantics models of Z and HOL. In [BH94],[BG95] relation between Z and HOL is studied and then the embedding of Z in HOL logic.

For HOL-CSP [TW97], the authors give a Failure-Divergence model of CSP within the framework of Isabelle/HOL.

For HOL-CCS, [Nes93] presents the embedding of CCS in HOL.

For PVS-B [Muñ99], the author deals with a support for the B-Method in PVS. It is an embedding of the abstract machine notation of B in the PVS logic in order to assist proofs checking and to provide higher-order notation. Another related work around B is done with PVS and Coq in [BF00].

For HOL-B [Cha98], the author describes a semantic embedding of the basic concepts of the B language in the higher-order logic instance of the generic theorem prover Isabelle (Isabelle/HOL).

6.4 Formal reasoning

Formal reasoning can be tackled within :

– the high-level language used in the semantic level or
– an appropriate logic defined (by extending already existing logics, modal logics, etc) for this purpose.

According to this latter level, which is the basis for the integrated method, the user can be provided with tools for writing specifications and reasoning about them.

6.5 Generalization for integration approach

In the purpose of integration, it is of some interests to use higher-order logics, or higher-order theories in the bridge between methods and for formal reasoning.

At the syntactic level, a language like HOL or PVS logic can be chosen as high-level to encapsulate, w.r.t. to compatibilities constraints, one or several languages (chosen as low-level languages) to be integrated. In this approach, the low-level language constructions should be translated to, or encapsulated by the high-level language. It is a shallow embedding.

At the semantic point of view, the approach is : choice of languages or theories (temporal logics, higher-order logics, type theory, category theory, etc) as high-level languages, description of each low-level specification/formalism in a high-level language, reasoning on the high-level. This refers to deep embedding by considering the semantic at the low-level formalism.

The idea can be summarized as in the figure 6 (syntactical level), a general bridge (considering the input as parameter) is used in this figure.

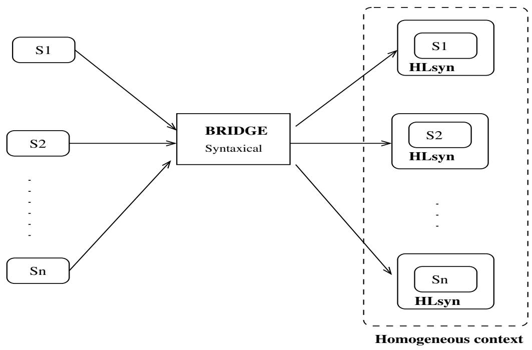

FIG. 6 – Generalization : syntactical level

The generalization of the semantic level can be summarized as in the figure 7. The point is that the integration and formal reasoning can be done in the scope of a third higher level context. Here, as semantic models can be different, we consider a different bridge for each model.

An example of integration basis is given in figure 8. It can be generalized as in figure 9.

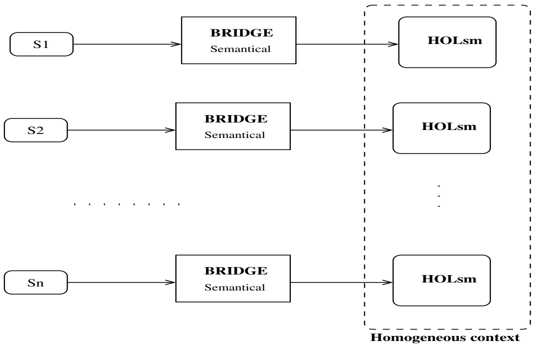

FIG. 7 – Generalization : semantic level

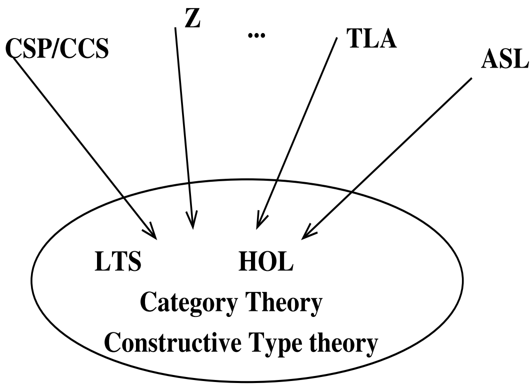

FIG. 8 – Integration basis

7 Appraisal of the work

Domain Compatibilities, syntactic and semantic compatibilities are important points for achieving methods integration. The idea presented are mainly oriented for building new integrated environment; although, they can be applied to integrating already existing techniques or methods.

The innovative and main points of our work reside in the basis of the integration approach :

- integration at methods and tools level (not only languages),
- compatible semantic models as the integration basis,
- multifaceted aspect of the integration basis,
- genericity of the approach in order to meet system requirements.

7.1 Related works

Since several years there are some proposals and results on the subject of methods integration; this appears under different terms : multiformalism approach, multiparadigm specifications, integrated methods, combining methods. We have already mention LOTOS and Raise. Some other interesting works are listed below.

A basic proposal has been done by Jackson Zave and Jackson [ZJ93] on semantic domain into which specification languages can be mapped. These authors present a multiparadigm specification as the composition of a finite set of partial specifications where a partial specification is equivalent to an assertion in first-order predicate logic with equality. They call composition of a set of partial specifications, the conjunction of their equivalent assertions.

Our work is closely related to these idea. Integration basis (semantic model) is at the place of the first order predicate logics (as semantic domain) used by Zave and Jackson. In place of the partial specifications we have paradigms or formal specification languages. This work by Zave and Jackson is followed by a specific multiparadigm

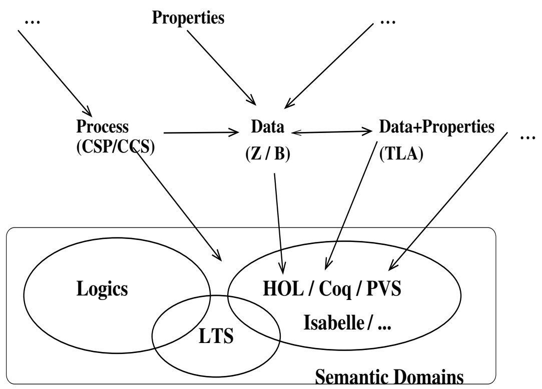

FIG. 9 – Generic Integration basis

technique [ZJ96] based on Z and automata.

Some more recents works are : the works by Paige [Pai97], [Pai99], Fischer [Fis98], Tej and Wolff [TW97], Ehrig and Orejas [EO98], Galloway [Gal96], Smith [Smi97].

Considering the work of Paige, some important points can be underlined here and compared with our work : Paige's work is concerned with formal method integration for specification, design and implementation. His approach is based on the use of an *heterogeneous basis*. This heterogeneous basis is a *set of related notations or formalisms*. This heterogeneous basis is mainly based on *data oriented languages*.

In our work, the formal integration method is more generally considered for *formal development*, the approach is based on the use of *integration basis*, defined according to *semantic models* and more generally *compatible semantic models* associated to multiple paradigms. The integration basis are multifaceted and includes data and behaviour (process).

Considering the work of Galloway, we note that it can be viewed as a particular case of the works on integration, which combine two (or more) formalisms syntactically or semantically. Indeed, this work results in a very expressive formalism combining the process algebra CCS and the Z language.

A work comparable to our approach is done in the scope of *UniForM* [KBPOB99]. The UniForM Workbench [AWK+98, KBPOB99] supports the combination of formal methods and provides tools for development and analysis of hybrid, real-time or reactive systems. We have found in this work some similarities with our work, especially at the semantic level, there is in the two works the idea of a high level logic to handle the integration and support tools. However some differences can be expressed : the UniForM Workbench enables one to integrate different formalisms or methods (Z, CSP, $CSP - OZ$, etc), equiped with tools (for example FDR for CSP, Moby/OZ for CSP-OZ) or not , in its environment and provides a new workbench for it. It is the instanciation of UniForM, an example is Z-Workbench [AWK+98]. UniForM does not provide the user with an *new* integrated method ; that means tools are integrated not the underlying paradigms.

In our work, the main concern is to provide user with multiparadigm-integrated methods and two possibilities are suggested : the first is to provide the user with "new" (multiparadigm-)integrated formalisms or methods according to the paradigms of interest and the second is like in UniForM to integrate already existing formalisms covering the different paradigms.

At the moment UniForM is more advanced than our work according to tools availability. Indeed, the UniForM Workbench [KBPOB99] provides a development environment based on the generic theorem prover Isabelle, the associated graphical interface IsaWin, the Transformation Application System (TAS) for development by transformation.

It is clear that the fundamental works on Isabelle [Pau93b],[Pau93c] and HOL gives basis for powerfull tools implementation, and on this point we can exploit the experiment of the UniForM Project.

7.2 Lessons for integrating methods

The integration of *data* and *process* paradigms, as basic paradigms, provides a means to cover a large part of real life (complex) software systems. Of course the extended approach of *process* paradigm (dealing with time) should be considered in order to tackle time constraints.

Formal models can serve as basis for a sane integration of different methods covering specification, reasoning and development.

The need of a sane integration basis in the case of methods integration viewed as *strong combination or homogeneous composition* should be contrasted with *weak combination or heterogeneous composition* where component methods are rather independent. In the former case, basic reasoning is done within a homogeneous scope, that means the common semantic domain. In the latter approach, some difficulties reside in maintaining several semantic domains, moving between the logics underlying each component method.

Considering *formal integration basis*, it is possible to envision a generic model for method integration.

Several ways exist to face the management of multiple paradigms in the level of formal reasoning : integrating reasoning tools (checkers, rewriters, provers), combining tools (checkers, provers).

Our feelings can be summarized as : *the right tools at the right place*. That means, it should be cases where combining tools will be preferable to integrating them.

However, in our work, according to the notion of compatibilities, we prefer integration and then weak combination when there is not obvious compatibilities.

8 Concluding remarks and future works

This work began as an attempt to synthetize and generalize some integration hints we have found in the literature and also noticed when dealing with a work on the definition of a framework for specifying multifaceted systems using multiple paradigms [Att98, Att99, Att00].

In this paper we have summarized the main concerns of method integration and give some outlines for realistic solutions ; this is done by considering almost all cases of methods integration and underlying the main locks especially the problem of semantic compatibility between models which relate formal specification and formal development.

We generally consider integration in the sense of composing or combining formalisms/methods with their associated semantics, in order to get an integrated method allowing homogeneous reasoning on the combined methods.

We suggest to base methods integration on multilevel architecture comprising syntactical level, semantic level and integrated reasoning level ; having in mind the sake of development, the kernel of the architecture is a *formal integration basis* acting as a common and homogeneous basis for reasoning and providing multiparadigm front-end functionalities to users.

Future works are naturally planned on putting into practice some idea presented here. Therefore, an already working illustration is the use, in an integrated formalism [Att98], of different integration basis : first order logic + set theory for data modeling and transition systems/process algebra for behaviour modeling.

The definition of a generic model as a framework for integrating methods constitutes the main topic of an ongoing work.

References

[AAH98] M. Allemand, C. Attiogbe, and H. Habrias, editors. *INVOICE'98 : Intl. Workshop on Comparing Software Specification Techniques*. Putting into Practice Software. IRIN edition, 98. ISBN 2-906082-29-5.

[Abr96] Jean Raymond Abrial. *The B Book*. Cambridge University Press, 1996.

[Arn92] André Arnold. *Systèmes de transitions finis et sémantiques des systèmes communicants*. Masson, 1992.

[Att98] Christian Attiogbé. Configuration Machines : A Simple Formalism For Specifying Multifaceted Systems. Technical Report 181, IRIN, University of Nantes, July 1998.

[Att99] Christian Attiogbé. Specifying The Dagstuhl Light Control System by Configuration Machines. Technical Report 193, University of Nantes, December 1999.

[Att00] Christian Attiogbé. Le système de contrôle d'accès : une spécification à base des machines à configurations. In *Proc. AFADL'2000*, pages 203–220, Grenoble, FR, Janvier 2000.

[AVA95] Jeffrey D. Ullman (Contributor) Alfred V. Aho. *Foundations of Computer Science (Principles of Computer Science Series)* . "W H Freeman and Co. ;", 1995. ISBN : 0716782847.

[AWK+98] Luth A, Karlsen E. W ;, Kolyang, Westmeir S., and Wolff B. HOL-Z in the UniForM Workbench - a case study in Tool integration for Z. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98 :The Z formal Notation*, volume 1493 of *Lecture Notes in Computer Science*, pages 116–134. Springer-Verlag, 1998.

[BE95] Didier Bert and Rachid Echahed. Multiparadigm logic programming : the case of the language LPG. Technical report, IMAG-LGI, 1995.

[BF00] Jean-Paul Bodeveix and Mamoun Filali. Formalisation de la méthode b en coq et pvs. In *Proc. AFADL'2000*, pages 96–110, Grenoble, FR, Janvier 2000.

[BG95] J. P. Bowen and M. J. C. Gordon. A Shallow Embedding of Z in HOL. *Information and Software Technology*, 37(5-6) :269–276, May/June 1995.

[BGG+92] R. Boulton, A. Gorgon, M.J.C. Gordon, J. Hebert, and J. van Tassel. Experience with embedding hardware description language in hol. In *International Conference on Theorem Provers in Circuit Design : Theory, Practice and Experience*, pages 129–156, Nijmegen, North-Holland, June 1992. IFIP TC10/WG 10.2.

[BGM89] M. Bidoit, M-C. Gaudel, and A. Mauboussin. *How to make algebraic specifications more understandable ?*, volume 394 of *Lecture Notes in Computer Science*, pages 31–67. Springer-Verlag, 1989.

[BH94] J. P. Bowen and J. A. Hall, editors. *Z and HOL*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[BK84] J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60, 1984.

[BKL+91] Michel Bidoit, H-J. Kreowski, P. Lescanne, F. Orejas, and D. Sanella, editors. *Algebraic System Specification and Development, a Survey and Annotated Bibliography*, volume 501. Springer-Verlag, 91.

[BKs83] R.J. Back and R. Kurki-suonio. Decentralisation of process nets with centralized control. In *2nd ACM SIGACT-SIGOPS Symposiun on Principles of Distributed Computing*, pages 131–142. ACM, 1983.

[But] Michael Butler. An overview of the csp2b tool. In *FM'99 – B Users Group Meeting – Applying B in an industrial context : Tools, Lessons and Techniques*, pages 1–2.

[But99] M. J. Butler. Csp2B : A practical Approach to Combining CSP and B. In *Proceedings of the International FM'99*, pages 490–, Toulouse, France, September 1999. Springer-Verlag LNCS 1708.

[CGR95] D. Craigen, S. Gerhat, and T. Ralston. Formal methods reality : Industrial usage. *IEEE Transactions on Software Engineering*, 21(2) :90–98, 1995.

[Cha98] Pierre Chartier. Formalisation of b in isabelle/hol. In Didier Bert, editor, *B'98 : Recent Advances in the Development and Use of the B Method*, volume 1393 of *Lecture Notes in Computer Science*, pages 66–. Springer-Verlag, 1998.

[CM88] Chandy and Misra. *Parallel Program Design : A foundation*. Addison-Wesley, 1988.

[cM96] Jean-François Monin. *Comprendre les méthodes formelles*. coll. CNET-ENST. Masson, Paris, 1996.

[CPR99] Christine Choppy, Pascal Poizat, and Jean-Claude Royer. A Global Semantics for Views. Rapport de Recherche 189, Institut de Recherche en Informatique de Nantes, December 1999.

[CW95] E. M. Clarke and J. M. Wing. Formal Methods : State of the Art and Future Directions. *ACM Computing Surveys*, 28(4) :626–643, December 1995.

[dR99] Maarten de Rijke. Combining reasoning systems. *Workshop on Automated Reasoning - Bridging the Gap between Theory and Practice*, 1999.

[Dye92] Michael Dyer. *The Cleanroom Approach to Quality Software Development*. Wiley Series in Software Engineering Practice, 1992.

[EFH83] H. Ehrig, W. Fey, and H. Hansen. ACT ONE : An Algebraic Language with two levels of semantics. *Bericht-Nr,Technische Universität Berlin*, (83-03), 1983.

[EO94] H. Ehrig and F. Orejas. Dynamic abstract data types : An informal proposal. In *Bulletin EATCS*, pages 162–169, 1994.

[EO98] H. Ehrig and F. Orejas. Integration and classification of data type and process specification techniques. Technical report, TU Berlin, 98. Workshop, Dagshtul.

[Fen96] P. C. Fencott. *Formal Methods for Concurrency*. Thompson Computing Press, 1996.

[FGJM85] Kokichi Futatsugi, Joseph A. Goguen, Jean-Pierre Jouannaud, and José Meseguer. Principles of OBJ2. In *Proc. of Twelfth ACM Symposium on Principles of Programming Languages*, pages 52–66. ACM, 85.

[Fis97] C. Fischer. CSP-OZ : A combination of Object-Z and CSP. In H. Bowmann and J. Derick, editors, *FMOODS'97 :*, volume 2, pages 423–438. Chapman and Hall, 1997.

[Fis98] C. Fischer. How to combine Z with a Process Algebra. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98 :The Z formal Notation*, volume 1493 of *Lecture Notes in Computer Science*, pages 5–23. Springer-Verlag, 1998.

[FJ92] L. M. G. Feijs and H.B.M Jonkers. *Formal Specification and Design*. Software Engineering. Cambridge University Press, 92.

[Gal96] Andy Galloway. *Integrated Formal Methods with Richer Methodological Profiles for the Development of Multi-Perspectives Systems*. PhD thesis, University of Teesside - School of Computing and Mathematics, 1996.

[GG89] Stephan Garland and John Guttag. An Overview of LP, the Larch Prover. In *Proc. of the Third International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1989.

[Gor93] M.J.C. Gordon. *Introduction to HOL : A Theorem Proving Environment* . Cambridge University Press, 1993.

[Gro] CoFI Group. CoFI : The Common Framework Initiative for Algebraic Specification and Development. *http ://www.brics.dk/Projects/CoFI.*

[Gro95a] The RAISE Language Group. *The RAISE Development Method*. BCS Practitioner Series. Prentice Hall, 95.

[Gro95b] The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall, 95.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In R. L. Probert L. Logrippo and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, Lecture Notes in Computer Science, pages 379–394. IFIP, 1990.

[GS97] A. Galloway and W.J. Stoddart. An operational semantics for ZCCS. In *ICFEM'97 :*, volume 4 of *IEEE*, pages 272–282, September 1997.

[HaCPM96] G. Huet and G. Kahn ang C. Paulin-Mohring. The Coq proof assistant, a tutorial. Technical report, INRIA Rocquencourt et CNRS-ENS Lyon, 1996.

[HJ95] Michael G. Hinchey and Stephen A. Jarvis. *Concurrent systems : Formal Development in CSP*. McGraw-Hill, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, NJ, 1985.

[Hun93] H. Hungar. Combining model checking and theorem proving to verify parallel processes. In *5th International Conference on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 154–165, 93.

[ISO89] ISO/IEC. LOTOS : A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, International Organization for Standardization, 1989.

[J.98] Atallah Mikhail J. *Algorithms and Theory of Computation Handbook*. CRC Press LLC, 1998. ISBN : 0849326494.

[Jac86] M. Jackson. *JSD*. in Computer Science. Prentice Hall, 86.

[Jon86] C. B. Jones. Systematic Program Development. *Software Specification techniques.*, pages 89–110, 1986.

[Jon90] Cliff B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 90.

[Kah87] G. Kahn. Natural semantics. In *STACS*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1987.

[KBPOB99] Bernd Kreg-Brukner, Jan Peleska, E-R Olderog, and Alexander Baer. The UniForM Workbench, a Universal Development Environment for Formal Methods. In *Proceedings of the International FM'99*, pages 1187–1205, Toulouse, France, September 1999. Springer-Verlag LNCS 1708.

[KW96] Santon Kolyang and T. Wolff. A structure preserving encoding of z in Isabelle/HOL). In *Proceedings of the International Conference on Theorem Proving in Higher Order Logic*. Springer-Verlag LNCS 1125, 1996.

[Lam94] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3) :872–923, May 1994.

[MD98] B. P. Mahony and J. S. Dong. Blending object-z and timed csp : An introduction to TCOZ. In *The 20th International Conference on Software Engineering (OCSE'98)*, pages 95–104. IEEE Computer Society Press, 98.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, NJ, 1989.

[MP84] Z. Manna and A. Pnuelli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, (4) :257–289, 1984.

[Muñ99] C. Muñoz. PBS : Support for the B-method in PVS. Technical Report SRI-CSL-99-01, SRI International, February 1999.

[Nes93] M. Nesi. Value-passing ccs in hol. In J. J. Joyce and C.-J. H. Seger, editors, *HUG'93 : Higher Order Logic Theorem Proving and its Applications : 6th International Workshop*, volume 780 of *Lecture Notes in Computer Science*, pages 352–365, Vancouver, B.C., August 1993. Springer-Verlag.

[NPW81] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1) :85–108, January 1981.

[Pai97] R. Paige. A Meta-Method for Formal Method Integration. In C. Jones, J. Fitzgerald, and P. Lucas, editors, *Proceedings of the International Formal Methods Europe Symposium (FME'97)*, pages 473–494, Graz, Austria, September 1997.

[Pai99] R. Paige. Specification and Refinement using a Heterogeneous Notation for Concurrency and Communication. In **, editor, *Proceedings of the Integrated Formal Methods (IFM'99)*, xx, UK, 1999.

[Pau88] Lawrence C. Paulson. A formulation of the simple theory of types (for Isabelle). In P. Martin-Löf and G. Mints, editors, *Proceedings of the International Conference on Computer Logic COLOG'88*, pages 246–274, Tallinn, Estonia, December 1988. Springer-Verlag LNCS 417.

[Pau93a] Lawrence C. Paulson. Co-induction and co-recursion in higher-order logic. Technical Report 304, University of Cambridge, Computer Laboratory, July 1993.

[Pau93b] Lawrence C. Paulson. Introduction to Isabelle. Technical Report 280, University of Cambridge, Computer Laboratory, 1993.

[Pau93c] Lawrence C. Paulson. Isabelle's object-logics. Technical Report 286, University of Cambridge, Computer Laboratory, 1993.

[Pau93d] Lawrence C. Paulson. Set theory for verification : I. From foundations to functions. *Journal of Automated Reasoning*, 11(3) :353–389, 1993.

[Plo81] G. Plotkin, editor. *A structural approach to Operational Semantics*. DAIMI FN-19, Computer Science Dpt, Aarhus University, Denmark, 1981.

[Rei85] W. Reisig. Petri nets : an introduction. In *EATCS Monographs on Theoretical Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 1985.

[RJD92] Milner R., Parrow J., and Walker D. A Calculus of Mobile Processes. *Journal of Information and Computation*, 100, 1992.

[Ros98] A.W. Roscoe. *The Theory and Practice of concurrency*. Series in Computer Science. Prentice-Hall, 1998.

[RWW94] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-Interference through determinism. In D. Gollman, editor, *ESORICS'94 :*, volume 875 of *Lecture Notes in Computer Science*, pages 33–54. Springer-Verlag, 1994.

[San98] Thomas Santen. On the Semantic Relation of Z and HOL. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98 :The Z formal Notation*, volume 1493 of *Lecture Notes in Computer Science*, pages 96–23. Springer-Verlag, 1998.

[Smi97] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent system. In *FME'97 :*, IEEE, 1997.

[Spi96] Michael Spiser. *Introduction to the Theory of Computation* . Pws Pub Co ;, 1996. ISBN : 053494728X.

[Spi92] Mike Spivey. *The Z Notation : a Reference Manual*. Series in Computer Science. Prentice Hall, 92.

[TA97] K. Taguchi and Keijiro Araki. The State-based CCS Semantics for Concurrent Z Specification. In *ICFEM'97 :*, IEEE, pages 283–292. IEEE Society Press, September 1997.

[TW97] H. Tej and B. Wolff. A corrected failure-divergence model for CSP in Isabelle/HOL. In C. Jones and J. Fitzgerald, editors, *Proceedings of the International Formal Methods Europe Symposium (FME'97)*, Graz, Austria, September 1997. To appear.

[Win89] J. Wing. What is a Formal Method. Technical report, CMU, November 89. CMU-CS-89-200.

[Wir86] M. Wirsing. Structured algebraic specifications : A kernel language. *TCS*, 42(2), 1986.

[ZJ93] Pamela Zave and Mickael Jackson. Conjunction as Composition. *ACM Transactions on Software Engineering and Methodology*, 2(4) :379–411, October 1993.

[ZJ96] Pamela Zave and Mickael Jackson. Where do operations come from ? A multiparadigm specification technique. *IEEE Transactions on Software Engineering* , XXII(7) :508–528, July 1996.

Institut de Recherche en Informatique de Nantes

Formal Methods Integration for Software Development: Some Locks and Outlines

Christian Attiogbé

Abstract

We present some locks related to the integration of software development methods. Here integration means putting together paradigms used by different methods in order to have an homogeneous environment suited for formal development. In a first part of the paper, after introducing formal methods in order to fix the vocabulary used in the sequel, we deal with the main difficulties for methods integration. This can be can be summarized by the following questions which are examined in the paper : What is method integration ? What methods can be integrated ? Does the integration always possible ? What are the practical locks for methods integration ? How to integrate methods ? What are the theoretical and technical solutions ?
In a second part, we present some outlines for methods integration through some elements to answer the above questions. They are : semantic domains as integration basis, domain compatibility, syntactical and semantic compatibility, refinement compatibility, specification bridges, integrated homogeneous reasoning context. Idea to build or generate integrated environments are sketched.

Formal methods[Integration]

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.4 [**Software Enfineering**]: Software/Program Verification; D.2.6 [**Programming Environments**]: Integrated environments; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Methods Integration, Specification, Development

Additional Key Words and Phrases: Formal specification, Formal Methods Integration, Compatibility, Integration basis, Specification bridge

IRIN, Université de Nantes
2, rue de la Houssinière
B.P. 92208 — F-44322 NANTES CEDEX 3