Natural Language Engineering **9** (3): 211–230. © 2003 Cambridge University Press
DOI: 10.1017/S1351324902003030 Printed in the United Kingdom

An approach to the formal specification of lingware†

BILEL GARGOURI, MOHAMED JMAIEL
and ABDELMAJID BEN HAMADOU

LARIS Laboratory, FSEG-SFAX B.P. 1088–3018 Sfax, Tunisia
e-mail: {Bilel.Gargouri,Abdelmajid.BenHamadou}@fsegs.rnu.tn,Mohamed.Jmaiel@enis.rnu.tn

(*Received 24 July 2001; revised 8 November 2001*)

Abstract

This paper has two purposes. First, it suggests a formal approach for specifying and verifying lingware. This approach is based on a unified notation of the main existing formalisms for describing linguistic knowledge (i.e. Formal Grammars, Unification Grammars, HPSG, etc.) on the one hand, and the integration of data and processing on the other. Accordingly, a lingware specification includes all related aspects in a unified framework. This facilitates the development of a lingware system, since one has to follow a single development process instead of two separate ones. Secondly, it presents an environment for the formal specification of lingware, based on the suggested approach, which is neither restricted to a particular kind of application nor to a particular class of linguistic formalisms. This environment provides interfaces enabling the specification of both linguistic knowledge and functional aspects of a lingware system. Linguistic knowledge is specified with the usual grammatical formalisms, whereas functional aspects are specified with a suitable formal notation. Both descriptions will be integrated into the same framework to obtain a complete requirement specification that can be refined towards an executable program.

1 Introduction

Most industrial applications (e-commerce, marketing, safety systems, text processing software, etc.) need to integrate a natural language component (lingware component) allowing, for instance, man-machine communication, machine translation, text summarization, document filtering, spelling and grammatical checking, and information retrieval. This requirement, which is in perpetual evolution, supposes that the development process of lingware components is performed in a way that guarantees the reliability and reduces the cost. However, in practice, we notice an almost total absence of development methodologies that integrate the whole lingware system life cycle. In particular, the requirement specification is usually informal, ambiguous and/or incomplete. The use of formal concepts is restricted to the specification of linguistic knowledge, whereas only algorithmic notations have been applied to

† Lingware: software related to Natural Language Processing (NLP).

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

describe functional aspects. Moreover, the development process is not supported by specific CASE tools that help, first, to choose the most appropriate linguistic formalism for an NLP application due to the large variety of linguistic formalisms, and secondly, to integrate different linguistic formalisms in the same lingware system. These observations can be consolidated by investigating the features of the existing environments and platforms devoted to lingware engineering such as ALEP (Meulemans 1994) and GATE (Cunningham, Gaizauskas and Wilks 1996).

To provide solutions to the problems mentioned above, particularly those related to lingware system specification and the integration of linguistic formalisms, we propose resorting to Formal Methods (FM) with associated tools. These methods have made their appearance in the specification and development of several complex systems in which reliability is of the essence, such as air-traffic control systems (Heimdahl and Leveson 1996), operating systems and communication protocols (Jmaiel 1996). Many FM have been adopted as standards like VDM (Jones 1986), Z (Spivey 1998), CSP (Hoare 1985) and LOTOS (Int 1987). Most FMs are supported by specific CASE tools ensuring the development of reliable software products.

In this paper, we demonstrate, through the results of the investigations that we conducted using VDM, that it is possible to give solutions for the problems indicated above. The main idea consists of transforming linguistic formalisms (HPSG, LFG, UG, ATN, FG, etc.) into a pivot language (VDM Specification Language: VDM-SL) to bring them back to a single notation. The transformation of these formalisms is based on a set of rules preserving their original semantics. In addition, the processing functions, also specified using VDM-SL, are encapsulated with the associated linguistic knowledge in a unified specification. The obtained (complete) requirement specification will be validated step-by-step and refined towards a design specification that is easy to implement. The validation and refinement steps are supported by a set of tools associated with the VDM method. We consolidated our approach by developing an environment enabling formal specification and development of lingware systems (Gargouri, Jmaiel and Ben Hamadou 2001). Furthermore, our environment provides evaluation tools permitting the engineer to choose, at the earliest stage, the appropriate linguistic formalisms for a piece of lingware (Gargouri, Jmaiel and Ben Hamadou 2000). This paper is organized as follows. First, we emphasize the advantages of using FM in lingware engineering. Then we propose a unified representation of linguistic knowledge using a pivot notation. After that, we detail the formal steps which lead to the design specification. Finally, we present the environment associated with the proposed approach.

2 Principal motivations

FMs are a set of tools and notations (with a formal semantics) used to specify, unambiguously, the requirements of a computer system that supports the proof of properties of that specification and proofs of correctness of a possible implementation with respect to that specification (Hinchey and Bowen 1995). They can greatly increase the understanding of a system by detecting inconsistencies, ambiguities and

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

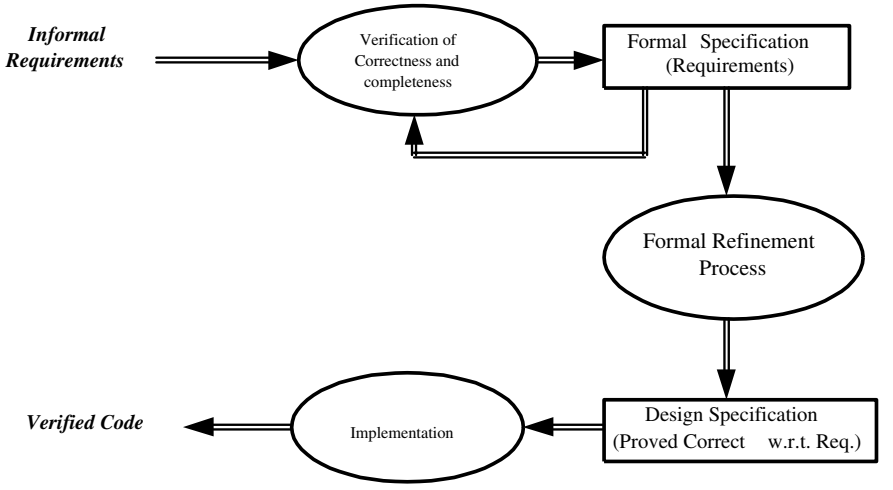


Fig. 1. Development approach using FM.

incompleteness allowing the construct of reliable software. The basic concepts of FM are:

- *Formal specification*: an abstract description of a system and its desired properties;
- *Formal refinement*: the process that transforms an abstract requirement specification to a detailed specification that represents the last version of the system;
- *Validation*: based on two approaches (1) the verification of the model (or model checking), and (2) the application of validation theorems (theorem proving). They are applied to analyze the properties of the system to be developed.

Figure 1 shows the general process of using FM in developing software.
The main advantages of FM can be summarized in the following points:

- First, the use of formal notation at the first stages of development (i.e. requirement specification) allows one to detect the limits of the system to be developed. This enables one to start the design stage on a consistent version of the system, which considerably reduces the backtracking process.
- Secondly, at the design level, the use of FM enables one to integrate data and processing functions in a single specification. The consistency of a specification is then systematically verified using formal proofs.
- Finally, formal methods permit one to formulate, by applying successive refinements, a set of reusable abstractions that should be available in a library (Darricau, Hadj Mabrouk and Ganascia 1997).

In the lingware development context, we may add that the use of a unique notation (i.e. formal notation) makes it possible to integrate into the same application a variety of linguistic knowledge specified with different formalisms. This will make for greater

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

coherence in the final specification. Furthermore, FM helps the developers choose the appropriate linguistic formalisms for a lingware system, by applying a set of formal evaluation criteria (Gargouri, Jmaiel and Ben Hamadou 2000). Finally, FM allows for the verification of the multilingual properties of any given piece of lingware.

Both general and specific advantages have been demonstrated by carrying out a complete and provably reliable specification of the CORTEXA system (Correction ORthographique des TEXtes Arabes) (Ben Hamadou 1993) using the VDM formal method. Besides the availability of the documentation concerning the conceptual data and the code, the choice of CORTEXA is also motivated by the diversity of approaches used to represent knowledge and processing.

The use of VDM for the specification of CORTEXA is motivated not only by the fact that VDM has made its appearance in the development of several applications, particularly industrial ones (Barroca and McDermid 1992; Hall 1996), but also for its high expressive power for describing applications. Indeed, we were able to develop an abstract specification of the CORTEXA system (also called implicit specification) that includes formal specifications of its error detection functions and its correction rules. We were also able to formally prove the correctness and the completeness of this specification. Finally, we generated a design specification (also called direct or explicit specification) from the abstract one by applying VDM refinement rules. This design specification has been easily transformed into an executable code (for more details, see Mnif (1997)).

In the following sections we propose a methodological approach and a set of tools that promote the application of FM in the context of lingware development.

3 Unified representation of linguistic knowledge

3.1 Unification approach

When we develop lingware system we are often led to use several linguistic formalisms describing different language levels (i.e. morphology, syntax, semantics). Then, the major problem that we have to deal with is first to find out a notation expressive enough for representing these formalisms, and secondly, to define a set of transformation rules which guarantee the equivalence between the original and the resulting representation.

In the following, we show how it is possible to use the VDM-SL (the VDM Specification Language (Dawes 1991)) as a unique formal notation to which we add a set of rules for transforming the elementary components (i.e. sets, production rules, maps, functions, features, etc.) shared by the linguistic formalisms (see section 3.3). This set of rules can be implemented to perform an automatic conversion. We thus facilitate the user's task, since he does not have to deal with a new notation for the representation of the linguistic knowledge. A similar approach was proposed in Newton (1993) using algebraic specifications as a single notation for describing grammars. So we propose that he acquires linguistic knowledge with the original formalisms, as shown in Figure 2.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

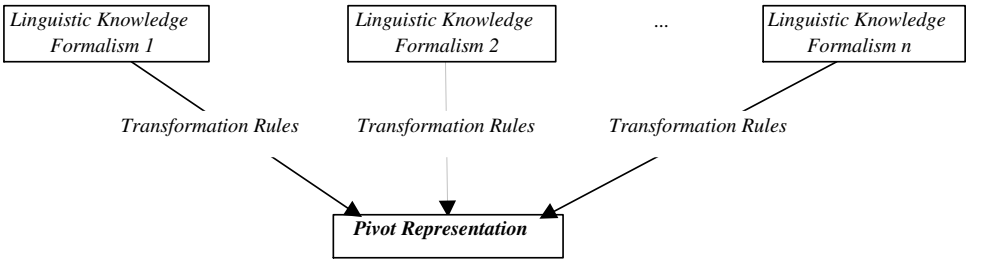


Fig. 2. Integration of the main existing formalisms for linguistic knowledge description.

The transformation rules are used according to the following steps, when they are applied to a given representation of linguistic data:

- first, the different components of the given global structure will be recursively decomposed until canonical data types are reached;
- then, canonical types are transformed into the corresponding VDM notations;
- finally, the global VDM representation will be constructed by composing these notations (reversing their decomposition path), using VDM composition instructions.

These steps are supported by the transformation function presented in section 3.3.

It is worth noting that the applied rules will preserve the names of the defined types as well. Thus, the names used in the VDM specification of linguistic data are those given initially by the user. This enables one to use them in the specification of the functional aspect without modification.

3.2 Representation of grammars in VDM

To illustrate how linguistic data can be represented in VDM-SL (VDM Specification Language), we give a formal specification of two kinds of grammars: Formal Grammars and Unification Grammars.

These kinds of grammars are widely known in the NLP field. Indeed, Formal Grammars have been the first to appear in this area, and they were at the origin of the majority of linguistics formalisms. Similarly, Unification Grammars have made their appearance in the description of linguistic knowledge. Several formalisms have been derived from these grammars, such as GPSG (Gazdar, Klein, Pullum and Sag1985) and HPSG (Pollard and Sag 1994).

Formal grammar representation. In this section, we give a VDM representation of a generic context-free grammar. An instance of such a grammar is given in section 6. Let us remind the reader that a context-free grammar is represented by a quadruple that contains a set of terminal symbols, a set of non-terminal symbols, a start symbol belonging to the set of non-terminal, and a set of production rules.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

A production rule is a pair composed of a non-terminal symbol and a sequence over the union of the sets of terminal and non-terminal symbols.

Let us introduce the following notation:

- Alphabet of non terminal symbols: $Alp_{Nterm} \stackrel{\text{def}}{=} \{A, B, \ldots, Z\}$.
- Alphabet of terminal symbols: $Alp_{Term} \stackrel{\text{def}}{=} \{a, b, \ldots, z\}$.

Then we define the data structure associated with a production rule as follows:

$$\begin{aligned} Prod_Rule \ :: \ & A : Alp_{Nterm} \\ & \alpha \ : \textbf{\textit{seq of}} \ (Alp_{Nterm} \cup Alp_{Term}) \end{aligned}$$

According to this definition, a context free-grammar in VDM-SL is a new type (referred to as $Grammar_{cf}$):

$$\begin{aligned} Grammar_{cf} \ :: \ & T : \textbf{\textit{set of}} \ Alp_{Term} \\ & N : \textbf{\textit{set of}} \ Alp_{NTerm} \\ & S : Alp_{NTerm} \\ & P : \textbf{\textit{set of}} \ Prod_Rule \end{aligned}$$

$$\begin{aligned} \textbf{\textit{where inv-}}Grammar_{cf}() \stackrel{\triangle}{=} \ & ((T \neq \emptyset) \wedge (N \neq \emptyset) \wedge (P \neq \emptyset) \wedge (S \in N) \wedge \\ & \forall r = \textbf{\textit{mk-}}Prod_Rule(A, \alpha) \ \in \ P : \\ & A \in N \wedge \alpha \in \textbf{\textit{seq of}} \ (N \cup T) \end{aligned}$$

Representation of Unification Grammars. Unification Grammars build a particular class of formalisms used in NLP. They have been applied to describe several language aspects, such as syntax, semantics, etc., using unified schemata. An unrestricted Unification Grammar has the expressive power of a Turing machine.

The representation of Unification Grammars in VDM proposed here is based on a formal description presented in Haas (1989). A *unification grammar* is defined as a quintuple $G = (S, (\Sigma, r), T, P, Z)$ where S is a set of sorts, (Σ, r) an S-ranked alphabet, T a finite set of terminal symbols, and Z a function letter of arity e in (Σ, r). Z is called the start symbol. P is a finite set of rules; each rule has the form $(A \longrightarrow \alpha)$, where A is a term of the ranked alphabet and α is a sequence of terms of the ranked alphabet and symbols from T. To define unification grammars in VDM, we first define the following sets:

- The set of possible sorts: $Sorts \stackrel{\text{def}}{=} \{s_1, s_2, \ldots, s_i, \ldots\}$.
- The set of possible function symbols: $FuncSym \stackrel{\text{def}}{=} \{f_1, f_2, \ldots, f_i, \ldots\}$ we suppose
  that $Sort \cap FuncSym = \emptyset$.
- We also define the type of signatures with the associated ranks:

$$\begin{aligned} Sign \ :: \ & S : Sorts \\ & Sig : FuncSymb \\ & r : \textbf{\textit{map}} \ FuncSymb \ \textbf{\textit{to}} \ (\textbf{\textit{seq of}} \ Sorts) \times \ Sorts \\ \textbf{\textit{where inv-}}Sign() \stackrel{\triangle}{=} \ & (\textbf{\textit{Dom}}(r) = Sig) \ \wedge \ \textbf{\textit{Rng}}(r) \subseteq (\textbf{\textit{seq of}} \ S) \times S \end{aligned}$$

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

- The set of production rules associated with the signatureΣ:

$$\begin{aligned} Rule \;::\; & A : Terms \\ & \alpha \;:\; \textbf{\textit{seq of}}\ (Alp_{Term} \cup Terms) \end{aligned}$$

- A unification grammar in VDM-SL corresponds to the type *UniGrammar*:

$$\begin{aligned} UniGrammar \;\textbf{::}\; & S : Sorts \\ & \Sigma : Sign \\ & T : Alp_{Term} \\ & Z : FuncSym \\ & P : Rule \end{aligned}$$

$$\begin{aligned} & \textbf{\textit{where inv-}}UniGrammar() \triangleq \\ & (T \neq \emptyset) \wedge (P \neq \emptyset) \;\wedge\; (\Sigma = \textbf{\textit{mk-}}Sign(Sig, r) \Rightarrow (Z \in Sig \wedge r(Z) \in S)) \wedge \\ & \forall p \in P.\ p = \textbf{\textit{mk-}}Rule(A,\ \alpha) \Rightarrow A \in T_\Sigma \wedge \alpha \in \textbf{\textit{seq of}}\ (T \cup T_\Sigma) \end{aligned}$$

In this definition we denote by T_Σ the set of terms over the signature Σ. We have omitted its VDM representation to keep the definition simple and readable.

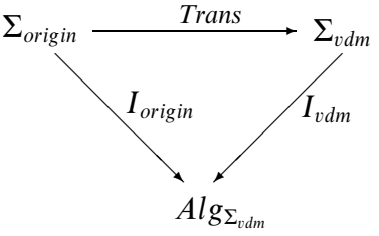
3.3 How to ensure equivalence

To prove formally that transformations preserve the meaning of the original grammars, we must first present them formally. In fact, the translation of grammars into VDM notation can be performed by a function from the domain of original grammatical notations to the syntactic domain of VDM. Then, the input domain collects all types needed to represent and analyse linguistic data. It includes elementary data types, such as natural numbers, characters and Booleans, as well as composed types, such as sets, relations, sequences, predicates, trees, schemata and recursive structures. In this context, it should be noted that the notation of VDM is expressive enough to cover all types used to represent linguistic knowledge. The transformation function will be defined inductively on the structures of data that belong to its domain. Then, to each elementary type corresponds an elementary VDM type. For example, natural numbers, characters and Booleans are represented in VDM as $\mathbb{N}$, *IC* and $\mathbb{B}$, respectively. Similarly, with each composed type we associate a corresponding composed type in VDM. For example, with the set of sequences over an alphabet *Alp* we associate the type ***seq of*** *Trans*(*Alp*), where *Trans*(*Alp*) is the set of characters that represent *Alp* in VDM. Moreover, since we also deal with terms such predicates, constraints and terms over sequences (as in Unification Grammars) we have to take into consideration operators associated with types. So the domains are defined as signatures, consisting of sorts and operators. Thus, our transformation function is defined from the signature of original grammatical types to the signature of VDM types.

Then, we define the semantics associated with each signature. Each signature is interpreted (using the function *I*) by an algebra that assigns to each sort a set of

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

data and to each operator an operation (Ehrig and Mahr 1985). These signatures turn out to have the same semantic domain, i.e. they are interpreted by the same algebra. Thus, we obtain the following relations between the different domains:



To guarantee the equivalence between the original notation and the VDM notation, we have to prove that the homomorphism "*Trans*" preserves the meaning of a transformed element. Hence, we have to prove the following theorem:

$$\forall F_G \in T_{\Sigma_{origin}} : I_{vdm}(Trans(F_G)) = I_{origin}(F_G)$$

The proof of this theorem can be trivially achieved by induction on the structures of sorts included in Σ_{origin} (A detailed proof is given in Nebli (2000)).

4 Processing specification and integration

In this section, we focus on the formal specification and development of Lingware system using a unified representation of linguistic knowledge. We demonstrate how we can obtain a complete specification of an NLP application using a formal language like VDM-SL (Dawes 1991). We start from the assumption that an NLP application is composed of a set of routines applied to a set linguistic knowledge (i.e. grammars). Considered are all kind of routines, including classical, probabilistic and statistical ones. In general, the functional and static aspects follow different processes of development and are dealt with, in most cases, at a low level of abstraction using an algorithmic or a programming language. In our approach, we attempt to manipulate them at a high level of abstraction using a unified framework that allows us to specify both aspects. This enables us to develop an application by applying the same development process for the static as well as for the functional aspect. Here, we do not need to take care of the different development steps and tools, instead we continue the development process in a purely formal way by applying the refinement rules associated with the adopted FM.

4.1 Processing specification

The first task to be performed at this stage is to develop a complete (w.r.t. the informal requirements) specification that formally describes the different routines that

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

should be implemented. The specification of the processing part of an application is made by a set of functions and operations, in the sense of VDM. These functions are given at a high level of abstraction. They formally describe "**What** to do?" and not "**How** to do?". A specification that includes this kind of function is called, in VDM, implicit or indirect specification. An implicit specification is composed of three parts:

- *The Heading*: this declares the function name and the names of its arguments with the corresponding types. It also defines the name and the type of the expected result. The name of the result will be used in the post-condition part.
- *The Pre-condition*: this sets conditions for the function arguments. The pre-condition indicates whether the function is partial or total. If it corresponds to *True*, this means that the function is total.
- *The Post-condition*: this describes, in an abstract way, with a set of formulas, how the result is related to the arguments.

An implementation of such a specification is considered to be correct, if the following property is guaranteed: *whenever the pre-condition is satisfied, the post-condition has to be satisfied too.*

Note that the pre- and post-conditions are based on first order logic, which is considered expressive enough to cover all the properties one may associate with an application.

We are not concerned, at this stage, with the definition of the required data structures, since they are described in a different module. All we need is to refer to these data structures by their names.

Example 4.1

To illustrate the use of formal methods to specify the functional aspect of lingware system, in this example we deal with the generation of context free languages. The following VDM specification describes a function that generates a language from a context-free grammar. Here, we consider only one derivation step, i.e. from a given string we collect all strings that can be derived from it by applying a production rule. The function heading declares the parameters as well as the result. The pre-condition part requires that the given string should be a sequence over the union of the sets of terminals and the non terminals. The post-condition states that if a production rule can be applied to the given string, then the derived string should be included in the result set. Additionally, it requires that the obtained set contains only strings that have been derived from the application of a production rule starting from the given string.

$$
\begin{aligned}
&\mathit{Generate} : (w : \mathit{String};\ G : \mathit{Grammar}_{cf})\ S_w : \textbf{\textit{set of}}\ \mathit{String}\\
&\textbf{Pre:}\quad G = \textbf{\textit{mk-}}\mathit{Grammar}_{cf}(V_N, V_T, S, P) \wedge w \in \textbf{\textit{seq of}}\ (V_N \cup V_T)\\
&\textbf{Post:}\quad \forall w' \in \textbf{\textit{seq of}}\ (V_N \cup V_T) :\\
&\qquad (w' \in S_w \Leftrightarrow \exists (B, \beta) \in P, \exists w_1, w_2 \in \textbf{\textit{seq of}}\ (V_N \cup V_T) :\\
&\qquad\qquad w = w_1 B w_2 \wedge w' = w_1 \beta w_2)
\end{aligned}
$$

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

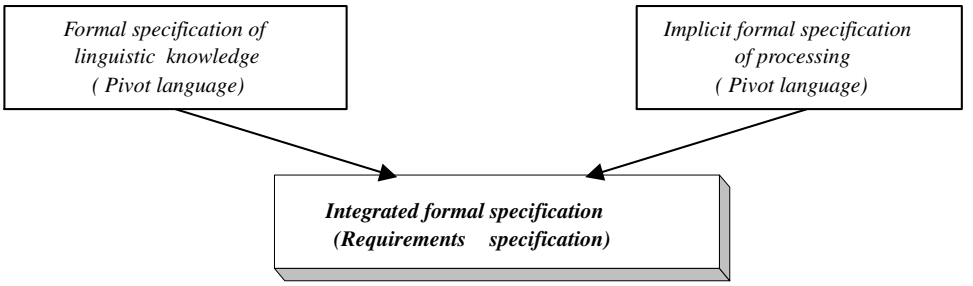


Fig. 3. Integration step of data and processing within a unified specification.

This example shows how precisely one can represent the function that treats linguistic knowledge.

4.2 Integration and refinement

Considering the development process, we initially start from a specification that describes the user requirements in an informal way using natural language in general. Based on this specification, we evolve an implicit specification using the VDM notation. In this way, we attempt to translate informal requirements to (formal) VDM implicit functions. Because there are no formal techniques for proving that informal and formal requirement specifications match, we follow general principles of requirement engineering (e.g. see Partsch (1991)).

In developing formal requirement specifications from informal ones, we take two aspects of specifications into account: *completeness* and *consistency*. Completeness means that the formal specification reflects all informal requirements. Obviously, it is difficult to verify such a property formally. All we can do is check that informal requirements are represented in the VDM specification by the corresponding functions. Consistency means that there is no contradiction within the formal specification. In contrast to completeness, consistency can be formally verified by showing that at least one model satisfies the specification. The linguistic knowledge required is separately described and transformed into a VDM specification (see section 3) before specifying the lingware functional aspects. The latter refer to the linguistic knowledge using their type identifiers. However, for reasons of simplicity, one can use the identifier types related to the initial linguistic formalisms and subsequently use a pseudo-pivot notation. Thereafter, the obtained specification is purified: transformed into a pure VDM specification.

In this way, we can integrate within a unique framework, as indicated in Figure 3, both types of formal specification (data and processing).

Consequently, we can apply the development process associated with the retained FM (VDM in our case) which is based on successive formal refinements of the requirement (abstract) specification. The refinement process provides a design specification which is guaranteed to be correct w.r.t. the requirement specification.

Figure 4 presents the use of the refinement and proof correctness process applied to the unified specification in the pivot language.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

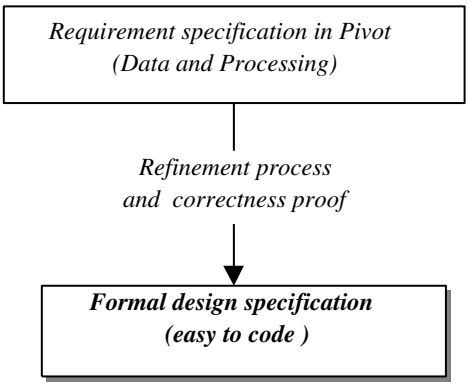


Fig. 4. Refinement and correctness proof.

So far, we have presented the overall formal aspects of our proposal for a lingware system specification. This approach offers solutions to most extant lingware engineering problems. However, we can enhance the benefits derived from this approach when we develop an associated environment and implement tools supporting it.

5 An environment for lingware specification

5.1 General presentation

This section presents the environment associated with the proposed formal approach for lingware system specifications. The main purpose of this environment is to assist lingware engineers, especially at the first steps of development (Gargouri, Jmaiel and Ben Hamadou 2001). This environment proposes to construct the design specification of a piece of lingware starting from its requirement specification. Accordingly, our environment provides a user-friendly interface enabling the description of both aspects (linguistic knowledge and related treatments) separately. The functional aspect is described directly in the pivot notation (VDM-SL), whereas the linguistic knowledge description is made by current grammar formalisms. In this way we release the user from mastering a new formal language for describing linguistic knowledge. Moreover, in order to favour reusability of verified specifications, our environment manages a dynamic library. This library includes, among others, linguistic knowledge (i.e. grammars in VDM), verified specifications of standard linguistic routines (i.e. generation, analysis, etc.), and verified requirement specification of Lingware system components.

Additionally, this environment can act as a platform for the evaluation of linguistic formalisms or grammars. This permits the engineer to make, at the earliest stage, the most appropriate choice of formalism for the development of a particular piece of lingware (Gargouri, Jmaiel and Ban Hamadou 2000).

5.2 Global architecture of the environment

Our environment is composed of the following units:

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

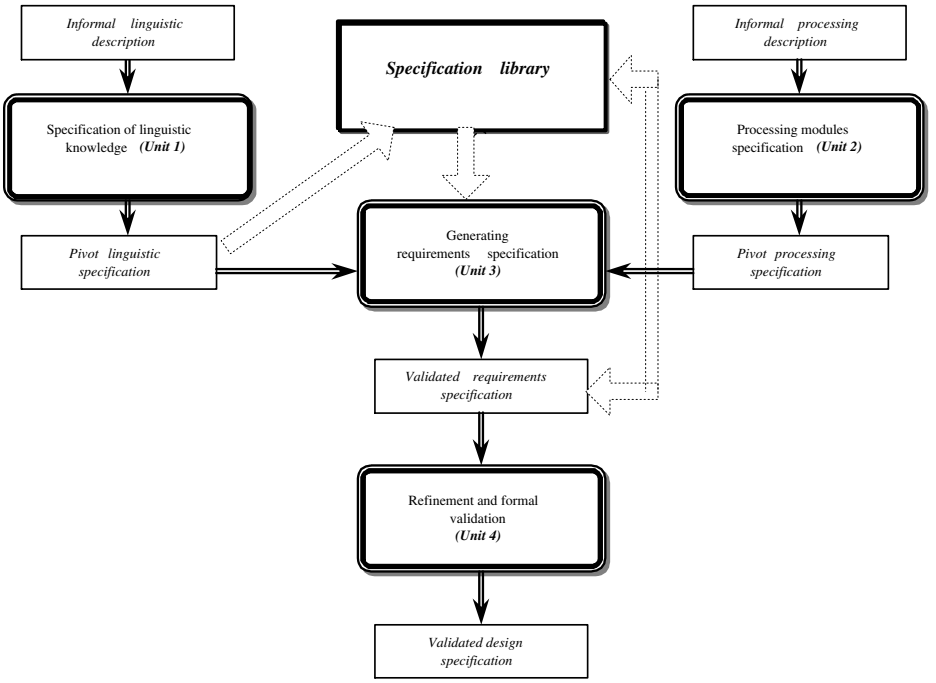


Fig. 5. Environment global architecture.

- Linguistic knowledge specification.
- Processing modules specification.
- Generation of the requirement specification.
- Refinement and formal validation.

Figure 5 gives the global architecture of the proposed environment for lingware system specification. It presents the interaction of its different units.

In this figure, unit3 (generating the requirement specification) links all the environment units. Indeed, it uses (integrates) the two kinds of specification obtained from unit1 (specification of linguistic knowledge) and unit2 (processing modules specification), respectively. Moreover, the output of unit3, constitutes the input of unit4 (Refinement and formal validation). These units will be described in detail in the following sections.

5.3 Specification of linguistic knowledge unit

This unit provides the necessary tools for generating a verified requirement VDM specification of the linguistic knowledge needed for the desired application. It includes the following components:

- A set of graphical interfaces.
- A set of syntactic and lexical checkers.
- Transformation tools.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

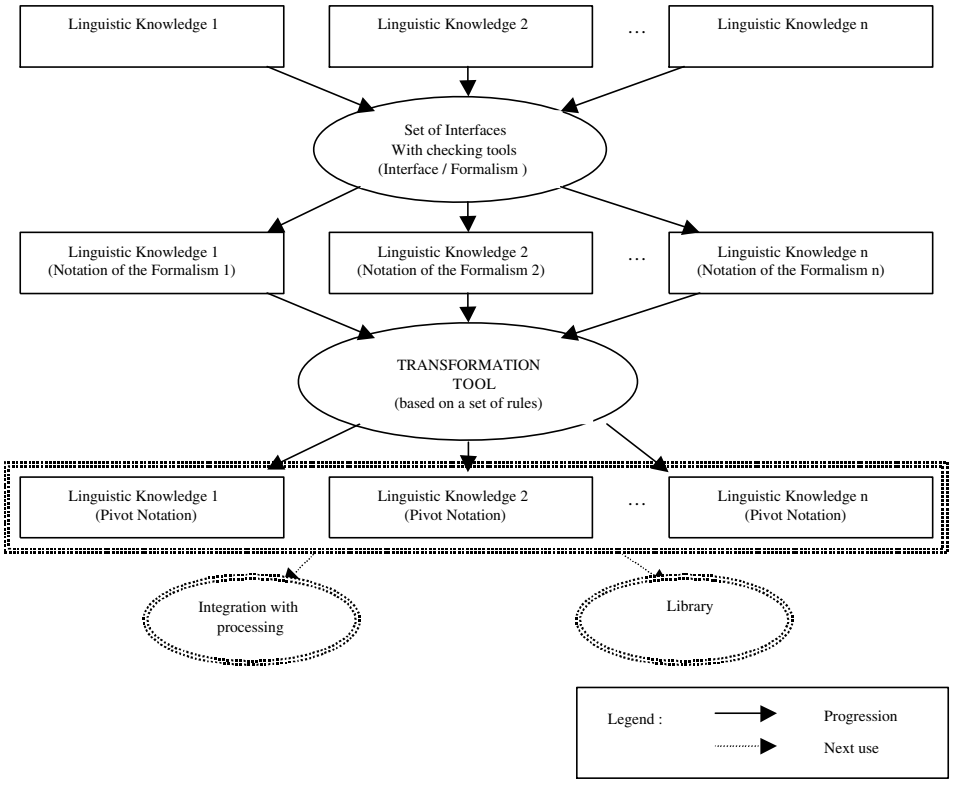


Fig. 6. Acquisition and transformation of linguistic knowledge.

Figure 6 explains how the present unit operates. The linguistic knowledge is meant to be introduced using the appropriate formalisms. For each formalism we provide a suitable interface. The descriptions given will be lexically and semantically analysed until validation. Thereafter, they will be transformed into the pivot language using a specific tool based on a set of rules. This transformation guarantees the equivalence between the original descriptions and the final ones. The resulting specification will be integrated into the processing specification of the piece of lingware to be developed. It may also be added to the library for future reuse. It should be noted that the present unit has already been implemented for several formalisms (i.e. Formal Grammars, Unification Grammars, Augmented Transition Networks). Also, it is open to extensions since it is always possible to consider new formalisms by adding appropriate tools handling them.

5.4 Processing specification unit

This unit concerns the specification of the functional part of a lingware system. Such a specification operates on the data types describing the linguistic knowledge. At this level, functional specifications are made in a pseudo-pivot notation. This notation is a slightly modified version of the pivot notation. Indeed, it is based on VDM-SL, but

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

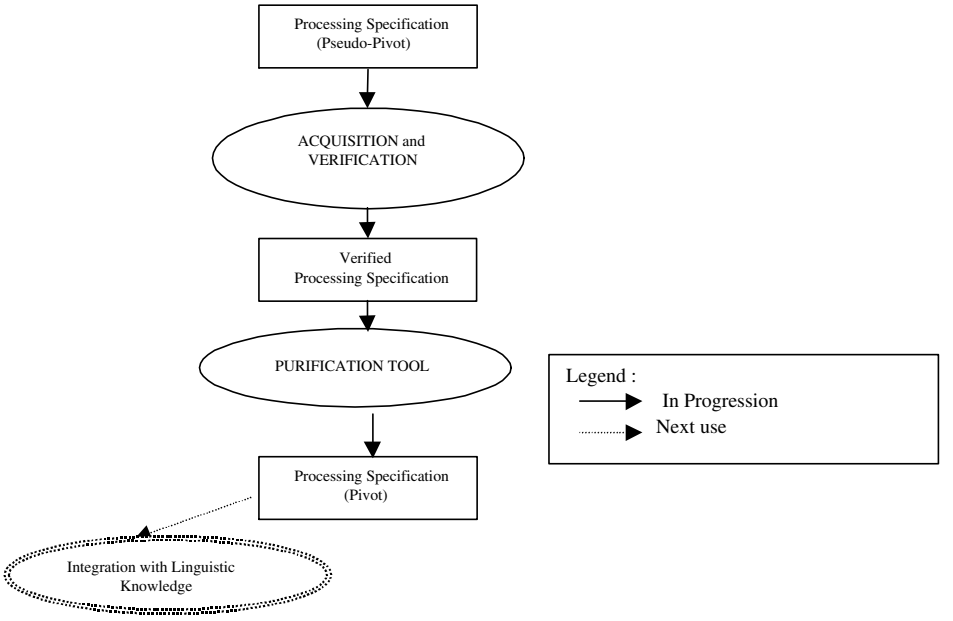


Fig. 7. Processing specification of a Lingware system.

it uses the original grammatical notations when handling linguistic data. In order to obtain the pure pivot specification we need to establish links between the VDM-SL and the grammatical notations used. Such a pseudo-pivot language facilitates the user's task, since he does not need to master the notation of the pivot language to describe linguistic knowledge.

This unit is composed of the following tools as represented in Figure 7:

- A tool that allows the acquisition of the formal processing specification in the pseudo-pivot notation.
- A syntactic analyser for specifications of functional aspects.
- A transformation tool, that translates a pseudo-pivot notation into a pivot notation.

The informal description of processing is introduced in a pseudo-pivot notation using a simple interface. The specifications of the processing modules will be verified with respect to the pseudo-pivot syntactical rules, especially as regards the identification of linguistic knowledge. This verification allows the purification of these specifications into pure pivot. This task consists of transforming all "non-pivot" identifiers into their equivalents in pivot notation. The verification is ensured since the transformation tool generates only syntactically correct specifications. The specification obtained will subsequently be used to construct the requirement specification of the lingware system under development. The semantic verification is made by tools associated with the pivot language.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

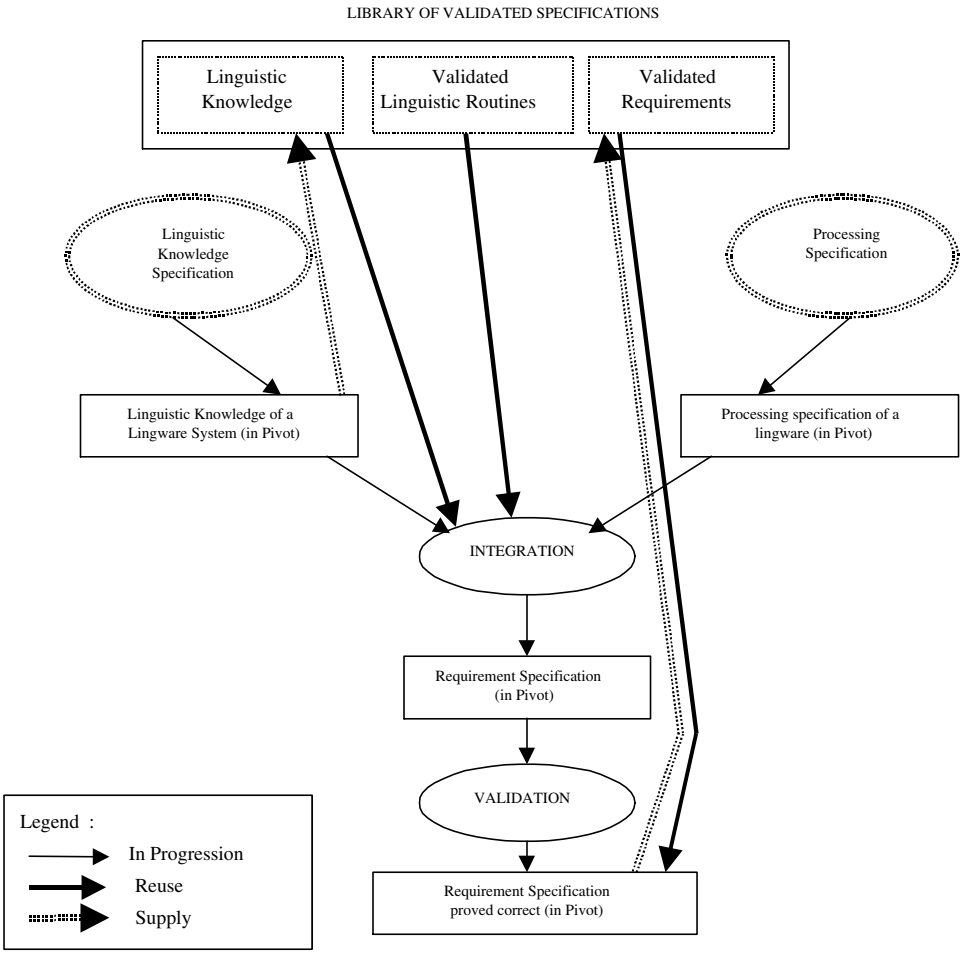


Fig. 8. Generation of a verified requirement specification.

5.5 Generating requirement specifications unit

The role of this unit is to construct a verified complete requirement specification of a Lingware system by integrating, in a single description, the processing and the related linguistic knowledge. This unit is composed of the following tools (see Figure 8):

- A library of verified specifications (linguistic knowledge and requirements).
- An integration tool.
- A validation tool (for requirement specification).

Figure 8 illustrates process of constructing a provably correct specification of a piece of lingware:

The resulting requirement specification will be verified (syntactically and semantically) using specific tools associated with the FM used. An already validated requirement specification, stored in the library, can be imported (reused) and added

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

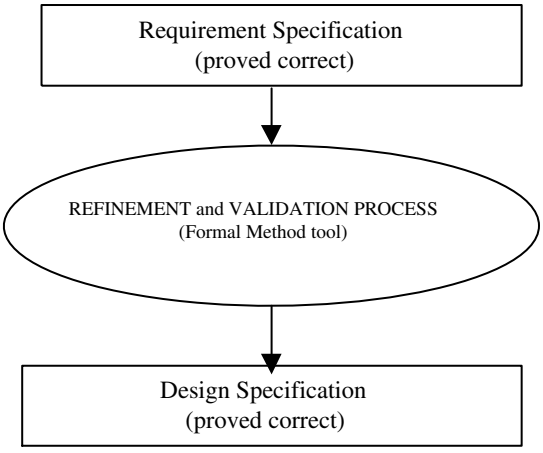


Fig. 9. Refinement and validation unit.

to the current specification for the purpose of completion. It should be noted that a newly constructed (verified) requirement specification can be inserted into the library for future reuse.

5.6 Refinement and formal validation unit

The aim of this unit is to generate a design specification which satisfies the requirement specification. Currently, we make use of tools, associated with the chosen FM (i.e. SpecBox, VDMTools), which allow one to ensure the refinement and validation of specifications.

Figure 9 presents the unit for refining and validating specifications. This figure shows the application of a process of refinement and successive validations on a verified requirement specification obtained from the integration of knowledge and functional specifications. This process ends with a design specification of the lingware system that satisfies the requirement specification. This process follows the methodology of the FM applied.

6 Implementation

Our environment has already been implemented in a first version using Visual C++. It runs on a standard PC having at least the Win95 operating system. Currently, it includes the majority of tools described in section 5. All interfaces are designed as standard windows including the usual services.

Figure 10 presents the main interface of the environment and illustrates its use. The interface includes a menu-bar which includes usual functionalities such as creating a new file, opening an existing one or saving a file, etc.

Moreover, this menu-bar offers the following services which are specific to the environment:

- "Tools": includes the following functionalities:

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

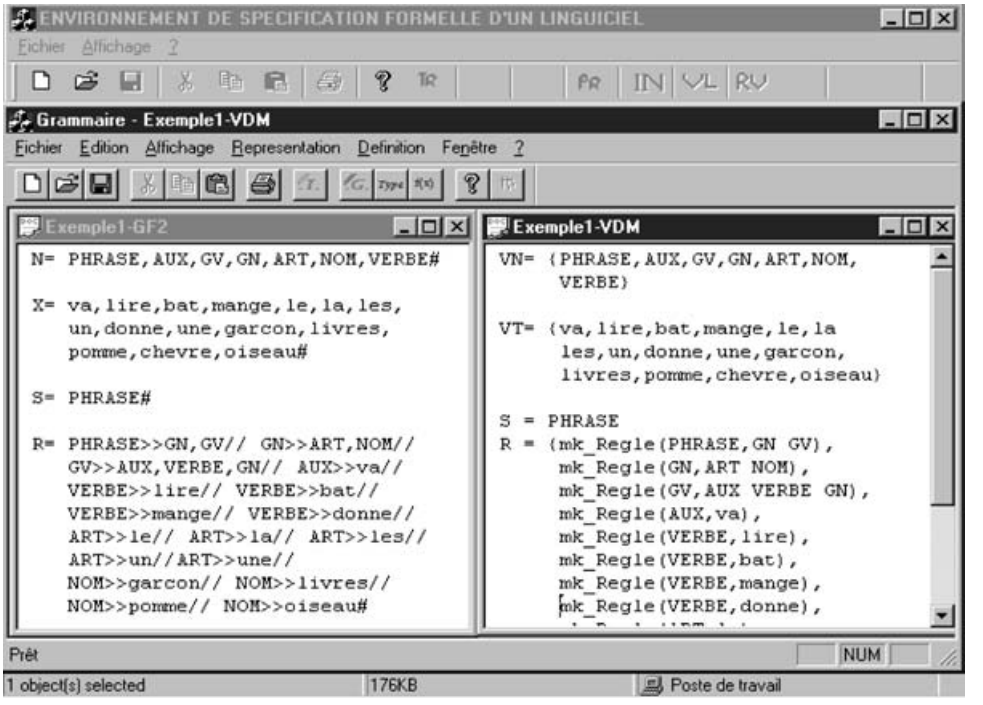


Fig. 10. The environment interface.

  - "Linguistic knowledge" : transforms the linguistic knowledge description into a VDM specification.
  - "Processing Module": transforms a processing specification into a pure VDM one.
  - "Integration": unifies the linguistic knowledge VDM specification and the related processing one. The result of this unification is a VDM requirement specification.
  - "Validation": validates the requirement specification obtained.
  - "Refinement": generates a design specification from the requirement specification.

- "Library": offers tools to browse the library for linguistic knowledge specifications and requirement specifications.
- "Application": provides tools for opening or creating a complete piece of lingware including all related specifications.

All these services are also represented using shortcut icons. The availability of services depends on the specification progress.

Additionally, Figure 10 presents the transformation tool. It illustrates the use of this tool by transforming a Context-Free Grammar. This tool includes specific services that are indicated in the menu-bar as well as in the shortcut icons-bar. These services are:

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

- "Representation": transforms the description of the linguistic knowledge into a VDM notation.
- "Definition": provides predefined specifications of types and functions (in VDM) associated with the initial linguistic formalism.

7 Discussion

Several environments and platforms related to the lingware engineering area already exist. These environments can be classified according to their objectives. We indicate different kinds of environment. First, we distinguish those environments that focus only on the manipulation of linguistic formalisms. Among these, we can mention D-PATR (Karttunen 1994) associated with unification grammars, HDRUG (Van Noord and Bouma 1997) and HPSG-PL (Erbach, Manandhar and Uszkoreit 1996) for the HPSG grammars. Secondly, we distinguish the environments that deal with the development of specific lingware system such as TIPSTER (Grishman 1996) which deals with specific applications that manipulate documents (i.e. indexing, extraction, etc.). Finally, the third kind of environment, which is relatively recent, concerns the normalised development of Lingware system. Some of them have been recently proposed in first versions, such as ALEP (Meulemans 1994), PLNLP (Jensen, Heidorn and Richardson 1993) and GATE (Cunningham, Gaizauskas and Wilks 1996). Others again are projects in their initial stages, such as MULTEXT/EAGLES (Veronis 1996).

In general, these environments manipulate a particular family of formalisms for linguistic knowledge description (i.e. formalisms based on unification) limiting the possibility of considering other types of knowledge in the same application. Usually, one is restricted to the same formalism for an application. Furthermore, some of these environments allow the integration of data and processing, using, in general, the object-oriented approach, but they do not perform any kind of formal verification. The focus is rather on reuse. These environments propose, generally, their proper languages for the description of linguistic knowledge and processing. Some of these environments use specific concepts (i.e. task, action, object and resolution for ALEP), which requires a special training for their users.

Compared with the environments mentioned above, the originality of our environment lies in the following points:

- The use of a formal approach that covers the whole life cycle of a lingware system.
- The use of a pivot representation for the linguistic knowledge while keeping, for the sake of simplicity, the initial notations as interfaces with users.
- The integration of data-processing in a unified framework.
- The verification of completeness and consistency (i.e. correcteness) of specifications.
- The reuse of provably (w.r.t. initial informal specification) correct and complete specifications.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

However, the current version of our environment still suffer from certain limitations, which we try to overcome in the coming versions. These limitations can be summarized in the following points:

- Our environment considers only the requirement specification stage of a lingware system. This stage includes the specification of both linguistic data and processing, as well as their integration and validation. The remaining stages of the development process could be supported by existing tools appropriate to the formal method chosen. These tools allow one to generate design specification and an executable code from the obtained requirement specification.
- It is always possible to extend our environment with new formalisms. However, this task requires the intervention of the administrator who should define the appropriate rules of transformation into the VDM notation. The user is not supposed to master the method of transforming linguistic formalisms. This limitation can be overcome by integrating an intelligent unit accessing an increasing database that manages a predefined set of transformation rules.
- Finally, our environment tolerates redundancies of the linguistic knowledge descriptions, particularly in the specification library, due to the fact that the latter is enriched with disjunct descriptions made initially by different formalisms. However, the pivot notation promotes future works aiming at the construction of a single and complete description of natural languages.

8 Conclusion

In this paper, we have presented an environment for formal lingware engineering that is based on a specific approach using FM. This approach presents solutions to the majority of lingware engineering problems. It also offers a methodology that guides the user in all steps of the development process.

Moreover, the environment that we have presented offers the following advantages. First, it covers all the approach phases and makes them automatic. Second, it facilitates several tasks of the user (i.e. many tasks are transparent to the user), which makes it more adapted to his needs. Finally, it meets the general principles of software development tools, since it is user-friendly, enables the reuse of previously verified specifications, and may be combined with other tools.

Currently, we are carrying out an experiment specifying two real applications (i.e. a morphological analyser (Ben Hamadou 1993) and a syntactic analyser (Larson, Warren, De Lima and Sagonas 1995)) using our formal approach and profiting from the available tools. It is hoped that the results of these experiments will lead to improved results.

References

Barroca, L. and McDermid, J. (1992) Formal methods: use and relevance for the development of safety-critical systems. *Computer J.* **35**(6).

Ben Hamadou, A. (1993) Vérification et correction automatiques par analyse affixale des textes écrits en langage naturel: le cas de l'arabe non voyellé. PhD thesis, Thése d'Etat en Informatique, Faculté des Sciences de Tunis.

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press

Cunningham, H., Gaizauskas, R. G. and Wilks, Y. (1996) A general architecture for text engineering (GATE) – a new approach to language engineering. Technical Report CS 95-21, Department of Computer Sciences, University of Shiffield. (Available at http://www.lanl.gov/ps/cmp-lg/9601009.)

Darricau, M., Hadj Mabrouk, H. and Ganascia, J. (1997) Une approche pour la réutilisation des spécifications de logiciels. *Génie logiciel* **25**(45), 21–27.

Dawes, J. (1991) *The VDM-SL Reference Guide*. Pitman.

Ehrig, H. and Mahr, B. (1985) *Fundamentals of Algebraic Specification 1*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag.

Erbach, J., Manandhar, S. and Uszkoreit, H. (1996) A report on the Draft Eagles Encoding Standard for HPSG. *Proceedings TALN'96'*, Marseille, France.

Gargouri, B., Jmaiel, M. and Ben Hamadou, A. (2000) Using a formal approach to evaluate grammars. *Proceedings Second International Conference on Language Resources and Evaluation (LREC'2000)*, Athens, Greece.

Gargouri, B., Jmaiel, M. and Ben Hamadou, A. (2001) An environment for formal specification and verification of lingware. *Proceedings 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Lecture Notes in Artificial Intelligence 2070*, pp. 539–545. Springer-Verlag.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. (1985) *Generalized Phrase Structure Grammar*. Blackwell.

Grishman, R. (1996) TIPSTER text phase ii architecture, version 2.2.

Hall, A. (1996) Using formal methods to develop an ATC information system. *IEEE Software* **12**(6), 66–76.

Heimdahl, M. and Leveson, N. (1996) Completeness and consistency in hierarchical state-based requirements. *IEEE Trans. Softw. Eng.* **6**(22), 363–377.

Hinchey, M. and Bowen, J. (1995) *Applications of Formal Methods*. Prentice Hall.

Hoare, C. (1985) *Communicating Sequential Processes*. Prentice Hall.

Int (1987) *Information Systems Processing – Open Systems Interconnection – LOTOS*.

Jensen, K., Heidorn, G. and Richardson, S. (1993) *Natural Language Processing: The PLNLP Aproach*. Kluwer Academic.

Jmaiel, M. (1996) Specification of communication protocols using temporal logic. *J. Syst. & Softw.*, Special Issue on Software Engineering for Distributed Systems **33**(3), 299–312.

Jones, C. B. (1986) *Systematic Software Development using VDM*. Prentice Hall.

Karttunen, L. (1994) D-PART: A development environment for unification based grammars. *Proceedings 11th International Conference on Computational Linguistics*, pp. 74–80. Bonn, Germany.

Larson, R., Warren, D. S., De Lima, J. F. and Sagonas, K. (1995) *Le systéme SYNTACTICA*. MIT Press.

Meulemans, P. (1994) ALEP arriving at the next platform. *ELSNews* **3**(2), 4–5.

Pollard, C. and Sag, I. (1994) *Head-Driven Phrase Structure Grammar*. University of Chicago Press/CSLI Publications.

Spivey, J. M. (1998) *Introducing Z: A specification language and its formal semantics*. Cambridge University Press.

Van Noord, G. and Bouma, G. (1997) Hdrug: a flexible and extendible development environment for natural language processing. *Proceedings Conference ACL-EACL'97*, Madrid, Spain.

Veronis, J. (1996) *MULTEXT Workpackage 2 Milestone B Deliverable Overview, version 1.0*, LRE 62050 MULTEXT. (Available at http://www.lpl.univ-aix.fr/projects/multext/.)

https://doi.org/10.1017/S1351324902003030 Published online by Cambridge University Press