Application of a Model Transformation Paradigm in Agriculture: A Simple Environmental System Case Study

André Miralles, Thérèse Libourel Rouge

Application of a Model Transformation Paradigm in Agriculture: A Simple Environmental System Case Study

André Miralles and Thérèse Libourel

Abstract In this chapter, the authors use the methodology presented in Chapter 2 to develop a system that manages the spreading of organic waste on agricultural parcels. The proposed method uses a process of iterative and incremental development. Two complete iterations of the development process are presented starting from the analysis model and ending with the code produced by the case-tools SQL code generator. The first iteration deals with the description of territory objects and the second one deals with the business objects used in the context of the spreading of organic waste. As a result of transformations applied, models are enriched with new concepts and, therefore, are more complex. The growing complexity of the model may negatively affect an actor's understanding, which may become an impediment by slowing down the analysis phase. The authors show how the software development process model, a modeling artifact associated with the continuous integration unified process method, avoids the apparent complexity of the model and improves productivity.

AQ1

1 Introduction

The main purpose of developing an application is to convert the requirements of the concerned actors (users, clients, stakeholders, etc.) into a software application. To attain this objective –shown in Fig.1– a method for conducting projects is used. Section 2 of Chapter 2 presents different methods for conducting a project.

To overcome this challenge, the developed application should not only correspond with the actors' requirements but should also satisfy them. Experience has shown that it is not easy to be successful in this aim. Often, there is a gap between the requirements expressed initially by the actors and those actually covered by the developed application.

A. Miralles
Centre for Agricultural and Environmental Engineering Research,
Earth Observation and GeoInformation for Environment and Land Development Unit,
Montpellier, France
e-mail: andre.miralles@teledetection.fr

P.J. Papajorgji, P.M. Pardalos (eds.), *Advances in Modeling Agricultural Systems*,
DOI: 10.1007/978-0-387-75181-8_3,

Fig. 1 Software development process [9]

This gap between the requirements that the application actually satisfies and the requirements expressed at the beginning of the project is the cause of most disagreements between the client and the company developing the application. To avoid this very situation, the companies that develop software applications have long adopted software development processes[1] for monitoring the *project's progress* [4] and for *cost and time control* [17].

At the very beginning, the first software applications were developed without any method. The first method to have been formalized and described is the one based on the waterfall cycle [18]. Since then, several development methods have been designed and implemented in different software development projects. In these past 10 years, a significant portion of computer-related research has focused on a multitude of new methods [1, 5, 7, 9, 11, 13, 19] because application-development companies and project leaders have realized the impact that the method used for a project can have on its success.

In the development process, the designer of an application should understand the business concepts of the actors to be able to reproduce as faithfully as possible the properties and the behavior of these concepts in the software components. A good understanding of the concepts is fundamental because this will determine whether development will succeed or fail. If the properties and behavior of the business concepts are not correctly reproduced, the developed application will have a different behavior from the one expected by the actors. To begin with, they will feel disoriented by thinking that they are the cause of the unexpected functioning. Later on, when they realize that they are not responsible for the unexpected functioning, it is likely that they will feel disappointed, even upset, that the application does not meet their expectations.

The understanding of concepts is part of a development phase that is most often called the *analysis phase*. Some methods may give this phase a different name, but the activity conducted during the phase corresponds exactly with analysis. For example, extreme programming calls it the *exploration* phase, during which the client describes the application's features in the form of *user stories*.

By whatever name it may be known, every method of development contains this appropriation phase. This phase always brings together the designer of the application and one or more actors of the concerned domain in analysis sessions. It cannot be otherwise unless one is actor and designer simultaneously, a situation that arises frequently in scientific circles.

---

[1] A software development process is the set of activities needed to transform a user's requirements into a software system.

As detailed in Chapter 2, the hours or days spent on the analysis is fundamentally important, because how well the designer understands and captures the business concepts will depend directly on it. Nevertheless, the number of hours or days spent is not the only important criterion for the appropriation phase. Experience has shown that the frequency of meetings between designers and actors is also important. When the analysis was conducted *in toto* at the beginning of the cycle (waterfall cycle, spiral cycle, RAD[2] method, etc.), the gap between the expressed requirements and those delivered by the application was often large. It is this observation that led to the design and adoption of iterative methods (UP,[3] FDD,[4] etc.). These latter all have the common characteristics of development cycles (analysis, design, implementation, validation) whose durations are fixed in advance, varying from 2 to 4 weeks. These cycles are called iterations. At each iteration, a new "brick" of the application is produced. During an iteration, the actors of the concerned domain participate not only during the analysis but also during the validation (i.e., both at the beginning and at the end of the iteration). This increased participation on the part of the actors allows them to see the application evolve in quasi–real time and to correct as soon as possible any deviation in the development. These new methods are often called *user-centric development*. With its *on-site customer* practice, the extreme programming method has pushed this idea to the limit. This practice involves either including a representative of the client within the development team or to have a team member plays the client's role. In this way, the frequency of the actors' participation becomes infinite. This practice is probably the one with the best communication between the designer and the actors and is the most notable feature of the extreme programming method.

2 The Continuous Integration Unified Process

The observations and the theoretical fundamentals that have led the authors to design and implement software artifacts for implementing the continuous integration unified process method are presented in Chapter 2 and, in much more detail, in Ref. 14. Nevertheless, for reasons of completeness, a brief recap of the broad ideas that form the basis of this method is presented here.

It is an extension of the unified process method [9]. Its specialty is in superimposing a rapid-prototyping cycle on the main cycle during the analysis phase (Fig.2). The objective of this rapid-prototyping cycle is to improve the exchange between actors and designer.

Building a prototype is the same as building an application during the main cycle. The main difference resides in the fact that the prototype has to be built

[2] RAD: rapid application development [13].

[3] UP: unified process [9].

[4] FDD: feature driven development [3].

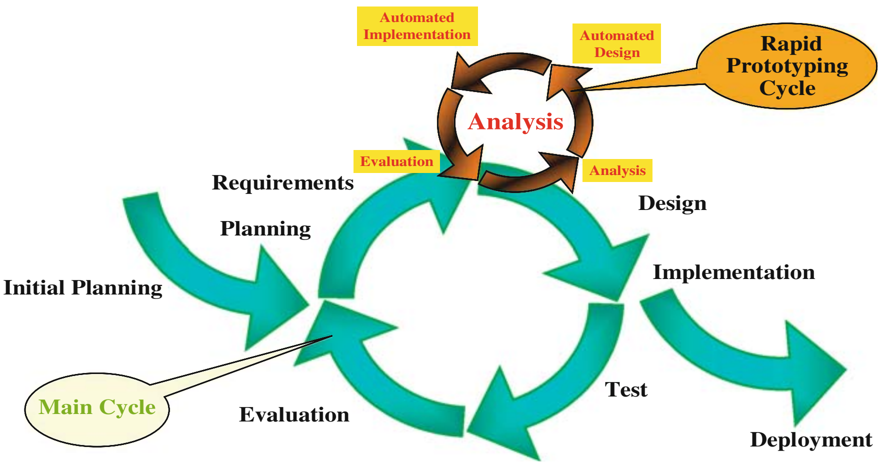


Fig. 2 Continuous integration unified process

automatically via model transformations, otherwise the actors present will soon become bored and lose interest in the ongoing analysis. It is this automated process that Anneke Kleppe calls full MDA[5] [10].

3 Transformations of the Continuous Integration Unified Process in Action

The aim of this section is show how the continuous integration unified process method can be used during the design of an information system.

At the very beginning of the development cycle, all the team has is a very short description of the system, often less than one page long [9]. The goal of the initial planning phase (Fig.2) is to impart "consistency" to the system. To do this, the domain's sponsors should *define the ultimate goal of the system* [6, 12, 15] and *demarcate the boundaries of the concerned domain* [4, 6, 12].

Wastewater from various human activities is now increasingly being processed at water treatment plants. These facilities improve the water quality before discharging it into rivers and streams. The major downside to such a system is the sludge produced. This sludge is heavy with organic wastes and has to be disposed of. One of the solutions currently proposed is to spread this organic waste on agricultural parcels as a replacement in part for nitrogenous fertilizer because of its high nitrogen content.

[5] Model driven architecture (MDA) approach, recommended by the Object Management Group [16].

The *ultimate goal of the system* that we shall present here is to improve the management of organic waste spread on agricultural parcels.

To simplify the case study,[6] the *boundaries of the concerned domain* were limited to parcels belonging to agricultural enterprises, the organic waste to be spread, and the qualitative study of the soil by regular analysis.

This case study is part of the SIGEMO project [21]. The model in Fig.3 is extracted from a general SIGEMO model and consists of concepts relevant to the case study. This model is also used in Chapter 4. It is this model that we shall construct step by step by applying the continuous integration unified process method.

Of course, the modeling process will implement the software development process approach and its reification, the software development process model (SDPM), an artifact for knowledge capitalization. The process will also implement the pictogramic language for expressing spatial and temporal properties of business concepts [2, 14] as well as all the transformations that are necessary to automatically evolve a model from analysis up to implementation. The SQL implementation model thus obtained will then be projected into SQL code.

To simplify the description, we will present here just two iterations: the first will be dedicated to the description of territory objects, and the second will be devoted to the business objects used in the context of the spreading of organic waste.

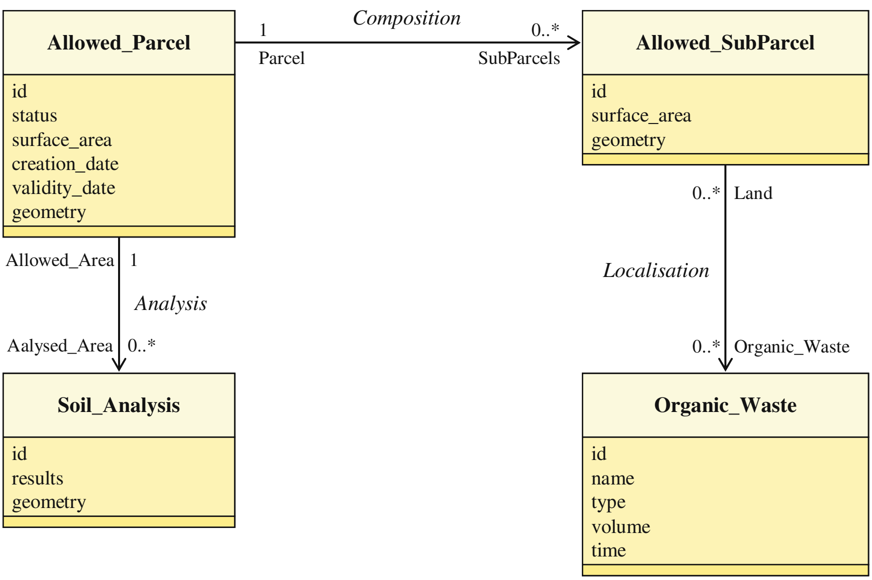


Fig. 3 Extract of the SIGEMO model

[6] This decision is based on the fact that models become complex very rapidly and hence difficult to read.

As mentioned above, the main interest in the continuous integration unified process method lies in its use of the rapid-prototyping cycle during the analysis phase. Because of this, we shall focus our explanations on this development phase. We recall that during this phase, the concerned domain's actors have to participate in the system analysis and, as such, will be present during the different analysis sessions.

3.1 *Construction of the Software Development Process Model*

We have decided to devote a section to describing the construction of the software development process model so as to simplify subsequent explanations. This is possible, on the one hand, because its construction can take place at the same time as the model's transformations and, on the other hand, because its construction always takes place during the first iteration. Once created, its structure does not change unless, during the development, the need arises to use a new programming language. In that case, a new implementation model becomes necessary.

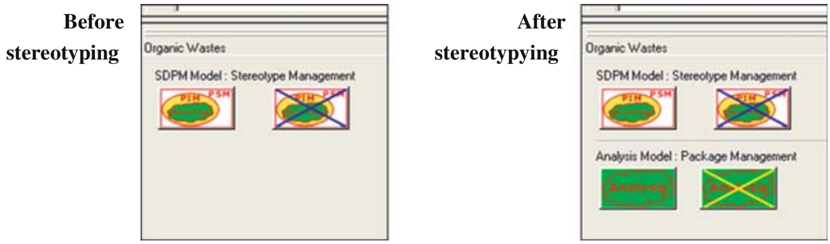
As said by Muller and Gaertner [15], the models are reified within packages. Because of this, the designer has to first specify the SDPM package that will contain all the analysis, design, and implementation models. To do this, the designer should annotate the chosen package with the *<<MDA: Software Development Process Model>>* stereotype. This Unified Modeling Language (UML) model element is associated with the pictogram . The SDPM package is shown in Fig. 4. To add this stereotype, click the button (Fig. 5). If the SDPM package is empty, it is possible to remove the stereotype *<<MDA: Software Development Process Model>>* by clicking the button.

Once the SDPM package is created, two new functionalities become available to the designer, which were not available earlier (Fig. 5). The first allows the designer to create the analysis model package (the button) and the second to delete it (the button).

A click on the button will create the analysis model (Fig. 4). The name of this package, "Organic Wastes – Analysis Model," is obtained by concatenating the SDPM package's name with the extension "- Analysis Model."

Fig. 4 Example of the structure of software development process model

AQ3



AQ4 **Fig. 5** Case-tool interface before and after stereotyping the SDPM package

To create the package for the next model, the designer should first select the package of the analysis model. Only then will the software toolbox make available to the designer the buttons for creating the package of the preliminary design model. As above, the name of the new package, "Organic Wastes – Preliminary Design Model," is the result of concatenating the SDPM package's name with the extension "- Preliminary Design Model."

Step by step, by selecting the last package created, the designer can thus construct the architecture of the SDPM, as shown in Fig. 5.

In the example that follows, only the SQL implementation model will be created as the aim is to show the full MDA process that was designed and implemented.

3.2 *First Iteration*

During the first analysis session, the actors and the designer will identify and describe the domain's relevant objects. Here, the important real-world objects are *Allowed_Parcels* and *Allowed_SubParcels*. The latter are geographical partitions of the former; that is, an *Allowed_Parcel* is composed of a set of *Allowed_SubParcels*. The cardinalities 1 and 0..* respectively of the classes *Allowed_Parcel* and *Allowed_SubParcel* translate this geographic structure (Fig. 6). AQ5

The concept of spreading brings with it the concept of surface area. This leads the designer to introduce the concepts of geometry and area. In the analysis model, the geometry is represented by the pictogram [2] and the area by the *surface_area*

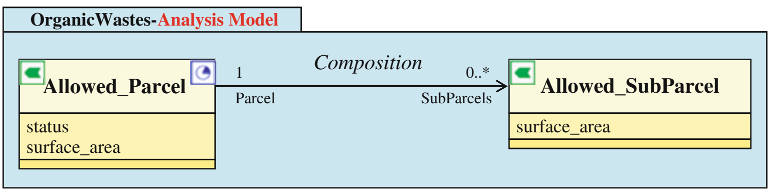


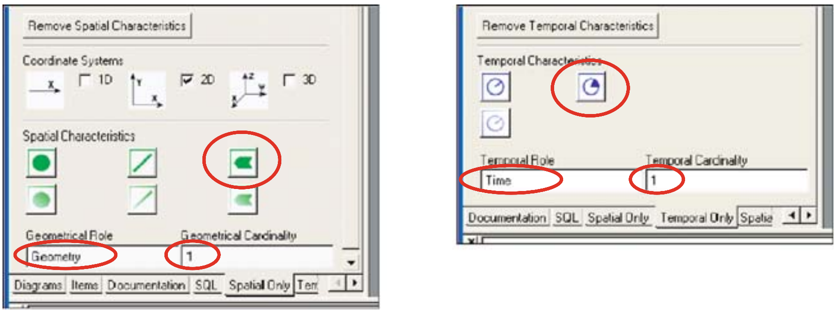
Fig. 6 The analysis model: iteration one

attribute. The authorization to spread organic wastes on the *Allowed_Parcels* begins on a given date and is not indefinite. This signifies that the spreading on an *Allowed_Parcel* is of a limited duration, which starts and ends on precise dates (i.e., the concept of period). In the analysis model, the concept of period is represented by the pictogram. Finally, the domain's actors would like to know if the *Allowed_Parcel* can always be used (or not) to spread organic wastes. The *status* attribute shows and stores this information.

To annotate the business concepts with pictograms, the designer can use the interfaces shown in Fig. 7. They are displayed only when the designer works on the analysis model. He has to fill in the circled fields and click the button with the pictogram of the spatial ( ) or temporal ( ) concept.

As the first iteration's analysis model is now created, the *Diffusion Transformation*[7] can be used to clone an analysis model within the preliminary design model (Fig. 8).

Subsequently, the *GIS design pattern generation transformation*[7] enriches the preliminary design model with spatial and temporal concepts corresponding with the pictograms used in the model, *Polygon* and *Time_Period* in our case. The state of the model at this stage can be seen in Fig. 9.[8] The business concepts,



AQ6

Fig. 7 Example of spatial and temporal user interfaces

Allowed_Parcel and *Allowed_SubParcel*, do not yet have any relationship with the spatial and temporal concepts that were added by the *GIS design pattern generation transformation*. This transformation applies on the preliminary design model and modifies only it. It does not alter the analysis model in any way.

The following stage consists of concretizing the relationships between the business concepts and the *Polygon* and *Time_Period* concepts. This is the job of the *pictogram translation transformation*.[7] This transformation links the business concepts annotated with pictograms to the corresponding spatial and

AQ7

[7] See Chapter 2.

[8] For the sake of simplification, the complete *GIS design patterns* are not shown on this figure, nor on the following ones. Only the spatial and temporal concepts corresponding with the pictorgrams used during the modeling have been added. For additional information on the *GIS design patterns*, refer to Ref. 14.

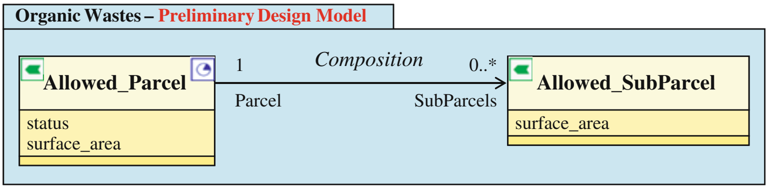


Fig. 8 The preliminary design model: iteration one

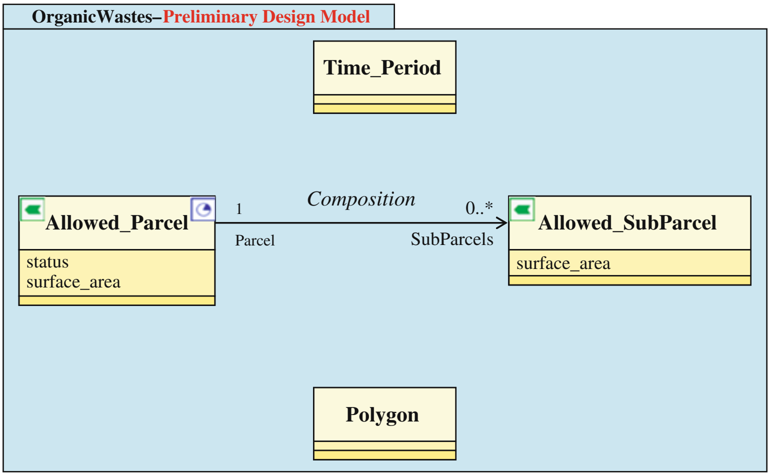


Fig. 9 The preliminary design model after the enrichment by the *design pattern generation transformation*: iteration one

temporal concepts. To do this, it creates *Spatial_Characteristic* and *Temporal_-Characteristic* associations. It also defines the roles (*Geometry* or *Period*) and the cardinalities (1) of the spatial and temporal concepts. It also specifies the roles (*Allowed_Parcel* or *Allowed_SubParcel*) and the cardinalities (0..*) of the business concepts. This latest evolution of the preliminary design model is shown in Fig. 10.

The *GIS design pattern generation* and *pictogram translation transformations* are triggered when the designer clicks the  and  buttons, respectively. This user interface is only displayed by the case-tools when the designer is working with the preliminary design model.

To continue the full MDA process, the preliminary design model is in its turn cloned into the advanced design model. We shall not linger on this latter model because, as of now, we have not identified and implemented any transformation on it.

Once again, the *diffusion transformation* is executed to clone the advanced design model into the SQL implementation model. At this stage, it is identical to that in Fig.10.

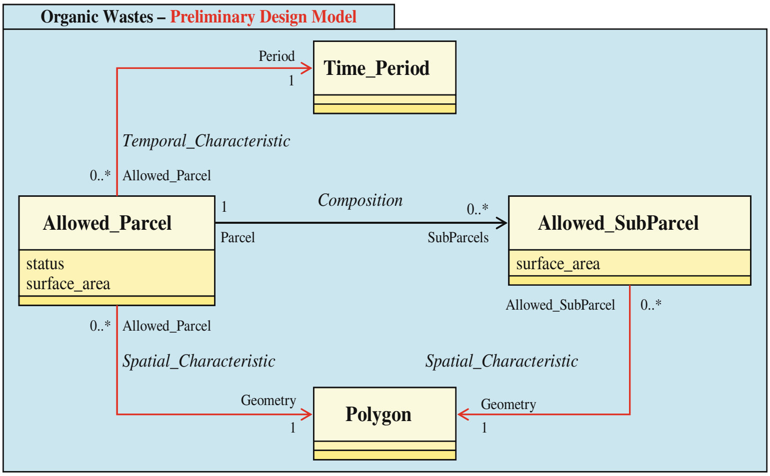


Fig. 10 The preliminary design model after the run of the *pictogram translation transformation*: iteration one

To be able to finalize the full MDA process, this last model has to be complemented with information specific to the SQL language and interpretable by the *SQLDesigner* code generator of the case-tools. The *SQL transformation*[7] was designed and implemented to assume this role. The added information is the *persistence* and an attribute. The attribute is introduced to play the role of *Primary Key*. Its name is made up of that of the class preceded by the "ID_" character string. A tagged value {*primaryKey(1)*} confers upon it this status (Fig.12). For all practical purposes, the *SQL transformation* automates the actions that would have had to be undertaken manually.

The SQL implementation model is now ready to be projected into SQL code. The projection rules are implemented in the SQLDesigner generator developed by the Softeam company. They are documented in Ref. 20. This generator produces a SQL script that contains all the queries that allow tables to be created in a database management system (DBMS)[9]. We provide below the code for creating the *tbAllowed_Parcel* and *tbPolygon* tables.

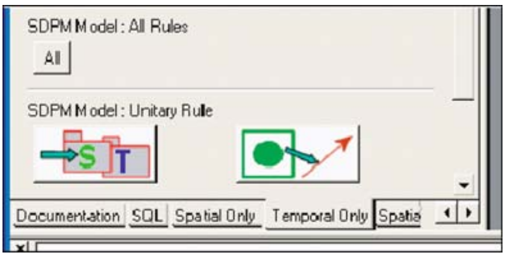


Fig. 11 User interface for GIS transformations

[9] To facilitate the comprehension of the codes used, the table names have been prefixed by "tb" and those of foreign keys suffixed by "_FK."

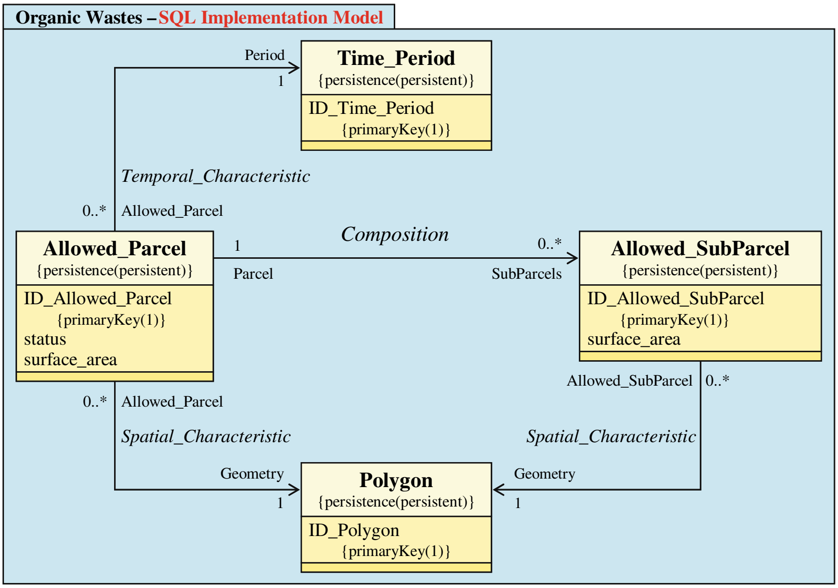


Fig 12 The SQL implementation model after the run of the *SQL transformation*: iteration one

```
CREATE TABLE tbAllowed_Parcel(
   ID_Allowed_Parcel INTEGER NOT NULL ,
   status INTEGER ,
   surface_area FLOAT ,
   ID_Polygon_FK INTEGER NOT NULL ,
   ID_Time_Period_FK INTEGER NOT NULL );


CREATE TABLE tbPolygon(
   ID_Polygon INTEGER NOT NULL );
```

The SQLDesigner generator also produces queries for indicating which is the table's field that plays the role of primary key. As shown by the code below, the primary key of the *tbAllowed_Parcel* table is the *ID_Allowed_Parcel* field and that of the *tbPolygon* table is the *ID_Polygon* field.

```
ALTER TABLE tbAllowed_Parcel ADD(
   CONSTRAINT tbAllowed_Parcel_PK PRIMARY KEY (ID_Allowed_
    Parcel));
```

```
ALTER TABLE tbPolygon ADD(
  CONSTRAINT tbPolygon_PK PRIMARY KEY (ID_Polygon));
```

Finally, the script also contains queries for establishing integrity constraints between the primary and foreign keys of the tables. This code shows an example:

```
ALTER TABLE tbAllowed_Parcel ADD(
  CONSTRAINT Geometry1_of_tbAllowed_Parc_FK FOREIGN KEY
  (ID_Poly- gon_FK)
    REFERENCES tbPolygon(ID_Polygon));

ALTER TABLE tbAllowed_Parcel ADD(
  CONSTRAINT Period_of_tbAllowed_Parcel_FK FOREIGN KEY
  (ID_Time_Pe- riod_FK)
    REFERENCES tbTime_Period(ID_Time_Period));
```

Loaded into a DBMS, the script creates database tables with their primary and foreign keys and the integrity constraints linking these two key types. The creation of the tables in the DBMS ends the first iteration.

Even though the database is complete, it should on no account be populated with data because, at the end of the first iteration, the analysis is far from over. Several more iterations will be necessary to stabilize the initial analysis model. In the following section, we present the major stages of the second iteration.

3.3 Second Iteration

After having described the territory that will be subject to spreading, the business concepts specific to the domain are identified and added to the analysis model of Fig. 6. The main concept is *Organic_Waste*, which is spread on a given date, whence the presence of the pictogram. To be able to monitor the proper spreading of the organic waste, the rules often require recording of name, type, and volume spread. These three items of information are characteristics, which are included as attributes in the *Organic_Waste* class. With these roles and its cardinalities, the *Localization* association allows the listing of the products spread on an *Allowed_SubParcel*.

The rules also require that *Soil_Analysis* be conducted to determine the composition of the *Organic_Waste* spread. The spatial position of the soil sampling is an important property that the information system has to manage.

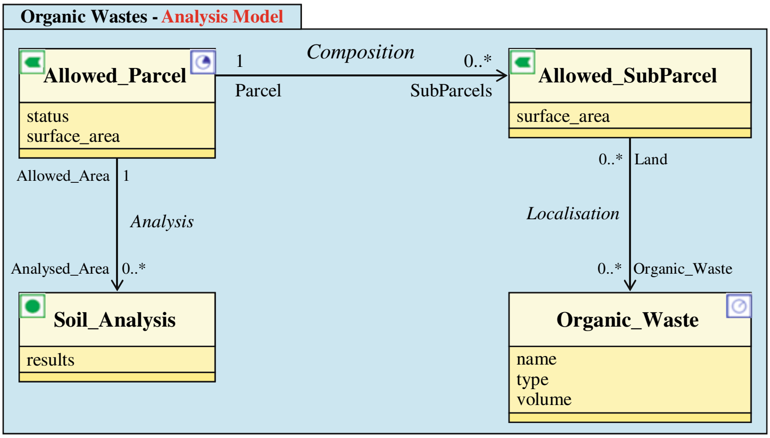


Fig. 13 The analysis model: iteration wo

This is the reason for the pictogram. The *Analysis* association establishes a relationship between the *Soil_Analysis* and the parcels from which samples have been collected.

The analysis model in Fig. 6 is now enriched by two new concepts (Fig. 13) that have to be diffused and processed in the design and implementation models of the software development process model to modify the database of the first iteration.

After the diffusion of the new concepts to the preliminary design model, the *pictogram translation transformation* establishes the *Spatial_Characteristic* and *Temporal_Characteristic* relationships with the *Point* and *Time_Point* concepts. These spatial and temporal concepts corresponding with the and pictograms were masked in the previous iteration to simplify the different models. The model resulting from this transformation is shown in Fig. 14. It is easy to see that the complexity of the model is increasing rapidly.

As earlier, the new concepts of the preliminary design model are diffused first and foremost to the advanced design model.

Because no transformation is available on this model, the new concepts are once again diffused to the SQL implementation model. The *SQL transformation* adds the persistence and the attributes that will play the role of the primary key in the classes that do not have them. The result of these transformations is shown in Fig. 15.

The *SQLDesigner* code generator is now ready to use the contents of the SQL implementation model to produce a SQL script containing the queries for generating the new database. This script consists of 33 queries, some of which we have extracted below to show the final result.

We have reproduced the code that creates the *tbAllowed_SubParcel* and *tbOrganic_Waste* tables and that implements the *Localization* association, which has the distinctiveness of being a N,N association (Fig. 6). This association typology necessitates the creation of the *Localization* table.

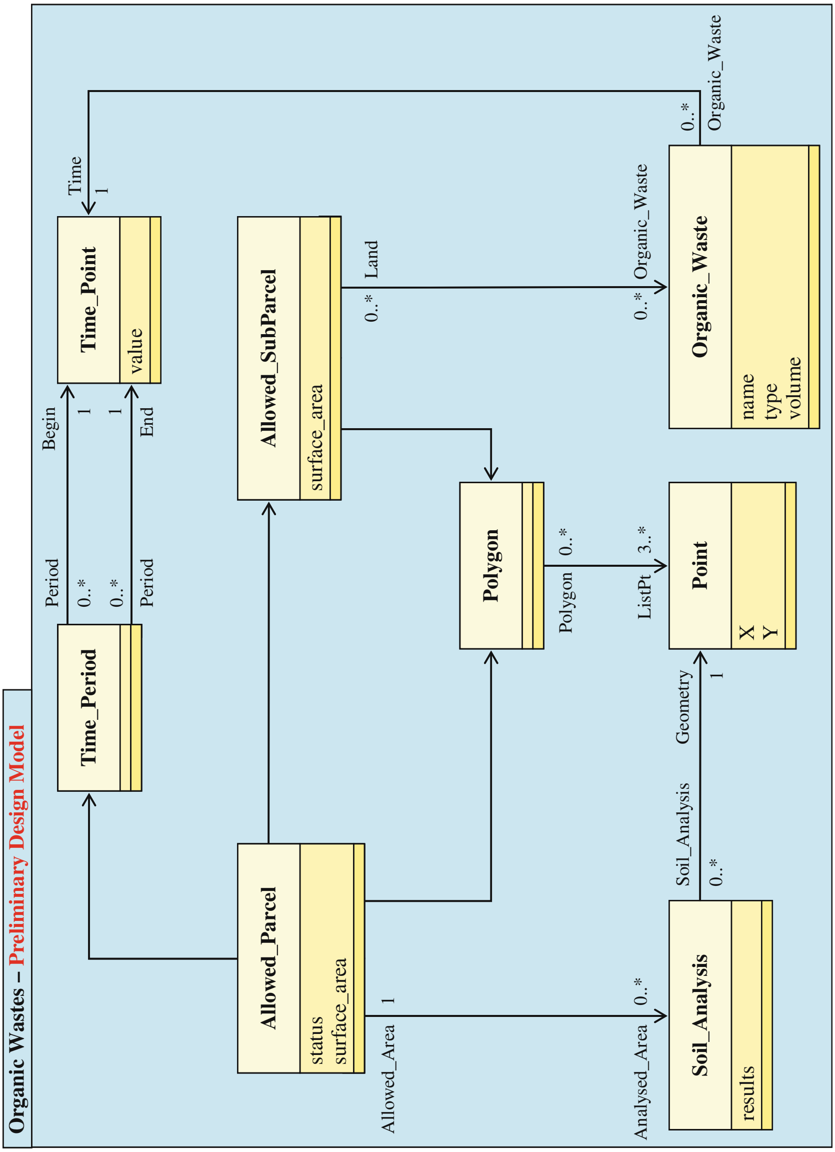


Fig. 14 The preliminary design model: iteration two

```
CREATE TABLE tbAllowed_SubParcel(
  ID_Allowed_SubParcel INTEGER NOT NULL ,
  surface_area FLOAT ,
  ID_Allowed_Parce_FK1 INTEGER NOT NULL , ID_Polygon
```

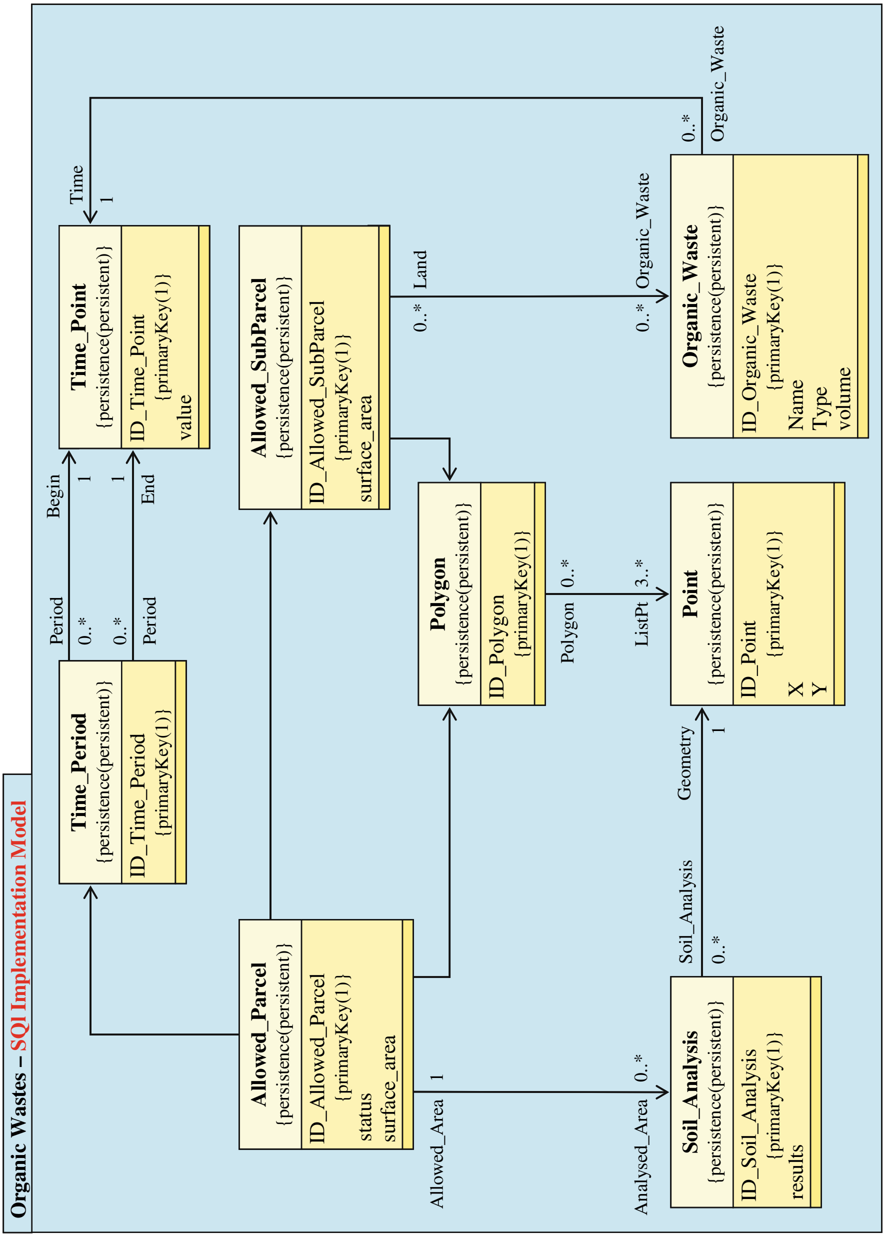


Fig. 15 The SQL implementation model: iteration two

```
INTEGER_FK NOT NULL );CREATE TABLE Localization(
  ID_Allowed_SubParcel_FK INTEGER NOT NULL ,
  ID_Organic_Waste_FK INTEGER NOT NULL );

CREATE TABLE tbOrganic_Waste(
  ID_Organic_Waste INTEGER NOT NULL ,
  name VARCHAR2(50) ,
  type VARCHAR2(50) ,
  volume FLOAT ,
  ID_Time_Point_FK INTEGER NOT NULL );
```

In the *Localization* table, the primary key is a composite of the foreign keys of the two tables *tbAllowed_SubParcel* and *tbOrganic_Waste*.

```
ALTER TABLE Localization ADD(
  CONSTRAINT Localization_PK PRIMARY KEY (ID_Allowed_
  SubParcel, ID_Organic_Waste));
```

4 Conclusions

We have shown over two iterations how the continuous integration unified process method is used and how models evolve in the full MDA process recommended by the Object Management Group.

Moreover, as described above, the continuous integration unified process method proceeds using successive enrichments. The concepts added progressively in the analysis, design, and implementation models increase the model's complexity and render it increasingly difficult to understand. The assimilation of the SQL implementation models of Figs. 12 and 15 will take longer than that of the analysis models from which they originate (see Figs. 6 and 13, respectively). From this observation arises one of the major benefits of the modeling artifact software development process model.

If the three model transformations, *GIS design pattern generation*, *pictogram translation transformations,* and *SQL transformation*, had all been applied to the same model, the analysis model of Fig. 6 would not exist at the time of the first iteration. The actors would have been confronted by the model in Fig.12, which is much more complex than the model of Fig. 6. They would have been disturbed or even upset by a model enriched by a whole host of concepts whose significance they would not have necessarily understood.

With the software development process model, this potential inconvenience disappears. In fact, during the course of an iteration, the analysis model remains unchanged; it is the design and implementation models that are enriched. At the beginning of the next iteration, the actors thus find the same analysis model. They therefore feel comfortable and do not have to make an extra effort to understand the significance of concepts added during the iteration because they do not see them. This leads to a direct gain in productivity during the analysis.

References

1. Beck K. 2000. eXtreme Programming Explained – Embrace Change. Addison-Wesley. 190 pp.
2. Bédard Y, Larrivée S, Proulx M-J, Nadeau M. 2004. Modeling Geospatial Databases with Plug-ins for Visual Languages: A Pragmatic Approach and the Impacts of 16 Years of Research and Experimentations on Perceptory. Presented at ER Workshops 2004 CoMoGIS, Shanghai, China.
3. Bénard J-L. 2002. Méthodes agiles (6) – Feature Drive Development. Développeur Référence. http://www.devreference.net/devrefv210.pdf. Last access: September 2004.
4. Bénard J-L. 2002. Méthodes agiles (7) – Unified Process. Développeur Référence. http://www.devreference.net/devrefv212.pdf. Last access: September 2004.
5. Boehm BW. 1988. A Spiral Model of Software Development and Enhancement. In: IEEE Computer, pp. 61–72.
6. Booch G, Rumbaugh J, Jacobson I. 2000. Guide de l'utilisateur UML. Eyrolles. 500 pp.
7. Fowler M, Highsmith J. 2001. The Agile Manifesto. Software Development magazine. http://www.sdmagazine.com.
8. Highsmith J. 2002. Agile Software Development Ecosystems. Addison-Wesley Professional. 448 pp.
9. Jacobson I, Booch G, Rumbaugh J. 1999. The Unified Software Development Process. Addison-Wesley. 463 pp.
10. Kleppe A. 2004. Interview with Anneke Kleppe. Code Generation Network. http://www.codegeneration.net/tiki-read_article.php?articleId=21. Last access: August 2006.
11. Kruchten PB. 1999. The Rational Unified Process: An Introduction: Addison-Wesley Professional. 336 pp.
12. Larman C. 2002. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice Hall PTR. 627 pp.
13. Martin J. 1991. Rapid Application Development. Macmillan Publishing. 788 pp.
14. Miralles A. 2006. Ingénierie des modèles pour les applications environnementales. Thèse de doctorat. Université Montpellier II, Montpellier. http://www.teledetection.fr/ingenierie-des-modeles-pour-les-applications-environnementales-3.html. 322 pp.
15. Muller P-A, Gaertner N. 2000. Modélisation objet avec UML. Eyrolles. 520 pp.
16. OMG. Object Management Group home page. http://www.omg.org/. Last access: October 2004.
17. Roques P, Vallée F. 2002. UML en Action – De l'analyse des besoins à la conception en Java. Eyrolles. 388 pp.
18. Royce WW. 1970. Managing the Development of Large Software Systems. Presented at IEEE Westcon, Monterey, CA.
19. Schwaber K, Beedle M. 2001. Agile Software Development with Scrum. Prentice Hall. 158 pp.
20. Softeam. 2003. Objecteering/UML – Objecteering/SQL Designer User guide – Version 5.2.2. 236 pp.

21. Soulignac V, Gibold F, Pinet F, Vigier F. 2005. Spreading Matter Management in France within Sigemo. Presented at 5th European Conference for Information Technologies in Agriculture (EFITA 2005), Vila Real, Portugal.
22. Vickoff J-P. 2000. Méthode RAD – Éléments fondamentaux. http://mapage.noos.fr/rad/radmetho.pdf. 32 pp.

Application of a Model TransformationTransformation Paradigm in Agriculture: A Simple Environmentalenvironmental System Case Study

Query No.	Line No.	Query
AQ1	18	Define acronym SQL.
AQ2	186	Define acronym SIGEMO.
AQ3	269	Please provide better quality figure
AQ4	279	Please provide better quality figure
AQ5	303	We have inserted the figure citation for “Figure 6” to maintain the sequential order.
AQ6	345	Please provide better quality figure
AQ7	353	Clarify use of superscript “7”; is a reference to footnote 7 intended?
AQ8	427	Clarify use of superscript “7”; is a reference to footnote 7 intended?
AQ9	446	Please provide a text citation for figure 11.
AQ10	731	Please provide text citations for references 3, 8, 16, 22.
AQ11	741	In Ref. 5, provide location of publisher.
AQ12	743	In Ref. 7, provide web address of article; provide date of access.
AQ13	764	In Ref. 20, provide publisher.