A new methodology to automate the transformation of GIS models in an iterative development process

André Miralles, Thérèse Libourel Rouge

A New Methodology to Automate the Transformation of GIS Models in an Iterative Development Process

André Miralles and Thérèse Libourel

Abstract In the majority of research today in areas such as evaluation of flood risks, management of organic waste as it applies to plants, and mapping ecological conditions of rivers, scientific advances are often aimed toward the development of new software or the modification of existing software. One of the particulars for software developed for agricultural or environmental fields is that this software manages geographic information. The amount of geographic information has greatly increased over the past 20 years. Geographic Information Systems (GISs) have been designed to store this information and use it to calculate indicators and to create maps to facilitate the presentation and the appropriation of the information. Often, the development of these GISs is a long and very hard process. Since the early 1970 s, in order to help project managers, software development processes have been designed and applied. These development processes have also been used for GIS developments. In this chapter, the authors present a new methodology to realize GIS more easily and more interactively. This methodology is based on model transformations, a concept introduced by the Object Management Group (OMG) in its approach called model driven architecture (MDA). When software is developed, models are often used to improve the communication between users, stakeholders, and designers. The changes of a model can be seen as a process where each action (capture of user concepts, modification of concepts, removal of concepts, etc.) transforms the model. In the MDA approach, the OMG recommends automation of these actions using model transformations. The authors have developed a complete set of model transformations that enable one to ensure the evolution of a GIS model from the analysis phase to the implementation phase.

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_2,

1 Introduction

The development of a software application is becoming increasingly difficult. Since the earliest developments of software, many methodologies have been designed and used to help the project leader in developing software. Over the past 15 years, Ivar Jacobson, Grady Booch, and James Rumbaugh have been major contributors to the improvement of the methodologies used to develop software [12]. They define a ***software development process*** as *the set of activities needed to transform a user's requirements into a software system* (Fig. 1).

These authors have also formalized the various "ingredients" taking part in the process of developing a computer application. This model is called the 4Ps model (Fig. 2).

This model dictates that the *Result* of a ***Project*** is a ***Product*** that requires ***People*** in order to describe the studied domain (actors) and to manage it (analysts, designers, programmers, etc.). The realization of the ***Project*** is conducted in accordance with *Templates* defining, organizing, and explaining the successive steps of the development ***Process***. In order to manage the development ***Process***, ***Tools*** facilitating the expression of the needs, the modeling, the project planning, and so forth, are needed.

This description of the development process paints a set of variety of topics and issues that a project manager in charge of application development should address. To illustrate the intrinsic complexity of a development, Muller and Gaertner [21] use two metaphors reported here *in extenso*:

Fig. 1 Software development process [12]

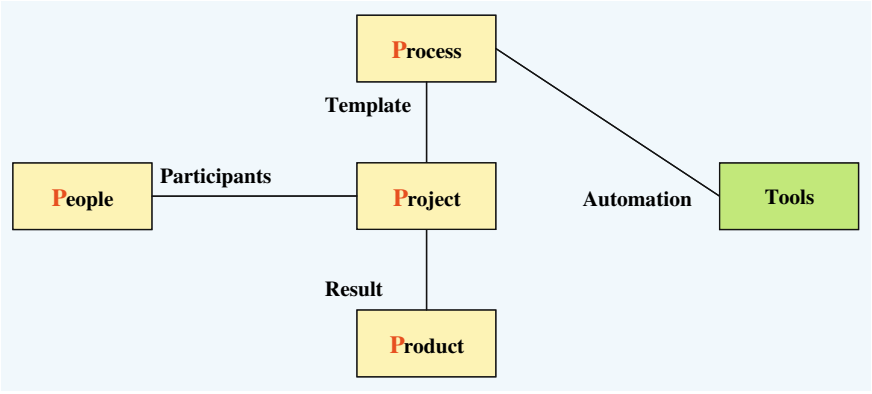


Fig. 2 The 4 Ps software development process [12]

- The first one is related to the management of the project: the development of software can be seen as the crossing of an ocean in a boat. The departure day is known; the arrival is not so well-known. In the course of the trip, it will be necessary to brave storms and to repair damage.
- The second one concerns the multidisciplinary character of necessary competencies to make a development. They write: if the computer programmer had produced furniture, he would begin by planting acorns, would cut up trees into planks, would dig the soil in search of iron ore, would manufacture ironworks to make his nails, and would finish by assembling everything to obtain a piece of furniture. . . . A mode of development which rests on the shoulders of some heroic programmers, gurus, and other magicians of software does not constitute a perennial and reproducible industrial practice.

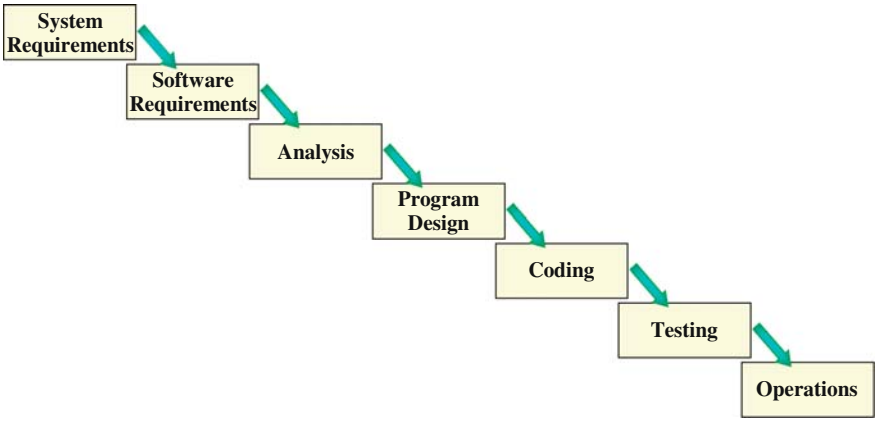
These two metaphors perfectly illustrate the challenge with which the project leader and the programmers are confronted when they take on the realization of a data-processing application. This challenge is not entirely imaginary. The statistics of Ref. 30 give an idea of the difficulty. According to these statistics, the failure risk of the development of an application is 23%, the risk of drift is 49%, and only 28% of the developments have to finish in delay and in the projected budget. These figures are from an investigation carried out on more than 150,000 developments achieved in the United States in 2000. It is noteworthy that in this investigation, the developments aimed at creating a new application are grouped with those aimed at the evolution of an existing application. Thus, it is quite likely that the figure of 28% is overestimated, as the failure risk linked to achieving a new application is much larger than the risk linked to the evolution of an application.

2 The Software DevelopmentProcess

The creators of the Unified Modeling Language (UML) have deliberately failed to define a methodology to successfully carry out a project of development [8] in order to let each designer freely choose the most suitable method adapted to his professional environment. Generally, the designer uses methods of project leading with the aim of *increasing the satisfaction level of the customers or of the stakeholders while making the development work easier* [3] and more rationally

There are a wide variety of software development processes that can be classified into two large families:

- The so-called traditional methods (waterfall life-cycle, V life-cycle, spiral life-cycle, unified process, rapid application development, etc.) are derived most often from the methods used in industrial engineering or in civil engineering (i.e., building and public works sector) [17].
- The *agile* methods, of which the most important are extreme programming, dynamic software development method, adaptive software development, SCRUM [3, 17], and so forth. Their major characteristics are their potential for adaptation and *common sense in action* [3].



AQ3 **Fig. 3** The 4 Ps software development process [26]

The software development processes that were used in the 1970 s were essentially of the linear type (Fig. 3); that is to say that the analysis was conducted at the start of the project. Then, the development of the application was conducted, step after step, without any intermediate validation from users, stakeholders, or sponsors. At the end of the process, the application was presented to the stakeholders. It was not rare that the application did not correspond with the needs of the users, the stakeholders, or the sponsors. In this case, the project manager was professionally in a difficult position, especially if the duration of the development was long (several months and even 1 or 2 years).

This situation is hardly surprising because it is difficult, and even impossible, for the users or the stakeholders, *to conceive the solution in its whole* [5]. This report is all the more true when the scale of the project is substantial.

To avoid facing this type of situation, the project managers have appealed more and more to the users and the stakeholders during the development process to validate the application progress.

The experience cumulated during this type of development processes enables better formalization of the participation of the stakeholders in the development of computer applications and allows for the proposal of new methods to conduct the project. The unified process method and extreme programming method are two key methods coming from this line of thinking.

The unified process method, relying on the modeling language UML, is the synthesis of the best practices of software development over three decades in various fields[1] [12]. It assumes the adoption of the following four principles: the development process should be *use-case driven,* but it should also be *iterative and incremental*, *architecture-centric,* and *risk-centric* [16, 21, 25].

[1] Telecommunication, aeronautic, defense, transport, and so forth.

The *use-case driven* development principle has been introduced by Ivar Jacobson [11] in order to pilot application development according to the requirements of the users or the stakeholders. A use-case is *a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system* [23]. This concept enables one to describe what the system should do. Once implemented, a use-case compulsorily resolves a requirement. If this is not the case, it is because the need was not properly described. Thus, it is important to describe the use-cases at the beginning of the project. In this vision, the use-cases can be used as a planning tool during the development.

The *iterative* development process (Fig. 4) has been designed to prevent the drawbacks caused by linear development. In order to do this, the system is structured into subsystems, and for each iteration, a subsystem is analyzed and implemented. Therefore, the model evolves following an *incremental* process. For Ivar Jacobson, *an increment is the result of an iteration* [12] that lasts between 2 and 4 weeks.

The principle of a development process that would be *architecture-centric* assumes that the structuring in subsystems must not be a simple description of the system under a graphic or a textual form, but that it should be materialized by a model in a *case-tool* [25].

The aim of a *risk-centric* development is to put as a priority the achievement of the systems or subsystems for which the designers have the least experience: implementation of new technologies, for instance. This principle of development enables one to take issues into account very early and to process them by anticipation.

Extreme programming [1, 4] is a method called *agile*, which recommends reducing activities that are not closely related to the production of a code, including documentation. The code is the main part of the production of the team. This method is hence often qualified as code-centric development. It is representative of the agile methods that rely on four values:

- **Communication** between the users, the stakeholders, and the designer to prevent situations described in the waterfall method.
- **Simplicity** of the code so that it is easily understandable and it is possible to integrate changes.

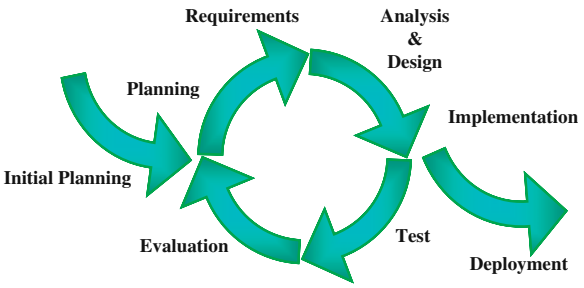


Fig. 4 Typical iteration flow in unified process [15]

- **Feedback,** which should be quick from the stakeholders and from the other members of the development team, enables the developer to have information on the quality of his development.
- **Courage** to tell things as they are and to make difficult decisions like changing a code structure or throwing it away [6].

The fulfillment of these four values is ensured by 12 practices with the aim of encouraging *quick feedback*, favoring the *incremental evolution* of the code, seeking *simplicity* in the produced code, and targeting the *code quality*.

Among these practices, that of *customer on-site*[2] is probably the most important. The aim of this practice is to fluidize the communication between the customer and the programmers by hosting the customer or his representative within the team. This practice ensures a strong reactivity and a high feedback. The main aim of this practice is to make up for the lack of detailed specifications.

The main task of the customer is the writing of the *user stories*, which will allow one to code the functionalities of the application. A second task that is just as important as the first one is the determination of the tests that the tester should implement to validate functionalities. The customer acts by fixing the priorities among the functionalities, by stating the specifications that have not been previously defined or that have remained fuzzy during the previous discussions, and so forth.

The presence of the customer in the team enables him to see the immediate result of his work of specification and to evaluate the progression of the application. This closeness also enables him to quickly assess the relevance of his specifications. If the project drifts or progress is slow, he will immediately realize it.

Actually, the practice of *customer on-site* gives a high level of interactivity to the development process, which associated with practice of the test-driven development reduces the number of bugs by a factor of 5 in some cases.

3 The Model Driven Architecture[3]

Model driven architecture (MDA) is a software design approach proposed by the Object Management Group (OMG) with the objective of improving application developments. It was conceived and formalized in 2001 to improve productivity but also to resolve problems of software portability, software integration, and software interoperability encountered during developments [14].

To achieve this objective, the MDA approach recommends that designers separate the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform [18]. For that, the authors of this approach suggest use of two types of models: the platform independent model (PIM) and the platform specific model (PSM).

---

[2] Part played by customer or his representative or failing by a member of the team.

[3] See Chapter 1, "The Model Driven Architecture Approach: A Framework for Developing Complex Agricultural Systems."

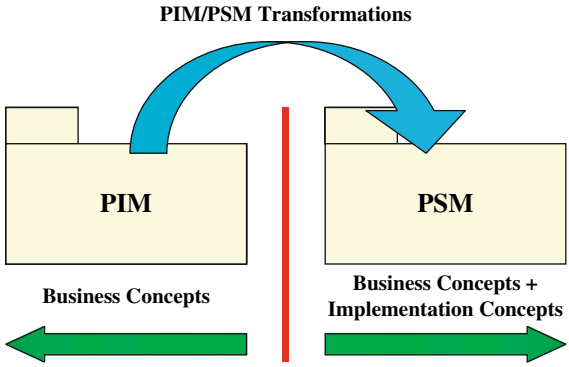


Fig. 5 Illustration of the separation of concepts and of the transformation notion

PIMs are models providing a description of the structure and functions of the system independently of platform specifications. PSMs are models defining how structure and functions of a system are implemented on a specific platform.

In fact, the MDA approach introduces a separation between concepts and specifications needed to develop software. PIMs only contain business concepts, whereas PSMs contain implementation concepts. Because all the PIM business concepts are included in PSMs, a PIM can be seen as a modified subset of a PSM [7]. Therefore, a PSM always derives from a model PIM through one or more transformations [18, 19].

Figure 5 illustrates this separation and transformation. If different platforms are used for the implementations (e.g., same standardized model implemented into different organizations), then more than one PSM may be derived from the same PIM.

The previous transformations, called PIM/PSM transformations, are not the only ones. In fact, the authors of MDA mention on the one hand the existence of PSM/PIM transformations converting a PSM into a PIM and, on the other hand, transformations whose model sources and targets are of the same standard (PIM/PIM transformations or PSM/PSM transformations).

In the process of development, PSM is not the last step as it is then necessary to project this model into a programming language. This projection is often considered as a transformation.

4 The New Interactive Development Method

4.1 The Principle of the Continuous Integration Unified Process Method

For about 40 years, the major aim of research bearing on the methods of development of computer applications has been to reduce the gap between the needs of the actors (users, clients, stakeholders, etc.) and the end product. To achieve this, the authors of the methods of development seek to associate and to

involve more and more the actors, who are the only ones that have a good knowledge of the studied system.

In the waterfall life-cycle, the actors act in the analysis phase at the start of the project, before the development team carries out the application, theoretically without any other participation of the actors. Practically, the actors mostly act when the project is of a significant size, but their interventions are not formalized.

In the unified process method, the iterative cycle requires organization of periodic meetings among the actors of the domain occurring at the beginning of each iteration, in the analysis phase, and at the end of the iteration to validate the iteration product.

The practice of customer on-site of the extreme programming method leads to the hosting of a representative of the actor within the development team. Within this framework, the actor is at the center of the development.

Actually, the increased participation of the actors enables, on the one hand, improvement in the capture of knowledge and the expression of the actors' needs and, on the other hand, to have, at a more or less continuous frequency, the validation of the evolution of the application. With this type of process, the semantic side of the application is of a higher quality. The direct consequence is that the increment developed during the iteration is more stable.

Building on that report, the authors have designed a new method called the *continuous integration unified process,* which allows an increase in the interactivity between the actors and the designer.

This new method is an extension of the unified process method incorporating some practices of the extreme programming method. It is based on the following report: in the analysis phase, the actors are in a situation similar to that of the customer on-site in the extreme programming method (see Section 2) – they are at the heart of the analysis. As communication is the key value of the extreme programming method, any technique or method increasing it will result in improvement of the quality of the end application. Dialogue around a prototype is one of these techniques or methods.

It is not rare that during the development of an application, one or several prototypes are produced so that the actors have a better understanding of what the end application will be. Then, the actors implicitly validate the concepts of the field and, if it is a "dynamic prototype" [24], they validate the assumed functionalities corresponding with their requirements. A prototype is a device that fluidizes the exchanges between the actors and the designer, but it also increases the area of shared knowledge [10] called *commonness* [27]. Moreover, the implementation of the prototype accelerates learning by the actors of the modeling language used by the designers [10].

The qualities of the prototype have led the authors to formalize its use in the analysis phase, a key phase for the capture of the knowledge and the actors' requirements.

To generate a prototype requires similar development to that of the final application. In this background, the development process includes simplified analysis, design, and implementation. If all the activities to develop the

prototype are done manually, the analysis will be interrupted by nonproductive slack periods that will prove to be expensive.

This exercise of analysis will quickly become tedious for the actors, and they will become demobilized and lose interest in the exercise. The result is that the analysis could be less relevant and the quality of the application could deteriorate. On the other hand, if the same activities are automated, then the slack periods do not exist, and the response of the actors to the prototype will be better than in front of a model for which all the finer points of the modeling language are not known. Then, the development process of the prototype is made according to a cycle with a very short duration, which is qualified as *rapid prototyping*.

Building on these thoughts, the definition of the new method is the following: the continuous integration unified process method superimposes, on the main cycle of the unified process method, a cycle of rapid prototyping (Fig. 6), which is provided with a process automating the evolution of the models from the analysis to the implementation.

The idea of automatic evolution of the models from the analysis up to the implementation can also be found in the concerns of the MDA community. It is obvious, from reading the fundamental texts of this approach [18, 19], that this was one of the objectives that were sought. Some authors [13, 28] describe as full MDA the complete automation of the evolution of the models. The *common warehouse metamodel* (CWM) standard [22] has been created to cover the complete cycle of design, completion, and management of the data warehouses [18].

Naturally, the challenge remains to design and implement a complete set of model transformations assuming a full MDA process. The authors reach such a

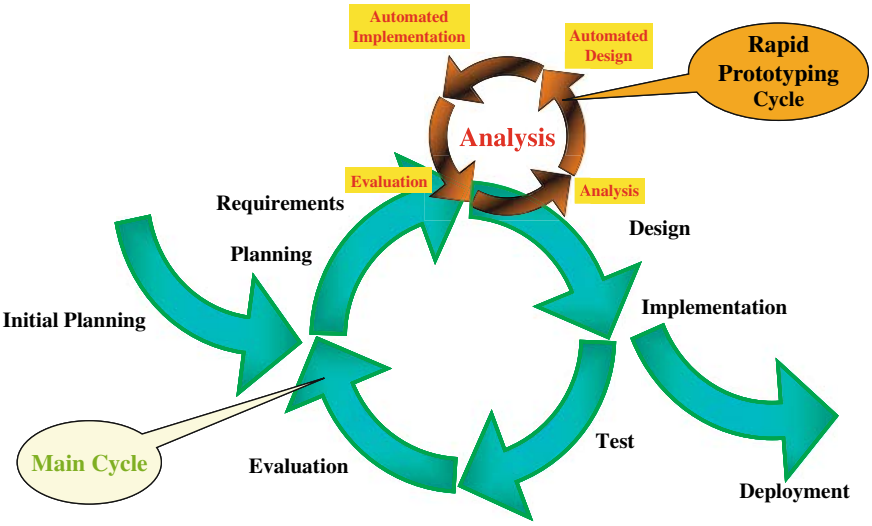


Fig. 6 Continuous integration unified process

challenge in generating automatically the structure of the database of a Geographic Information System (GIS).

Given such a set, the duration of the phases of design and implementation of the cycle of rapid prototyping is reduced to the unique time of completion of these transformations, time linked to the volume of concepts contained in the models.

4.2 The Software Development Process Approach: A Generalization of the MDA Approach

When an application is developed, one of the main preoccupations for a project manager and for the company in charge of the software development is the capitalization of knowledge and the re-use of the knowledge accumulated during development.

The capitalization of knowledge is not just the problem of separating the business concepts and implementation concepts according to the MDA vision presented in Section 3, as at each phase of the development, the type of mobilized knowledge is different. Thus, another approach involves capitalizing on the knowledge at each phase of the application development process. The *software development process approach* proposed by the authors is founded on this report [20]. Thus, in this new approach, a model is associated at each one of the phases.

In fact, this approach generalizes the MDA approach by refining the PIMs into three types of models: the analysis model, the preliminary design model, and the advanced design model. The first one is used to analyze the system with the actors, the second one is dedicated to the concepts coming from a domain in relation to the studied domain (point, line, or polygon from geomatic domain, for example), and the third is specialized for the description of the computer concepts or models independent of the programming language (ASCII files, host, for example). Although the refinement of PSMs is theoretically possible, we have not worked on this subject for the moment.

4.3 The Software Development Process Model: A Modeling Artifact for Knowledge Capitalization

To apply the software development process approach, archiving models after each phase is a solution that is often used, even if it is not formalized. Often, these archives are also physically independent, so any change, including, for example, changing the name of a concept, quickly becomes attempting the impossible, as it is difficult and expensive to pass along the change to all the archived models. Quickly, the models diverge and their coherence deteriorates.

Software development process model (SPDM) is an artifact that enables one to keep the models associated with the phases of development coherent. It has been conceived and implemented by Miralles [20] into case-tools. This modeling

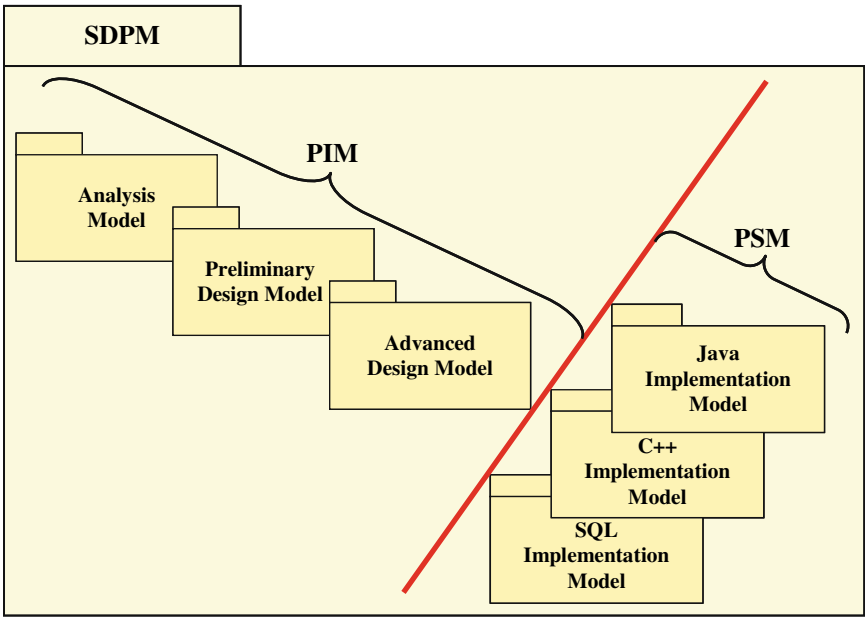


Fig. 7 The software development process model

artifact contains the different models associated with the phases of the software development process. In the vision of the authors, the software development process model is the MODEL of the application under development.

Figure 7 shows the software development process model for the development of software following the *two-track unified process* method [25], a method derived from the unified process method. This figure also shows that the PIM/PSM separation introduced by the MDA approach occurs when the project moves from the advanced design phase to the implementation phase.

4.4 The Complete Set of Transformations Enabling a Full MDA Process for Databases

To realize a full MDA[4] process for GIS, the first set of transformations implemented into the case-tool is in charge of diffusing the captured business concepts from the analysis model to the implementation models (see Section 4.4.1).

[4] Normally, a full MDA process must include the model verification and the model compilation. Currently, the model verification is not made but it is one of the future subjects of research. The model compilation is held by code generators (C++ and C#, Java, Corba, and SQL) proposed by case-tool.

To describe the spatial properties (point, line, and polygon) and temporal properties (instant and period) in the analysis model, the authors adopted the pictogrammic language of Perceptory [2]. These pictograms are introduced into the business concept via stereotypes (UML concepts with which it is possible to associate a pictogram). In the analysis model, the stereotype/pictogram couple only has an informative value. The second type of transformation developed reifies the stereotype/pictogram couple into UML modeling elements (see Section 4.4.2).

Finally, the last type of transformation developed is in charge of adapting the SQL implementation model after cloning the SQL code generator of the case-tool (see Section 4.4.3).

4.4.1 Diffusion Transformation and Management of the Software Development Process Model

This transformation clones a concept from a source model into the next model. Step by step, the concepts captured in the analysis phase and added into the analysis model are transferred to the implementation models.

To guarantee the consistency of the software development process model, a *cloning traceability architecture* is automatically built by the *diffusion transformation*. After cloning, this transformation establishes an individual cloning traceability link between each one of the source concepts and the cloned concepts. Figure 8 illustrates the cloning traceability architecture.

In an iterative development process, the *diffusion transformation* adds, with every iteration, a new clone of the same source into the following model. To avoid this problem, when an individual cloning traceability link exists, the *diffusion transformation* does not clone the concepts but only carries out one update of the clone.

4.4.2 The GISTransformations

The GIS Design Pattern Generation Transformation

The spatial and temporal concepts have stable relationships that are completely known. They constitute recurrent minimodels having the main property of design patterns[5]: recurrence [9]. It is this property that led authors to call these minimodels *design patterns*. These GIS design patterns do not have the

[5] A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context [9].

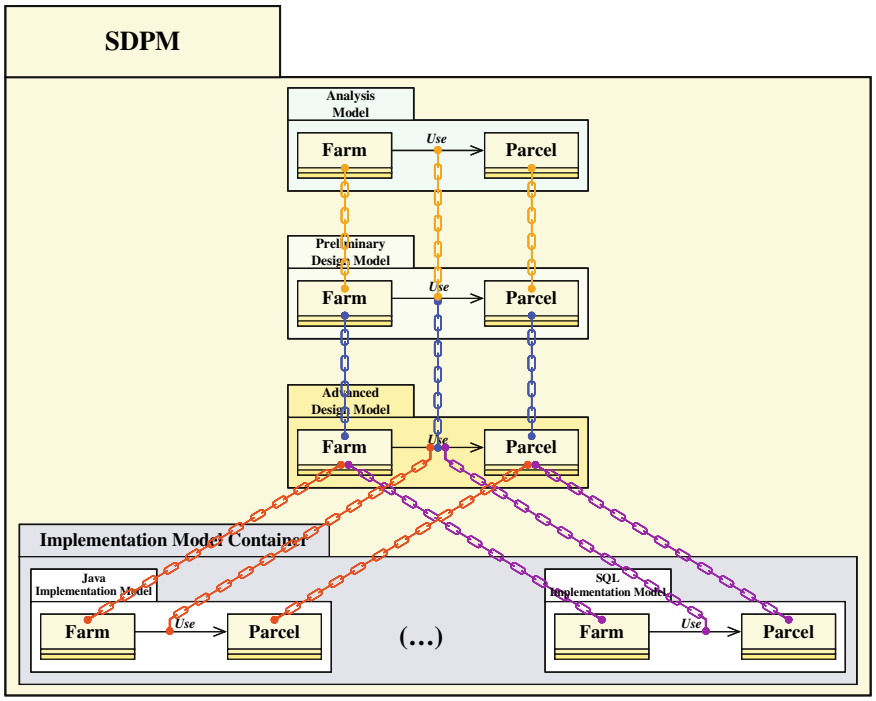


Fig. 8 Example of the cloning traceability architecture

same statutes as the design patterns described in Ref. 9, but they are fundamental design patterns in the geomatic domain. Figure 9 shows an example of design pattern of the GIS domain. The set of these patterns is called the *GIS design pattern*.

Given that the design patterns are always identical, they can be automatically generated with a case-tool without any difficulty. The *GIS design pattern generation transformation* is the transformation in charge of generating the set of GIS design patterns.

The Pictogram Translation Transformation

Once the GIS design patterns have been created, the business and the spatial or temporal concepts represented by the pictogram are totally disassociated (Fig. 10, "Before"). The goal of the *pictogram translation transformation* $\boldsymbol{T_p}$

Fig. 9 Example of a GIS design pattern

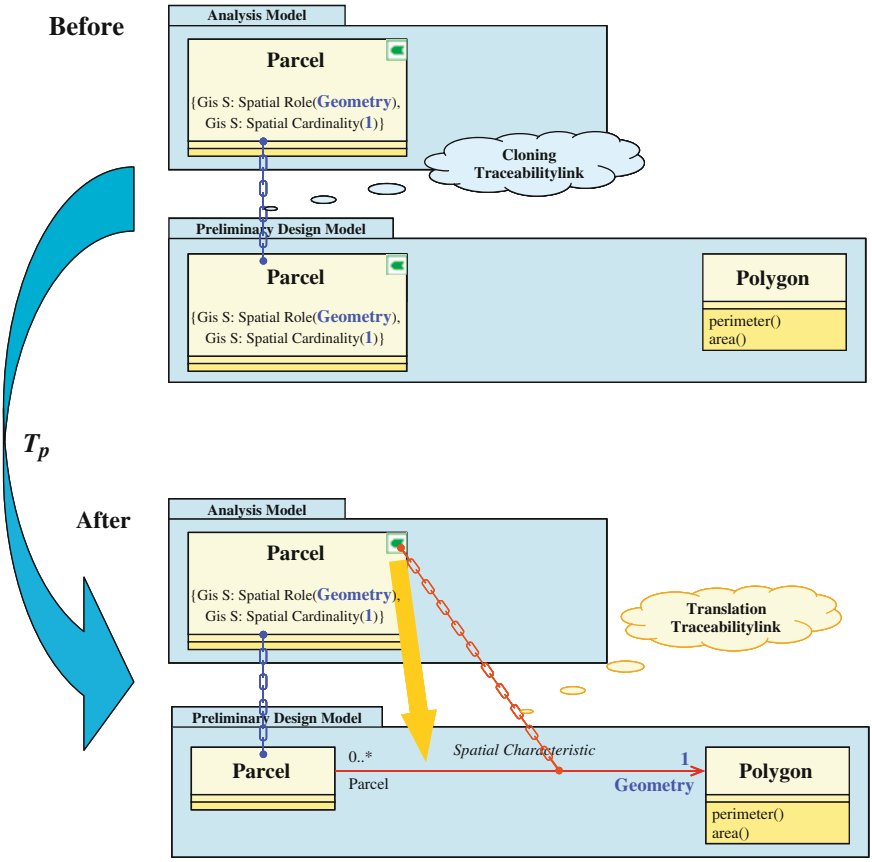


Fig. 10 The pictogram translation transformation T_P

(Fig. 10) is to automatically establish a relationship between the *Parcel* and *Polygon* concepts. This transformation creates an association, called *spatial characteristic*.

During the capture of the pictogram, two tagged values are added to the business concept to specify the role of the spatial concept ({*Gis S: Spatial Role(Geometry)*}) and its cardinality ({*Gis S: Spatial Cardinality(1)*}). By default, this role and this cardinality have the values *Geometry* and *1*, respectively, but the designer can subsequently modify them. In this association, the entity name has been allocated to its role, *Parcel* in this example, and its cardinality value is 0.1. Once the association has been created, the stereotype/pictogram and the two tagged values are deleted because this information becomes redundant with the association.

To ensure traceability, the transformation T_p creates a traceability link, called *translation traceability link*, between the pictogram of the business entity of the analysis model and the *spatial characteristic* association.

4.4.3 The SQL Transformation

To achieve a full MDA process, the SQL transformation T_{SQL} has been conceived and implemented. It is applied on the SQL implementation model. The objective of this transformation is to adapt the SQL implementation model after cloning (Fig. 11, "Before") to the SQL code generator of the case-tool. To do this, it adds SQL concepts, such as persistence and key primary (Fig. 11, "After"), to the business concepts. These SQL concepts are not systematically added to all the business concepts but only to a certain number of them.

Persistence is an "SQL" property that should be added to all concepts that should be converted into tables (Fig. 11, "After"). Considering that the "son" concepts involved in a hierarchy of business concepts inherit properties of "father" concepts, the persistence property should be put in the "root" concept of the hierarchy.

Although the primary key properties are as essential as the concepts involved in a relationship of association, of aggregation, or of composition, the transformation T_{SQL} systematically adds the primary key property on all the persistent classes (Fig. 11, "After"). Just like for persistence, the "son" concepts of a hierarchy of concepts inherit the primary key of the "root" concept.

Annotated with SQL concepts, the SQL code generator can be applied on the SQL implementation model to produce the SQL code for creating the database.

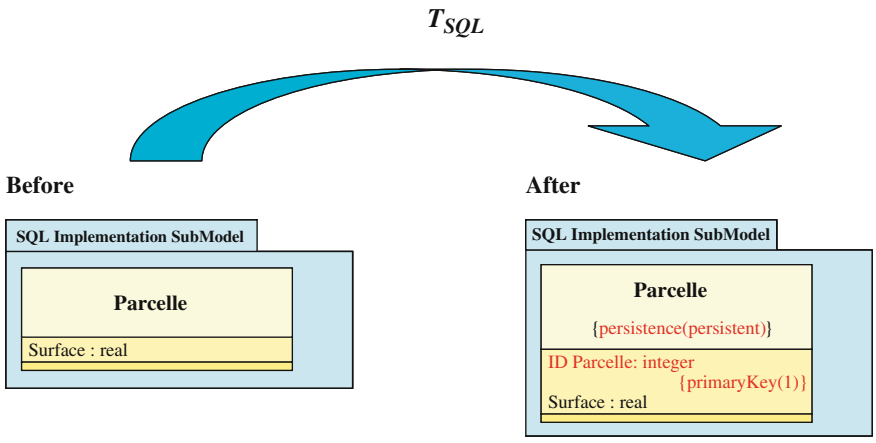


Fig. 11 The SQL transformation T_{SQL}

5 Conclusions

The continuous integration unified process method proposed above presents a number of advantages. Below, we clarify three important advantages.

The first advantage is to have split research of excellence into the subcategories *semantic excellence* and *technical excellence*. This dichotomy introduced by the continuous integration unified process method has been obtained by superimposing a cycle of rapid prototyping in the analysis phase onto the main cycle (Fig. 6). Successive prototypes completed during this cycle of rapid prototyping are devices that form the topics of discussion and criticisms from the actors. Progressively, these prototypes tend toward "the ideal application" and the actors of the field need this. To reach this ideal state, the actors describe the business concepts, and their description becomes finer as one proceeds through the iterations of the cycle of rapid prototyping. The semantic excellence is then reached, and the model coming from the phase of analysis in the main cycle can be stabilized from the semantic point of view. In the phase of design and implementation of the main cycle, the development team has the leisure to address and to solve the technical sides linked to the production of the binary code. During this main cycle, the aim is toward technical excellence.

The second advantage is the capitalization of knowledge. The generalization of the idea to separate the business concepts from those of implementation, an idea suggested by the MDA approach, has led to the design of the software development process approach, which associates a model with each of the phases of the development cycle of an application. Thus, the software development process model, an artifact that reifies the software development process approach, groups together all the models that are associated with the development phases. Thus, the project manager has both a global view of development through the software development process model and a detailed view through each model that makes up the software development process model. It can, at any time, identify at what phase of development a concept has been introduced (business, from an associated domain, implementation, etc.) by scanning the content of the models. These models are actually "capitalization planes" of knowledge implemented at each phase needed to produce the application.

The third advantage is a gain in quality linked to automation of the evolution by the model transformations. During the development of an application, some actions or activities are conducted in a repetitive way tens, and even hundreds, of times. For instance, one should add the persistence and the primary key properties for all the concepts except those involved in a hierarchy (see Section 4.4.3). If this activity is done manually, more time will be needed than if it is done by a transformation that has been designed and implemented in the case-tools. Moreover, it is not rare that, when done manually, some of the concepts are forgotten and that these mistakes are noticed during the creation of the database. In this case, all the implementation process should be resumed. A designer, who may be poorly experienced in SQL language, may also add this

information to all the concepts of a hierarchy even though this is not necessary. In this case, the coherence of the model deteriorates. In the software development process approach, nothing forbids the designer to add these SQL properties in the design or analysis phase. The capitalization is then affected. The implementation of the transformation T_{SQL} on the SQL implementation model avoids this problem. The quality of the SQL implementation model is better and the productivity is increased. It is the same for all the other transformations described in Section 4.4. These thoughts are corroborated by the study conducted by The Middleware Company [29]. This consulting business has been in charge of conducting a productivity analysis between two teams with an equivalent competency level: the first one was to develop an application in a traditional way, and the second one had to create the same application according to the MDA approach. The team working according to the MDA approach completed the application with a time savings of 35% and a gain in quality as the team did not have to correct development bugs. The analysts of the consulting business attribute this gain in quality to the automated transformations of the models.

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. 2001. Méthodes agiles (1) – Panorama. Développeur Référence. http://www.devreference.net/devrefv205.pdf. Last access: September 2004.
4. Bénard J-L, Bossavit L, Médina R, Williams D. 2002. Gestion de projet eXtreme Programming. Eyrolles. 298 pp.
5. Booch G, Rumbaugh J, Jacobson I. 2000. Guide de l'utilisateur UML. Eyrolles. 500 pp.
6. Cros T. 2001. La conception dans l'eXtreme Programming. Développeur Référence. http://www.devreference.net/devrefv201.pdf. Last access: September 2004.
7. Desfray P. 1994. Object Engineering – The Fourth Dimension. Addison-Wesley. 342 pp.
8. Fayet E. 2002. Forum Utilisateurs Rational – Le discours de la méthode. Développeur Référence. http://www.devreference.net/devrefv220.pdf. Last access: September 2004.
9. Gamma E, Helm R, Johnson R, Vlissides J. 2001. Design patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. 416 pp.
10. Guimond L-E. 2005. Conception d'un environnement de découverte des besoins pour le développement de solutions SOLAP. Thèse. Université Laval, Québec. 124 pp.
11. Jacobson I. 2003. Use Cases – Yesterday, Today, and Tomorrow. http://www.ivarjacobson.com/html/content/publications_papers.html; http://www.ivarjacobson.com/publications/uc/UseCases_TheRationalEdge_Mar2003.pdf. Last access: August 2005.
12. Jacobson I, Booch G, Rumbaugh J. 1999. The Unified Software Development Process. Addison-Wesley. 463 pp.
13. Kleppe A. 2004. Interview with Anneke Kleppe. Code Generation Network. http://www.codegeneration.net/tiki-read_article.php articleId=21. Last access: August 2006.

14. Kleppe A, Warmer J, Bast W. 2003. MDA Explained: The Model Driven Architecture—Practice and Promise. Addison-Wesley Professional. 170 pp.
15. Kruchten PB. 1999. The Rational Unified Process: An Introduction. Addison-Wesley Professional. 336 pp.
16. Larman C. 2002. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice Hall PTR. 627 pp.
17. Larman C. 2002. UML et les Design Patterns. CampusPress. 672 pp.
18. Miller J, Mukerji J. 2001. Model Driven Architecture (MDA). OMG. http://www.omg.org/cgi-bin/apps/doc? 07-01.pdf. Last access: September 2004.
19. Miller J, Mukerji J. 2003. MDA Guide Version 1.0.1. OMG. http://www.omg.org/cgi-bin/doc? -01. Last access: May 2006.
20. 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.
21. Muller P-A, Gaertner N. 2000. Modélisation objet avec UML. Eyrolles. 520 pp.
22. OMG. 2001. Common Warehouse Metamodel – Version 1.0. OMG. http://www.omg.org/cgi-bin/doc ?ad/2001-02-01. Last access: June 2004.
23. OMG. 2003. Unified Modeling Language – Specification – Version 1.5. http://www.omg.org/cgi-bin/apps/doc? formal/03-03-01.pdf. 736 pp.
24. Région Wallonne. 2004. Le prototypage: Définition et objectifs. Portail Wallonie.
25. Roques P, Vallée F. 2002. UML en Action – De l'analyse des besoins à la conception en Java. Eyrolles. 388 pp.
26. Royce WW. 1970. Managing the Development of Large Software Systems. Presented atIEEE Westcon, Monterey, CA.
27. Schramm WL. 1954. How communication works. In: The Process and Effects of Communication. University of Illinois Press. pp. 3–26.
28. Softeam. 2005. Formation sur les Modèles Objet et UML.
29. The Middleware Company. 2003. Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach – Productivity Analysis.

AQ6
AQ7

A New Methodology to Automate the Transformation of GISGIS Models in an Iterativeiterative Development Process

Query No.	Line No.	Query
AQ1	109	Clarify sense/ rewrite phrase: “and only 28% of the developments have to finish in delay and in the projected budget”; are 28% delayed? do they finish within budget?
AQ2	134	Define acronym SCRUM.
AQ3	150	Check legend of Fig. 3; identical to that of Fig. 2.
AQ4	235	In footnote 2, clarify phrase: “or failing by a member of the team”; how does this relate to term customer on-site?
AQ5	505	Define acronym SQL.
AQ6	782	In Ref. 28, provide name of publisher.
AQ7	783	In Ref. 29, provide name of publisher.