Information and Software Technology 44 (2002) 65–75

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Bases for the development of LAST: a formal method for business software requirements specification[☆]

Jesús Manuel Almendros-Jiménez[a,*], Luis González-Jiménez[b]

[a]Department of Languages and Computation, Universidad de Almería, E-04120 Almería, Spain
[b]Department of Business and Economics, Universidad de La Rioja, E-26004 Logroño, Spain

Received 17 January 2001; revised 29 October 2001; accepted 30 October 2001

Abstract

This paper proposes a possible approach to IS requirements specification. It relies on the application of standard (i.e. conventional) discrete mathematics, more precisely, it uses a fairly limited number of concepts from the fields of linear algebra and set theory (hence its name, LAST). The use of LAST for data definition and query–answer are discussed in some detail, given the data-rich quality of Business IS and the fact that a solid data-model is therefore essential to their specification. The proposed approach implies integration with other semiformal specification methods, two of the possibilities being integration with UML–OCL and with the Entity Relationship Model, which are discussed in this paper. Finally, mapping of LAST specifications to the Relational Model is also addressed; this possibility having an interest both, for (partial) implementation and for model simulation. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Software engineering; Information systems; Formal methods; UML; Entity relationship model; Relational model

1. Introduction

As indicated in the title, this article conveys information about a proposed approach for the specification of requirements. This approach relies on the use of discrete mathematics and chooses to limit its scope to the Information Systems (IS) domain. As it seems only fair to give some justification for the relevance of such undertaking, this introduction will try and address briefly the following issues:

1. What is it that makes IS specially relevant and, at the same time, deserving of specific treatment regarding requirements analysis and specification?
2. Is it sound, meaning effective and efficient, to apply Formal Methods (FM) in practice?
3. At what stage or stages of the project life-cycle are FM applicable?
4. How and why are FM used in combination with existing semiformal specification methods?
5. Why not use one of the already established FM for the indicated endeavor?

There are several factors that make business IS requirements specification a very important subject, both from a practitioner's and a researcher's point of view.

- On the demand side, changes in the economic environment, more concretely, the so-called (economic) globalization, which basically implies the removal of barriers to trade and investment [31], has led to increased competition in the marketplace. Simultaneously, Information and Communication Technologies, as the Internet or the Enterprise Resource Planning (ERP) systems, have become part of the *basic technology* that, being in principle accessible by any business, no company can afford to give up altogether, simply because that implies giving a potential competitive advantage to the other companies in the same industry.
- On the supply side, the Software Industry has been active in providing increasingly sophisticated solutions to tend to the resulting needs and even to anticipate new ones. In this regard, trends in IS development point to web-enabled ERP + CRM integrated systems, providing web-based integrated Back and Front Office. ERP systems [24] are designed to support and automate the business processes of medium and large companies, including manufacturing, distribution, personnel, project management, payroll, and financials. Customer Relationship

[☆] This a revised and improved version of the paper appeared in Ref. [2].
[*] Corresponding author. Tel.: +34-9500-15677; fax: +34-9412-99393.
E-mail addresses: jalmen@ual.es (J.M. Almendros-Jiménez), luis.gonzalez@dee.unirioja.es (L. González-Jiménez).

Management (CRM) systems are [24] enterprise-wide applications that allow companies to manage every aspect of their relationship with customers. These solutions have very much improved the level of IS-clients satisfaction. This can arguably be attributed to the fact that they allow for intense customization.
- There is another emerging factor that will clearly increase the need for better solutions to IS requirements specification methods and procedures: ERP and CRM outsourcing to ASP (application service providers), which involves mass customization.

Yet, however complex, commercial applications follow a very specific pattern [4]. They are based around one or more databases, shared by many users, that reflect the state of some business domain; function is provided to users to support business operations; as the later progress, the databases are updated to reflect the past, current and planned future states. On the other hand, commercial application development projects frequently take longer and cost more than their sponsors would wish [4]. An early and influential field study [9] underscored three major problems affecting software productivity and quality:

- the thin spread of application domain knowledge,
- fluctuating and conflicting requirements, and
- communication bottlenecks and breakdowns.

In the case of IS, there are organizational, human and social issues associated with the development of business SW [37,38].One problem area is the 'uncomfortable join' between the works of business analysts and programmers [4]. A similar communication problem seems to exist between clients and developers [39]. To discuss whether it is reasonable to expect the use of FM to contribute to the improvement of the IS development process, specifically where requirements specification is concerned, it is relevant to briefly review some recent trends on FM selective and integrated application.

The need to uncover and correct software defects during the requirements analysis and design phases of software development has lead to an interest in Formal Specification Techniques (FSTs) [28]. FSTs are mathematically based techniques that provide formal notations for precisely modeling system properties and a mechanism for analyzing the resulting specifications. They are used at the requirements and design phases primarily to uncover ambiguities, missing details and inconsistencies in software models. Their ability to do this stems from their use of precise notations that can be rigorously analyzed.

Despite the advantages that derive from their application, the use of FM for SW development is far from widespread for a number of reasons [6,7,43].

FM complexity may be one of the obstacles to a more frequent use. On the other hand, FM devised for the specification and verification of systems, which are both safety-critical and highly complex, may prove unnecessarily sophisticated for other types of systems. Thus, recent years have witnessed an increasing interest on the possibilities of FM, both outside the domain of critical systems, and as specification, rather than verification, tools [7]. In addition, more domain-focused use of FM may reduce the level of complexity that FM application involves. VDM has been applied outside the critical sector; for instance, by Baan Front Office Systems for the development of a general tool—SalesPlus—for the configuration of services and products, and by GAO (Germany) for a Banknote processing system.

Another problem arises from the understandable reluctance on the part of SW engineers and the companies they work for to give up long established semiformal techniques at which they excel. It is quite possible that an evolutionary change [6], where semi formal methods and techniques are combined with formal ones, or where FM are used selectively, at different stages or for specific purposes, is likelier to be adopted. A compromise of sorts that some FM advocates have taken to in recent years in different ways.

FM may be applied at different stages of the development process. Wordsworth's [43] discussion is illuminating on this regard, on how FM may play a role in different stages of the project life-cycle. However, selective use of FM tends to focus on requirements specification [3,30]. This is what the so-called lightweight formal modeling [1,11,22,26,27,35] does. Under this approach, FM are basically used, in the early stages of the software development cycle, as a flaw-detector and, capitalizing on the intrinsic FM abstraction, as a means of reducing complexity and improving the understanding of requirements by the development team. Verification properties are sacrificed, but it has been contended [21,37] that formal specifications may in fact be valuable independently of their use for program verification. Lightweight modeling has been applied to the modeling of EDI applications [19] in Object-Z [10,38] and in FOOM methodology [14], which combines formal modeling with less-than-formal specifications in MOSES graphic language [40]. Other approaches have used the formal language VDM++ combined with UML-style graphical interfaces [32].

The integrative approach [8,42] to the use of FM is equally consistent with the purpose of making a selective use of them. Clarke and Wing [8] argue that FM can complement less-than-formal specifications to cover the whole system development process. They could be used not instead of, but in addition to, informal methods. It is specially recommended to use FM for requirements specification [41,42].

Integration of FM with other methods has several advantages [5,16,18], one of them being that it improves the chances of FM being adopted by industry. The lack of commercial success for formal methods, especially in non-critical systems development, is partly explained by common misconceptions [5,21], but a major factor is the lack of engineering context for formal approaches [25,34]. A key realization is that a practical development modeling and analysis approach requires (1) a judicious mixture of formal and informal techniques [15], and (2) a set of integrated tools supporting the construction, analysis, and transformation of software models, and the linking of software models across development phases (traceability).

Integrated methods were motivated by a wish to counter fear of formal methods in the industry community. There are two distinct approaches:

- The predominant approach takes a structured specification and applies a systematic, perhaps automated, translation, based on a formal definition of the syntax and semantics of the notations. This approach has been applied to a variety of structured methods, using various formal notations [17]. Recently, research has focused on the formalization of UML syntax and semantics [12].
- The alternative is to treat the formal and various structured models as overlapping views, none of which is a complete representation of the system. The approach is favored by the SAZ project [36], and also by works on OMT/UML and B [13]. It is most relevant where the system requirements are incomplete, or the specification evolves during the development of the formal description. The approach exploits the formalization process, i.e. the precision of thinking required to construct formal models, to clarify system details.

The second approach does not presume a correct or consistent set of structured models. Rather, the error and inconsistencies of the structured models motivate the review. The result is greater understanding of the system specified, rather than a precise representation of the structured models.

However, selective and/or combined use of FM is not the sole condition for their industrial application. Most of the projects discussed in Ref. [23], for instance, place great emphasis on tool support. This is by no means coincidental, but rather follows a trend, which is expected to result in integrated workbenches to support formal specification, just as CASE workbenches support system development using more traditional structured methods. A range of basic tools are now widely available, many of them in the public domain. For example, for support using the *Z* notation, ZTC, fuZZ, and CADiZ. The Mural system provides support for the construction of VDM specifications and refinements and IFAD's VDM-SL Toolbox is a set of tools which supports formal development in draft standard VDM-SL. The B-toolkit from B-Core (UK) Ltd, is a set of integrated tools which augments Abrial's B-Method for formal software development by addressing industrial needs in the development process.

This paper contends that an approach is possible to the use of FM for IS requirements specification that is cost-efficient and, and the same time, may contribute to the improvement of the quality of IS requirements. Along the lines mentioned in Ref. [6], the technical, social and economic contexts of the IS application domain must be brought to bear on that effort.

It is equally desirable to sacrifice whenever is possible, logical completeness to applicability, tractability and comprehensibility [6]. Thus, the proposed method should not be more complex than strictly required and, as far as possible, should make the most of the common ground that business analysts, software engineers, and management experts may share. When it comes to providing an interface among professionals with different fields of expertise, it would seem an step in the right direction to use a well established formal notation, which is more or less readily available in the background of these professionals: (elementary) discrete mathematics. The use of standard (i.e. conventional) mathematical notation seems therefore advisable and, in the view of this paper, sufficient for the stated purposes. Though there are alternatives to sharing a common language, if that may be made possible without impairment of other aspects of the development process, it cannot but be accepted as a desirable property of a method. Indeed, the need of an interface between the different project stakeholders is self-evident [29]. That does not necessarily imply that all of them need communicate using formal specifications, but these may provide precision when and where that is desirable.

This paper being a report on the advance of a particular line of research [2,20], it cannot cover all the aspects of the requirements analysis that could be addressed using the proposed method. Given the central role that information modeling plays in IS development [33], it will focus on data modeling and query–answer definition. Section 2 addresses the use of LAST for these tasks. Section 3 deals with the mapping of the formal specifications thus obtained to UML–OCL. Finally, in Section 4, correspondence with E/R Diagrams and mapping to the Relational Model are briefly discussed.

2. IS requirements specification with LAST: a sample

LAST is an attempt to build a formal method, specific to IS requirements definition, that is not more complex than strictly necessary. Consistently with this purpose, a very limited number of fairly accessible mathematical tools will be necessary: regular mathematical notation, boolean operators, basic set theory, and a few elements of linear algebra: vector, vector-entry and coordinate function, i.e. a function that acts on a vector $V = (v_1, v_2, \ldots, v_n)$, returning the designated vector-entry: $\vec{X}_i(V) = v_i$.

2.1. Data definition with LAST

LAST's two basic conceptual constructs for data modeling are the *set of transactions* and the *set of categories*. For each (business) transaction type and for each category type, the specification will define a (universal) set representing all the valid instances of the type that may be stored in the DB of the specified IS at any given moment. A set definition will establish:

- Which type (of either transaction or category) does the set represent, and notation for the set:

$$U_X = \{\text{name of the type}\} = \{X\}$$

where X (in upper case) designates the associated vector type. The later may adopt one of two forms:

- fixed length vectors: $X = (e_1, \ldots, e_m)$, where m is a constant; or
- variable length vectors:

$$X = (h_1, \ldots, h_m, l_{1,1}, \ldots, l_{1,n}, \ldots, l_{p,1}, \ldots, l_{p,n})$$

  where m and n are constants, and p is a variable. In addition, it is adopted the convention: $p = h_m$; that is, the last entry in the section of the vector that cannot be repeated, henceforth referred to as the *heading*, indicates the number of *lines* contained in an instance of the vector; a *line* being a group of entries which is or may be repeated in different instances of a variable-length vector type. As a result of said convention, it holds for this second vector form that: $|X| = m + n \cdot h_m$.

Transaction types, where binary relationships with 1:*n* mappings are pervasive, will be associated with variable length vectors; whereas in the case of categories, associated vector types will have a fixed length.

- The attributes of the type (of transaction or category), and, for each of them: which entry in the associated vector represents its value, and the set of valid values of such entry. When defining the later, both *entity integrity* and *referential constraints*, plus the *minimum* and, where applicable, the maximum *number of lines* (only for transaction types), have to be taken into account. In all cases, the set of valid values of an entry will be a subset, proper or not, of a *data type*. This constraint will be implicit for entries representing *foreign keys*. By convention, data types will necessarily be linearly ordered sets $((\forall(a,b) \in \underline{X} : a \preceq b \vee b \preceq a)$, where $\underline{X}$ is the data type. For their representation, conventional mathematical notation will be used if available (in the case of sets of numbers: $\underline{N}$, $\underline{Z}^+$, $\underline{Q}^+$, etc.); otherwise, an underlined word or phrase, clearly identifying the data type will be used, e.g. *strings*.

For instance, the following would be the specification of the type ‘sales order’.

`Transaction type`: sales order

$$U_T = \{\text{sales\_orders}\}$$

$$= \{T$$

$$= (D, N, Z, S, C, R, I_1, Q_1, P_1, \ldots, I_r, Q_r, P_r, \ldots, I_R, Q_R, P_R)\}$$

where D is the date of the transaction; N, the (reference) number of the transaction; Z, the sales region; S, the sales person; C, the customer; R, the number of lines (these six vector-entries being the *heading* in this case); and for every $r \in [1,R]$, the triple: $I_r =$ item, $Q_r =$ quantity and $P_r =$ price; which constitute a *line*.

Note that the length of a member of this set is: $|T| = 6 + 3 \cdot \vec{R}(T)$. Obviously, the number and meaning of the entries making up the heading or belonging to each line will be different for each defined type.

Let us assume that, in the case of the running example, the following holds

- Sales orders are to be identified by their date and reference number.
- The number of items in a sales order may range from 1 to 100.
- Quantities and prices have to be positive numbers with two decimal places.
- Attributes Z, S, C and I are enumerated types and, at the same time, are relevant for classification purposes, so that a category type will be defined for each of them. In addition, suppose that every category type that will be defined in the context of the running example will have a *single key* or identifier. In such case, the following convention may be adopted: the first entry of a vector type associated to a category type is the *identifier* of the category and will be represented by the letter K. Therefore, if X is a category type, the following will necessarily hold:

$$\forall X, X' \in U_X : \vec{K}(X) = \vec{K}(X') \Leftrightarrow X = X'$$

- Finally, as most if not all IS include time limits for transaction-data writing and, unless the database is perpetual, for retrieval as well, the system database will contain transaction data only for the period $[\rho, \tau]$ (i.e. $\forall T \in U_T$, $\vec{D}(T) \in [\rho, \tau]$) while $[\sigma, \tau]$, $\sigma \subseteq [\rho, \tau]$, will be the time interval for which transaction data are allowed to be entered by the end-user.

Considering all of the above, the set of valid values for each of the entries in the vector type T will be:

1. $\vec{D}(T) \in \{x \in \underline{Dates} \mid \rho \preceq x \preceq \tau\}$;
2. $\vec{N}(T) \in \underline{Z}^+ - \{\vec{N}(T') \mid T' \in U_T, T' \neq T \text{ and } \vec{D}(T') = \vec{D}(T)\}$;
3. $\vec{R}(T) \in \{x \in \underline{Z}^+ \mid x \leq 100\}$;
4. $\forall r \in [1, \vec{R}(T)] : \vec{Q}_r(T), \vec{P}_r(T) \in \{x \in \underline{Q}^+ \mid x \times 10^2 \in \underline{Z}^+\}$;

5. $\vec{Z}(T) \in \{\vec{K}(Z) \mid Z \in U_Z\}$, $\vec{S}(T) \in \{\vec{K}(S) \mid S \in U_S\}$, $\vec{C}(T) \in \{\vec{K}(C) \mid C \in U_C\}$, and $\forall r \in [1,R] : \vec{I}_r(T) \in \{\vec{K}(I) \mid I \in U_I\}$.

Note that *entity integrity* is guaranteed by item 2 above, because the definition of the set of valid values implies that: $\forall T, T' \in U_T : (\vec{D}(T), \vec{N}(T)) = (\vec{D}(T'), \vec{N}(T')) \Leftrightarrow T = T'$. As regards *referential constraints*, they are dealt with in item 5.

By way of illustration of the procedure for category types, let us now define the category *items*. In this case, the following assumptions (which will be client's requirements in practice) are made:

- Each item will be identified by a reference code, the form of which has not been as yet specified.
- The database will contain a short description of each item.
- Quantities, prices and average costs have to be positive numbers with two decimal places.

Category type: item

$$U_I = \{\text{items}\} = \{I = (K, D, Q, P, C)\}$$

where K is the item reference code; D, the description of the item; Q, the quantity on hand; P, the current selling price; and C is the average cost (per unit) of the stock on hand.

The sets of valid values will be:

1. $\vec{K}(I) \in \underline{\text{Item\_Codes}} - \{\vec{K}(I') \mid I' \in U_I, I' \neq I\}$;
2. $\vec{D}(I) \in \underline{\text{Dates}}$; and
3. $\vec{Q}(I), \vec{P}(I), \vec{C}(I) \in \{x \in \underline{Q}^+ \mid x \times 10^2 \in \underline{Z}^+\}$.

To close this subsection, there are two issues that are worth considering:

- Transactions will necessarily have, among their attributes, their date. In addition, it is customary in business to number transactions sequentially. When there are (rather uncommon) exceptions to this, the time of the transaction accompanies the date. This fact considered, it does not seem detrimental to the generality of the method to establish the following general restriction for modeling purposes:
  - If the transaction type includes date and number among its attributes, then, the first two entries of the associated vector type will stand for the values of these two attributes, and they will constitute the *identifier* of the transaction.
  - Otherwise, the first two entries will represent the date and the time of the transaction, these pair of vector entries forming the *identifier* of the transaction.
- As regards categories, it will be more often than not that there will be a identifying code to start with. If that is not the case, instances of a category may always be sequentially numbered. Therefore, it seems equally unrestrictive to the generality of the method to establish as a general constraint the assumption made in above example. That is, the first entry of a vector type associated with a category type will always represent its *identifier*.

2.2. Database-query (reporting) specification with LAST

2.2.1. Non-aggregated reports

These reports are the output of what is commonly referred to as on-line inquiry. They will consist in (linearly) ordered lists of transactions or categories that verify certain conditions. The specification must establish:

- Notation and description of the parameters of the query that the end-user will be asked to enter.
- The constraints on those parameters; i.e. definition of their valid values.
- Set-builder expression of the set of transactions or categories that verify the search and retrieval criteria for the values of the query parameters.
- Set-builder expression of the ordered set.

An instance of specification of this type of report is provided below.

Report description: list of the sales orders received in a particular sales region, between two specified dates, ordered by date, the number being the secondary criterion.

1. *Query parameters*: k is the sales region identifier; a, the starting date; and b is the closing date.
2. *Constraints on the parameters*: $[a,b] \subseteq [\rho, \tau]$, and $\exists Z \in U_Z \mid \vec{K}(Z) = k$.
3. *Search and retrieval:* $L_T = \{T \mid T \in U_T, \vec{D}(T) \in [a,b], \vec{Z}(T) = k\}$.
4. *Sorting:* Ord $L_T = \{T_i : i = 1, 2, \ldots, \text{Card } L_T\}$, such that: $T_i < T_j \Leftrightarrow \vec{D}(T_i) < \vec{D}(T_j) \vee (\vec{D}(T_i) = \vec{D}(T_j) \wedge \vec{N}(T_i) < \vec{N}(T_j))$.

2.2.2. Aggregated reports

These are relatively complex reports. Their specification will consist in:

- Notation and description of the query parameters.
- Constraints on above said parameters.
- Set-builder expression of the set of *strings of vector entries* that verify the search and retrieval criteria for the values of the query parameters.
- Set-builder expression of the classified set.
- Set-builder expression of the set of report components, which will be a function of the values of one or more vector entries of the elements in the classified set.
- Set-builder expression of the ordered set.

For instance, a report disclosing the sales of each item in each sales-region, for a given period $[a, b]$, would be specified as follows.

Report description: breakdown, by item and sales region, of the sales orders received in a given period.

1. *Query parameters*: a is the starting date; and b is the closing date.
2. *Constraints on the parameters*: $[a, b] \subseteq [\rho, \tau]$.
3. *Search and retrieval*: $L_S = \{S = (Z, I, Q, P) = (\vec{Z}(T), \vec{I}_r(T), \vec{Q}_r(T), \vec{P}_r(T)) \mid T \in U_T, \vec{D}(T) \in [a, b], r \in [1, \vec{R}(T)]\}$
4. *Classification*. Prior to aggregation, elements in L_S have to be classified, according with the two established criteria: sales-region (Z) and stock-item (I_r). The result may be described as a set, X, of Card $U_I \times$ Card U_Z subsets of L_S:

   $X = \{X_{i,j} : i = 1, \ldots, \text{Card } U_I; j = 1, \ldots, \text{Card } U_Z\}$

   such that $X_{i,j} = \{S \in L_S | \vec{Z}(S) = \vec{K}(Z_j) \text{ and } \vec{I}(S) = \vec{K}(I_i)\}$, where $Z_j \in U_Z$ and $I_i \in U_I$
5. *Aggregation*. In this case, it is necessary to compute a report-component, $Y_{i,j}$, for every element of X:

   $$Y_{i,j} = \begin{cases} 0, & X_{i,j} = \phi \\ \sum_{S \in X_{i,j}} \vec{Q}(S)\vec{P}(S), & X_{i,j} \neq \phi \end{cases}$$

   $i = 1, \ldots, \text{Card } U_I, \; j = 1, \ldots, \text{Card } U_z$
6. *Sorting*. Assuming the following disclosure (Table 6)

   The set $Y = \{Y_{i,j} : i = 1, \ldots, \text{Card } U_I; j = 1, \ldots, \text{Card } U_Z\}$ would be have to be ordered as follows:

   $\text{Ord } Y = \{Y_k : k = 1, \ldots, \text{Card } U_I | Y_i < Y_j \Leftrightarrow \vec{K}(I_i) < \vec{K}(I_j)\}$

   where

   $Y_k = \{Y_{k,l} : l = 1, \ldots, \text{Card } U_Z | Y_{k,i} < Y_{k,j} \Leftrightarrow \vec{K}(Z_i) < \vec{K}(Z_j)\}$

   where $n = card\ U_I$ and $m = card\ U_z$

Note that if the criteria for classification, aggregation and sorting are not strictly based on defined categories, the cardinals (Card) used to establish the upper limits of enumerations making part of the corresponding specification, would not be cardinals of sets of categories, but those of subsets (proper or not) of the later.

3. LAST-UML/OCL correspondence

This section describes the correspondence between LAST and UML–OCL. Data definition with LAST is translated into a class diagram where classes and associations represent sets of transactions and categories.

Table 1

Item	Sales region				
	Z_1	$\cdots$	Z_j	$\cdots$	Z_n
I_1	$Y_{1,1}$	$\cdots$	$Y_{1,j}$	$\cdots$	$Y_{1,n}$
$\vdots$			$\vdots$		
I_i	$Y_{i,1}$	$\cdots$	$Y_{i,j}$	$\cdots$	$Y_{i,n}$
$\vdots$			$\vdots$		
I_m	$Y_{m,1}$	$\cdots$	$Y_{m,j}$	$\cdots$	$Y_{m,n}$

Table 1 shows translation for the running example, where the association *Sales_Orders* represents the set of transactions U_T each one with a *Heading* and a set of *Lines*. Attributes D, N, R, Q and P have an attribute associated in the relevant class, typed in the corresponding UML pre-defined data types. According to the referential constraints (case (5) of the LAST specification of U_T) attributes Z, S, C and I are UML associations with the element of the corresponding category (in the table the associations Z and I are the only shown). The transaction-lines cardinality can be expressed through an OCL class constraint (i.e. class invariant): `self.sales_orders` $\rightarrow$ `forall(s | s.lines` $\rightarrow$ `size = s.R)`.

The correspondence in UML of a category type is also a UML class (*Items* for elements of U_I and *Sales_Regions* for elements of U_Z, similarly for U_C and U_S).

With respect to entity integrity some of them are expressed by means of qualified associations, such is the case of the key D and N for *Sales_Orders* (representing (2) of the LAST specification of U_T) and the key K in *Items* and *Sales_Regions* sets, representing the former, the case (1) in the LAST specification of U_I.

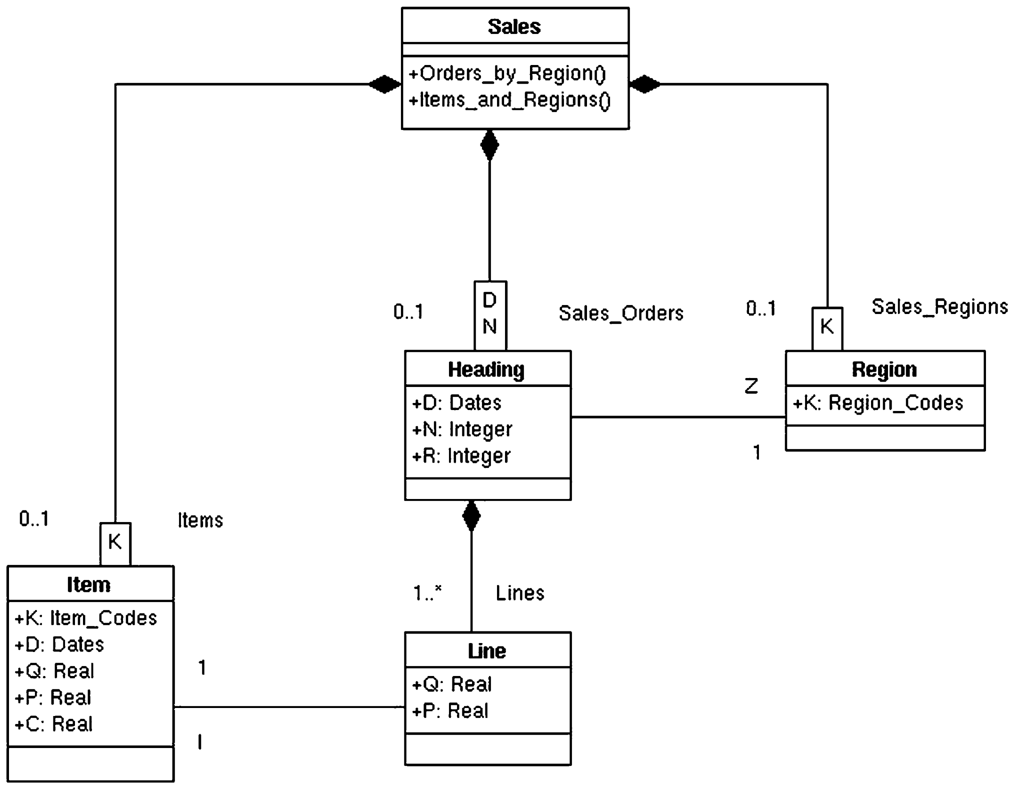
The set of valid values for attributes can be also described by means of OCL constraints, for example in the LAST specification of U_T, (1) can be written as `self.sales_orders` $\rightarrow$ `forall(s|`$\rho \leq$ `s.D and s.D` $\leq \tau$`)`, (3) is expressed as `self.sales_orders` $\rightarrow$ `forall(s|0` $\leq$ `s.R and s.R` $\leq$ `100)`, and similarly for (4).

With respect to the reports description, the combination of UML and OCL makes possible to write pre- and post-conditions for methods of the relevant class. For *non-aggregated reports* the *query parameters* are formal parameters in the method implementing the report, and the *constraints on the parameters* and the *search and retrieval* specification are expressed by means of a OCL pre-(respectively post-) condition.

In some cases, reports can be stored in auxiliary data structures which can be designed in UML and the *sorting* is expressed by means of a {*ordered*} UML clause and the use of OCL type *sequence* in the post-condition. Table 2 (partially) shows this structure for the running example and the report can be specified as follows:

```
+ sales_by_region(in a: date, in b: date,
in k: Region_Codes, out Report1:
```

Table 2
Data definition with UML



```
TReport1)
pre: ρ ≤ a and a ≤ b and b ≤ τ
and  self.sales_regions → exists(r|r.K =
k).
post:
% Search and Retrieval:
Report1.elements → asSet() =
self.sales_orders → select(a ≤ D and
D ≤ b and Z.K = k) and
% Sorting:
Set{1..Report1.elements → size} →
forall(i|
Set{1..Report1.elements → size} →
forall(j|
(Report1.elements → at(i).D <
Report1.elements → at(j).D) or
(Report1.elements → at(i).D =
Report1.elements → at(j).D and
Report1.elements → at(i).N <
Report1.elements → at(j).N)
implies i < j))
```

For *aggregated reports*, *search and retrieval* specification and *sorting* criteria are expressed likewise, but in this case, *classification*, *aggregation* processes have to be considered as well. In most of cases, the entire process can be splited, and consider on one hand, *search and retrieval* and other hand *aggregation*, *classification* and *sorting*. Such is the case of the running example, where we can consider a private method for search and retrieval and a public method implementing the remaining work. The report structure, as in the case of non-aggregated report, may be represented using a data structure,

Table 3
Non-aggregated reports

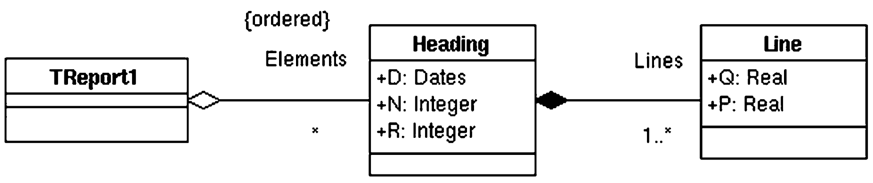
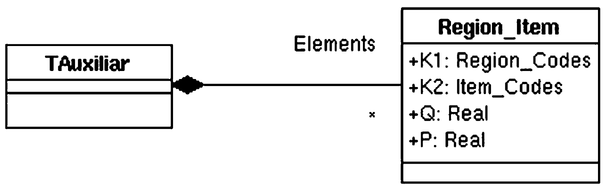


Table 4
Auxiliary data structure



but in addition search and retrieval can be implemented in an auxiliary data structure. Tables 3 and 4 show these structures for the running example, where the output table of the report is represented by means of a association class associating a value *Y* for each item and each region. The auxiliary structure is used to store *LS* as is defined in Section 2.2.2. The methods implementing the report can be specified as follows:

```
-search_dates(in a: date, in b: date, out LS: TAuxiliar)
pre: ρ ≤ a and a ≤ b and b ≤ τ
post:
LS.elements = self.sales_orders →
collect(Z.K,lines → collect(I.K,Q,P)
| a ≤D and D ≤ b)
```

Taking *LS* as input the following method computes the aggregated report in the data structure described in Table 4.

```
+ items_and_regions(in a: date, in b: date, in LS: TAuxiliar, out Report2: TReport2)
pre: ρ ≤ a and a ≤ b and b ≤ τ
post:
% Classification and Aggregation
Report2.items → asSet() = self.items and
Report2.items → forall(i|i.regions →
asSet() = self.sales_regions) and
Set{1..self.items → size} →
forall(i|
Set{1..self.sales_regions → size} →
forall(j|
Report2.items → at(i).regions →
at(j).table.Y = LS.elements →
collect(P * Q|
K1 = Report2.items → at(i).K and
K2 = Report2.items → at(i).regions →
at(j).K) → sum)) and
% Sorting
Set{1..self.items → size} → forall(i|
Set{1..self.items → size} → forall(j|
Report2.items → at(i).K <
Reports.items → at(j).K  implies  i < j))
and
Set{1..self.items → size} → forall(i|
Set{1..self.sales_regions → size} →
forall(j|
Set{1..self.sales_regions → size} →
forall(k|
Report2.items → at(i).regions →
at(j).K <
Report2.items → at(i).regions → at(k).K
implies j < k)))
```

The correspondences between LAST and UML for the treatment of other IS components should not differ substantially from the ones addressed above.

4. LAST-ER/Relational Model correspondence

This section describes the correspondence between LAST and the Entity Relationship and how to implement them into the Relational Model.

4.1. AST and the Entity Relationship Model correspondence

Some of the elements of the Data definition with LAST are translated into Entity Sets and Relationships in the ER model.

In the case of transaction types (of non-fixed length) headings and lines become entities while transactions must be represented by means of relationships.

Table 5 shows translation for the running example, where the entity *Heading* represents each heading of the set of transactions (U_T), *Item* (respective *Region*) is an entity representing the sets of items (U_I) (respectively the sets of sales regions (U_Z)), and *Lines* represents U_T.

The referential constraints are represented by means of relationships. In the table we have considered the relationships *Lines* and *Sales_Regions* associating each line to an item and each heading to a sales region. In addition, it should be considered two additional relations one for each attribute *S* and *C*.

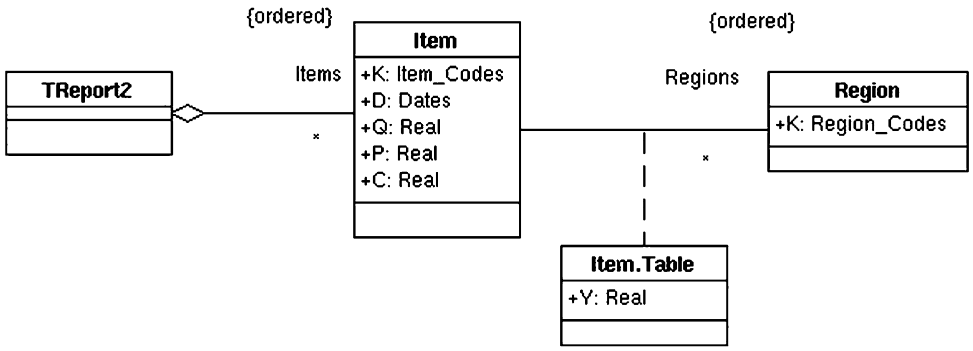
With respect to the entity integrities, they are expressed by means of keys in the ER diagram, for instance, *D* and *N* for headings and *K* for items and sales regions.

4.2. LAST and the Relational Model mapping

Some of the elements of a Data definition with LAST cannot be expressed in the ER model. For instance, time limits constraints will be added to the relational modeling. For instance, the constraint $\vec{D}(T) \in \{x \in \underline{Dates} | \rho \preceq x \preceq \tau\}$ is added in the check clause of the create table SQL statement, when the relational model is considered in our running example, as follows:

```
CREATE TABLE HEADING
(D DATE NOT NULL,
N NUMBER NOT NULL,
```

Table 5
Aggregated report



```
R NUMBER NOT NULL)
PRIMARY KEY (D,N)
CHECK RHO ⇐ D AND D ⇐ TAU.
```

Similarly for the constraint over *R*. In the relational modeling, the relationships become tables as usual, for instance, *Lines*, and similarly *Sales_Regions*, becomes a table using two foreign keys as follows:

```
CREATE TABLE LINES
(D DATE NOT NULL, N NUMBER NOT NULL,
K NUMBER NOT NULL, P NUMBER NOT NULL,
Q NUMBER NOT NULL)
PRIMARY KEY (D,N,K)
FOREIGN KEY (D,N,K)
REFERENCES HEADING(D,N),ITEM(K)
CHECK RHO < = D AND D < = TAU.
```

Finally, each category type is represented by a table:- CREATE TABLE ITEM

```
 (K NUMBER NOT NULL,
 D DATE NOT NULL,
 Q NUMBER NOT NULL,
 P NUMBER NOT NULL,
 C NUMBER NOT NULL)
 PRIMARY KEY K.
```

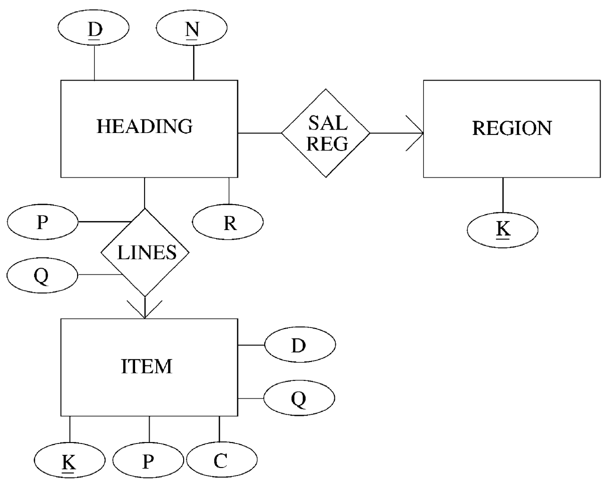
Reports in the LAST specifications can be defined in SQL. For non-aggregated reports, search and retrieval specification and sorting criteria are specified by means of the SQL SELECT together with the ORDER BY statement, for instance in our running example, the non-aggregated report can be specified as follows:

```
SELECT  D,N,S,C,LINES.K,SALES_REGIONS.
K,Q,P
FROM HEADING, LINES, SALES_REGIONS,
SALES_PERSONS, CUSTOMERS WHERE
HEADING.D = LINES.D AND
HEADING.N = LINES.N AND
HEADING.D = SALES_REGIONS.D AND
HEADING.N = SALES_REGIONS.N AND...AND
a < = HEADING.D AND HEADING.D < = b AND
SALES_REGIONS.K < = k
ORDER BY HEADING.D ASC, HEADING.N ASC.
```

For aggregated reports, search and retrieval specification and sorting criteria are expressed likewise, but in this case, classification and aggregation processes have to be considered as well. In this case, search and retrieval specification is implemented as a SQL view, sorting by means of the ORDER BY clause, classification using the GROUPED BY clause and aggregation using derived attributes. In our running example, the report disclosing of the weight of each item in the sales of each sales-region, for a given period $[a, b]$, would be specified as follows:

```
CREATE VIEW LS AS
SELECT  Z  AS  SALES_REGIONS.K,I  AS
LINES.K,Q,P
```

Table 6
Data definition with ER



```
FROM HEADING, LINES, SALES_REGIONS WHERE
HEADING.D = LINES.D AND
HEADING.N = LINES.N AND
HEADING.D = SALES_REGIONS.D AND
HEADING.N = SALES_REGIONS.N AND
a ⇐ HEADING.D AND HEADING.D ⇐ b.
```

Let remark us that this view is corresponded with *LS* as was defined in Section 2.2.2. The output table can be described with the SELECT clause over the view *LS*.`SELECT Z, I, SUM-IJ AS SUM(Q*P)`

```
FROM LS
GROUPED BY Z,I
ORDER BY Z ASC, I ASC.
```

The correspondences between LAST and ER and Relational Model for the treatment of other IS components should not differ substantially from the ones addressed above.

5. Summary and directions for future research

This paper has described the basics of a Formal Method (LAST) that uses elements of linear algebra and set theory for IS-specification. There is plenty of work that needs to be done to make LAST a fully fledged formal method, thus making full use of its potential expressiveness, and to make it viable as a professional SW specification tool, so that it may have a positive impact on IS-development practice. Among others, the following tasks are necessary:

- definition of a procedure for non-ambiguity, consistence and completeness checking of specifications written in LAST;
- development of adequate CASE tools that would include LAST user-interfaces and translators from LAST to specific architectures.

References

[1] S. Agerholm, P.G. Larsen, A light weight approach to formal methods, Proceedings of the International Workshop on Current Trends in Applied Formal Methods, LNCS 1641 Springer, Berlin, 1999, pp. 168–183.
[2] J.M. Almendros-Jiménez, L. Gonzalez, The LAST Project: development of a formal method for IS-specification and of a CASE-tool for IS-design, Proceedings of the Asia-Pacific Software Engineering Conference, APSEC'00, IEEE Computer Society Press, Silver Spring, MD, 2000 pp. 54–61.
[3] D.M. Berry, Formal methods: the very idea some thoughts about why they work when they work, Electronic Notes in Theoretical Computer Science (2000) 25.
[4] D. Bevington, Technical note—business function specification of commercial applications, IBM Systems Journal 39 (2) (2000) 315–335.
[5] J.P. Bowen, M.G. Hinchey, Seven more myths of formal methods, IEEE Software 12 (4) (1995) 34–41.
[6] M. Broy, Software technology—formal methods and scientific foundations, Information and Software Technology 41 (1999) 947–950.
[7] R. Butler, C. Holloway, Impediments to industrial use of formal methods, IEEE Computer (1996) 25–26.
[8] E.M. Clarke, J.M. Wing, Formal methods: state of the art and future directions, ACM Computing Surveys 28 (4) (1996) 626–643.
[9] B. Curtis, H. Krasner, N. Iscoe, A field study of the software design process for large systems, Communications of the ACM 31 (11) (1988) 1268–1287.
[10] R. Duke, G. Rose, Formal Object-Oriented Specification Using Object-Z, Macmillan, New York, 2000.
[11] S. Easterbrook, R.R. Lutz, R. Covington, J.C. Kelly, Y. Ampo, D. Hamilton, Experiences using lightweight formal methods for requirements modelling, IEEE Transactions on Software Engineering 24 (1) (1998) 1–11.
[12] A. Evans, R.B. France, K. Lano, B. Rumpe, Developing the UML as a formal modelling notation, Proceedings of the Unified Modelling Language, UML'98, LNCS 1618, Springer, Berlin, 1999 pp. 336–348.
[13] P. Facon, R. Laleau, H.P. Nguyen, The invoicing system problem: from OMT diagrams to B specifications. Proceedings of the International Workshop on Comparing Systems Specification Techniques, France, IRIN, 1998.
[14] D.C. Fowler, Formal methods in a commercial information systems setting: the FOOM methodology. PhD Thesis, Centre for Information Systems Research, Swinburne University of Technology, Australia, 1996.
[15] R.B. France, J.-M. Bruel, M. Larrondo-Petrie, An integrated object-oriented and formal modeling environment, Object-Oriented Programming 10 (7) (1997) 25–34.
[16] R.B. France, J.-M. Bruel, M. Larrondo-Petrie, E. Grant, Rigorous object-oriented modeling: integrating formal and informal notations, Proceedings of the Algebraic and Software Technology, AMAST'97, LNCS 1349 1997, pp. 216–230.
[17] R.B. France, M. Larrondo-Petrie, A two-dimensional view of integrated formal and informal specification techniques, Proceedings of the International Conference of Z Users, LNCS 967 Springer, Berlin, 1995, pp. 434–448.
[18] R.B. France, R. Busser, M. Boughdadi, Incorporating a Formal Design Technique in an Industrial Setting, Proceedings of the Ninth International Symposium on Software Reliability Engineering (ISSRE98), IEEE Press, New York, 1998.
[19] R.D. Galliers, P.M.C. Swatman, P.A. Swatman, Strategic information systems planning: deriving comparative advantage from EDI, Information Technology 10 (1995) 149–157.
[20] L. Gonzalez, C. Ruíz, Linear algebra and sets theory (LAST)-based formal modeling for IS client-specification, Proceedings of the Pacific Asia Conference on Information Systems, Hong Kong, 2000, pp. 798–817.
[21] A. Hall, Seven myths of formal methods, IEEE Software 7 (1990) 11–19.
[22] D. Hamilton, R. Covington, J.C. Kelly, Experience in applying formal methods to the analysis of software and system requirements, Proceedings of the Workshop on Industrial-Strength Formal Specification Techniques, IEEE Computer Society Press, Silver Spring, MD, 1995 pp. 30–43.
[23] M.G. Hinchey, J.P. Bowen, Applications of Formal Methods, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[24] D. Howe (Ed.), The free on-line dictionary of computing, http://www.foldoc.org.
[25] M. Jackson, Formal methods and traditional engineering, Systems and Software 40 (3) (1998) 191–194.
[26] D. Jackson, J. Wing, Lightweight formal methods, IEEE Computer 29 (4) (1996) 22–23.

[27] C.B. Jones, A rigorous approach to formal methods, IEEE Computer 29 (4) (1996) 20–21.

[28] P.G. Larsen, J. Fitzgerald, T. Brookes, Applying formal specification in industry, IEEE Software 13 (7) (1996) 48–56.

[29] A. van. Lamsweerde, Requirements engineering in the year 00: a research perspective, Invited Paper of the 22nd International Conference on Software Engineering (2000) 5–19.

[30] A. van Lamsweerde, Formal specification: a roadmap, in: A. Finkelstein (Ed.), The Future of Software Engineering, ACM, New York, 2000, pp. 147–159.

[31] The millennium forum, Final discussion paper on: facing the challenges of globalisation: equity, justice and diversity, http://www.millenniumforum.org, 2000.

[32] P. Mukherjee, Computer-aided validation of formal specifications, Software Engineering (1995) 133–140.

[33] J. Mylopoulos, Information modeling in the time of the revolution, Information Systems 23 (3/4) (1998) 127–155.

[34] D.L. Parnas, Formal methods technology transfer will fail, Systems and Software 40 (3) (1998) 195–198.

[35] F. Polack, A case study using lightweight formalism to review an information system specification, Software Practice and Experience 31 (8) (2001) 57–80.

[36] F. Polack, M. Whiston, K.C. Mander, The SAZ project: integrating SSADM and Z, Proceedings of Formal Methods Europe 93: Industrial Strength Formal Methods, LNCS 670, Springer, Berlin, 1993 pp. 541–557.

[37] P.A. Swatman, P.M.C. Swatman, Formal specification: an analytic tool for (management) information systems, Information Systems 2 (2) (1992) 121–160.

[38] P.A. Swatman, D. Fowler, Extending the useful applications domain for formal methods, Proceedings of the Z User Workshop, Springer, Berlin, 1992 pp. 125–144, Workshops on Computing Series.

[39] A. Taylor-Cummings, Brindging the USER-IS gap: a study of major information systems projects, Information Technology (13) (1998) 29–54.

[40] E.N. Wafula, Graphical Representations of Object-Z specifications using MOSES. Mbus Thesis, Centre for Information Systems Research, Swinburg University of Technology, Australia, 1995.

[41] J.M. Wing, A specifier's introduction to formal methods, Computer September (1990) 8–24.

[42] R. Wieringa, E. Dubois, Integrating semi-formal and formal software specification techniques, Information Systems 23 (3/4) (1998) 159–178.

[43] J.B. Wordsworth, Getting the best from formal methods, Information and Software Technology 41 (1999) 1027–1032.