



HAL
open science

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

André Miralles, Thérèse Libourel Rouge

► **To cite this version:**

André Miralles, Thérèse Libourel Rouge. A new methodology to automate the transformation of GIS models in an iterative development process. *Advances in Modelling Agricultural Systems*, 25, Springer, pp.19-36, 2009, Springer Optimization and Its Applications, 978-0-387-75180-1. lirmm-00351444

HAL Id: lirmm-00351444

<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00351444v1>

Submitted on 9 Jan 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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 1970s, 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

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*:

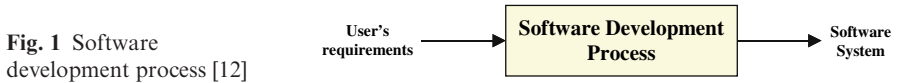


Fig. 1 Software development process [12]

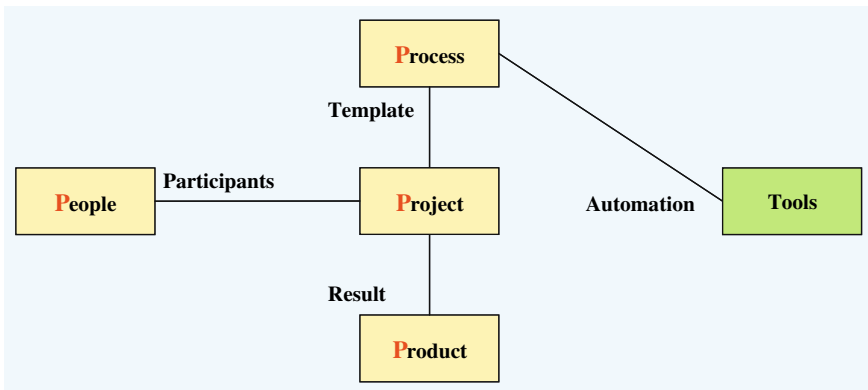


Fig. 2 The 4 Ps software development process [12]

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

103 These two metaphors perfectly illustrate the challenge with which the project
104 leader and the programmers are confronted when they take on the realization of a
105 data-processing application. This challenge is not entirely imaginary. The statistics
106 of Ref. 30 give an idea of the difficulty. **According to these statistics, the failure risk
107 of the development of an application is 23%, the risk of drift is 49%, and only 28%
108 of the developments have to finish in delay and in the projected budget.** These
109 figures are from an investigation carried out on more than 150,000 developments
110 achieved in the United States in 2000. It is noteworthy that in this investigation, the
111 developments aimed at creating a new application are grouped with those aimed at
112 the evolution of an existing application. Thus, it is quite likely that the figure of
113 28% is overestimated, as the failure risk linked to achieving a new application is
114 much larger than the risk linked to the evolution of an application.

117 2 The Software Development Process

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

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

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

AQ1

AQ2

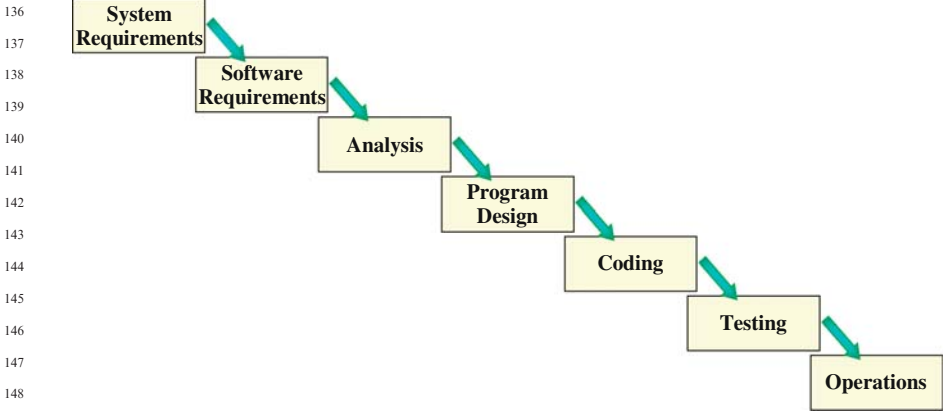


Fig. 3 The 4 Ps software development process [26]

The software development processes that were used in the 1970s 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 fields¹ [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.

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

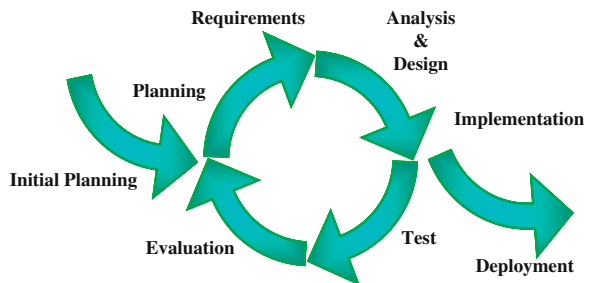
190 The *iterative* development process (Fig. 4) has been designed to prevent the
191 drawbacks caused by linear development. In order to do this, the system is
192 structured into subsystems, and for each iteration, a subsystem is analyzed and
193 implemented. Therefore, the model evolves following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

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

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

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

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



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

- 226 • **Feedback**, which should be quick from the stakeholders and from the other
227 members of the development team, enables the developer to have informa-
228 tion on the quality of his development.
- 229 • **Courage** to tell things as they are and to make difficult decisions like chang-
230 ing a code structure or throwing it away [6].

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

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

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

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

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

254 3 The Model Driven Architecture³

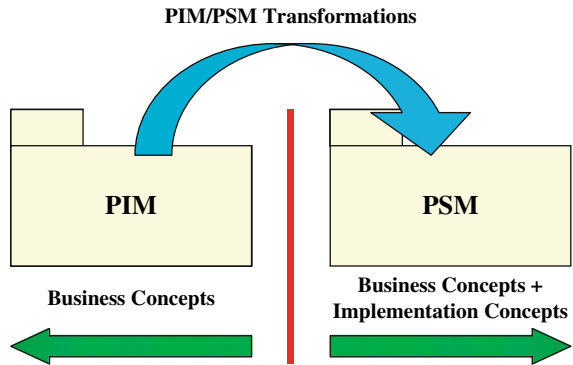
256 Model driven architecture (MDA) is a software design approach proposed by
257 the Object Management Group (OMG) with the objective of improving applica-
258 tion developments. It was conceived and formalized in 2001 to improve produc-
259 tivity but also to resolve problems of software portability, software integration,
260 and software interoperability encountered during developments [14].

261 To achieve this objective, the MDA approach recommends that designers
262 separate the specification of system functionality from the specification of the
263 implementation of that functionality on a specific technology platform [18]. For
264 that, the authors of this approach suggest use of two types of models: the
265 platform independent model (PIM) and the platform specific model (PSM).
266

267
268 ² Part played by customer or his representative or failing by a member of the team.

269 ³ See Chapter 1, “The Model Driven Architecture Approach: A Framework for Developing
270 Complex Agricultural Systems.”

271 **Fig. 5** Illustration of the
 272 separation of concepts and
 273 of the transformation notion



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

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

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

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

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

306 4 The New Interactive Development Method

308 4.1 *The Principle of the Continuous Integration Unified* 309 *Process Method*

310
 311
 312 For about 40 years, the major aim of research bearing on the methods of
 313 development of computer applications has been to reduce the gap between the
 314 needs of the actors (users, clients, stakeholders, etc.) and the end product. To
 315 achieve this, the authors of the methods of development seek to associate and to

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

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

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

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

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

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

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

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

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

358 To generate a prototype requires similar development to that of the final
359 application. In this background, the development process includes simplified
360 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

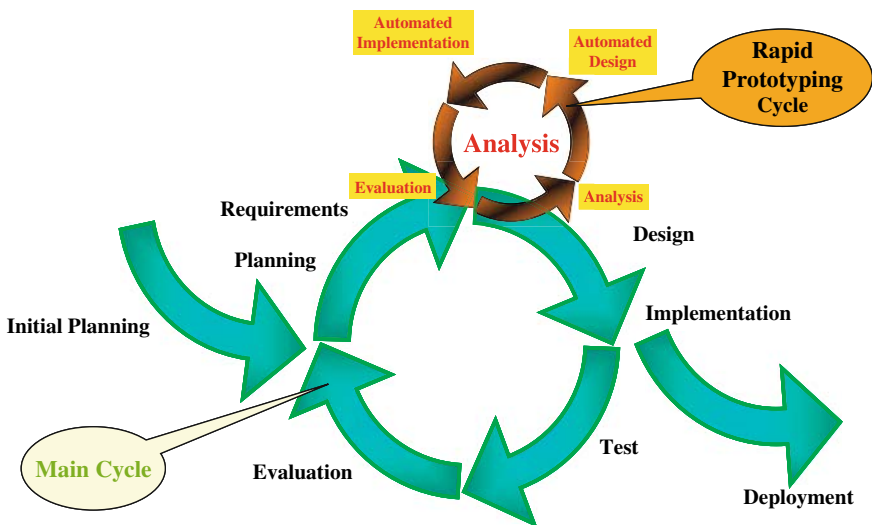


Fig. 6 Continuous integration unified process

406 challenge in generating automatically the structure of the database of a Geo-
407 graphic Information System (GIS).

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

411 412 413 ***4.2 The Software Development Process Approach: 414 A Generalization of the MDA Approach*** 415

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

420 The capitalization of knowledge is not just the problem of separating the
421 business concepts and implementation concepts according to the MDA vision
422 presented in Section 3, as at each phase of the development, the type of mobilized
423 knowledge is different. Thus, another approach involves capitalizing on the
424 knowledge at each phase of the application development process. The *software*
425 *development process approach* proposed by the authors is founded on this report
426 [20]. Thus, in this new approach, a model is associated at each one of the phases.

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

436 437 438 ***4.3 The Software Development Process Model: A Modeling 439 Artifact for Knowledge Capitalization*** 440 441

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

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

451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470

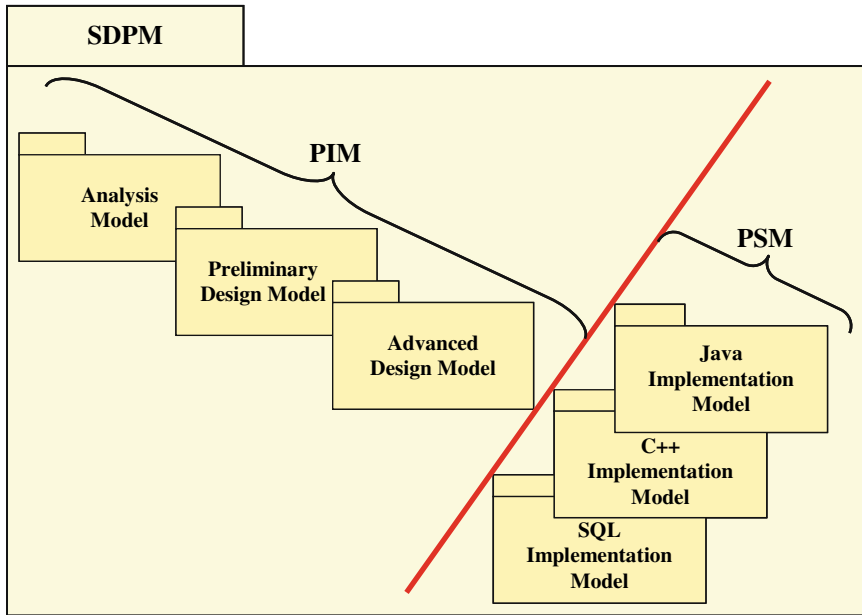


Fig. 7 The software development process model

471
472
473
474
475
476
477
478
479
480
481

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.

482
483

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

487
488
489
490

To realize a full MDA⁴ 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).

491
492
493
494
495

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.

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

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

509 4.4.1 Diffusion Transformation and Management of the Software 510 Development Process Model

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

514 To guarantee the consistency of the software development process model, a
 515 *cloning traceability architecture* is automatically built by the *diffusion transfor-*
 516 *mation*. After cloning, this transformation establishes an individual cloning
 517 traceability link between each one of the source concepts and the cloned con-
 518 cepts. Figure 8 illustrates the cloning traceability architecture.

519 In an iterative development process, the *diffusion transformation* adds, with
 520 every iteration, a new clone of the same source into the following model. To
 521 avoid this problem, when an individual cloning traceability link exists, the
 522 *diffusion transformation* does not clone the concepts but only carries out one
 523 update of the clone.
 524
 525
 526

527 4.4.2 The GIS Transformations

529 The GIS Design Pattern Generation Transformation

530 The spatial and temporal concepts have stable relationships that are completely
 531 known. They constitute recurrent minimodels having the main property of
 532 design patterns⁵: recurrence [9]. It is this property that led authors to call
 533 these minimodels *design patterns*. These GIS design patterns do not have the
 534

535
 536 ⁵ A design pattern systematically names, motivates, and explains a general design that
 537 addresses a recurring design problem in object-oriented systems. It describes the problem,
 538 the solution, when to apply the solution, and its consequences. It also gives implementation
 539 hints and examples. The solution is a general arrangement of objects and classes that solve the
 540 problem. The solution is customized and implemented to solve the problem in a particular
 context [9].

541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562

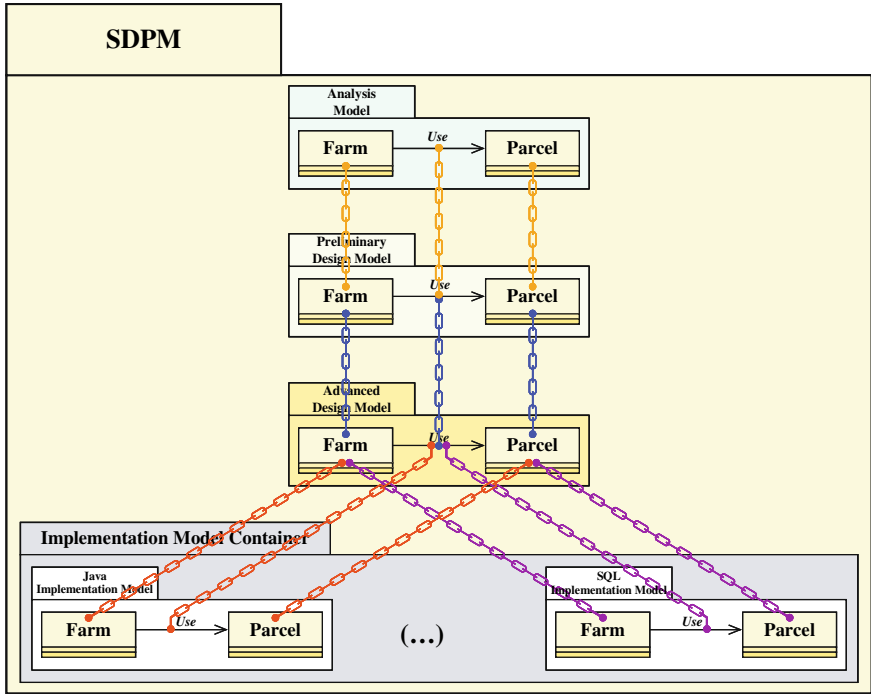


Fig. 8 Example of the cloning traceability architecture

566
567
568
569

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*.

570
571
572
573

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.

574
575

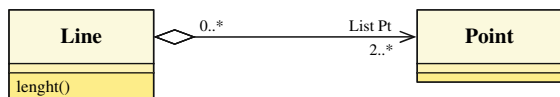
The Pictogram Translation Transformation

576
577
578
579
580

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* T_p

581
582
583
584
585

Fig. 9 Example of a GIS design pattern



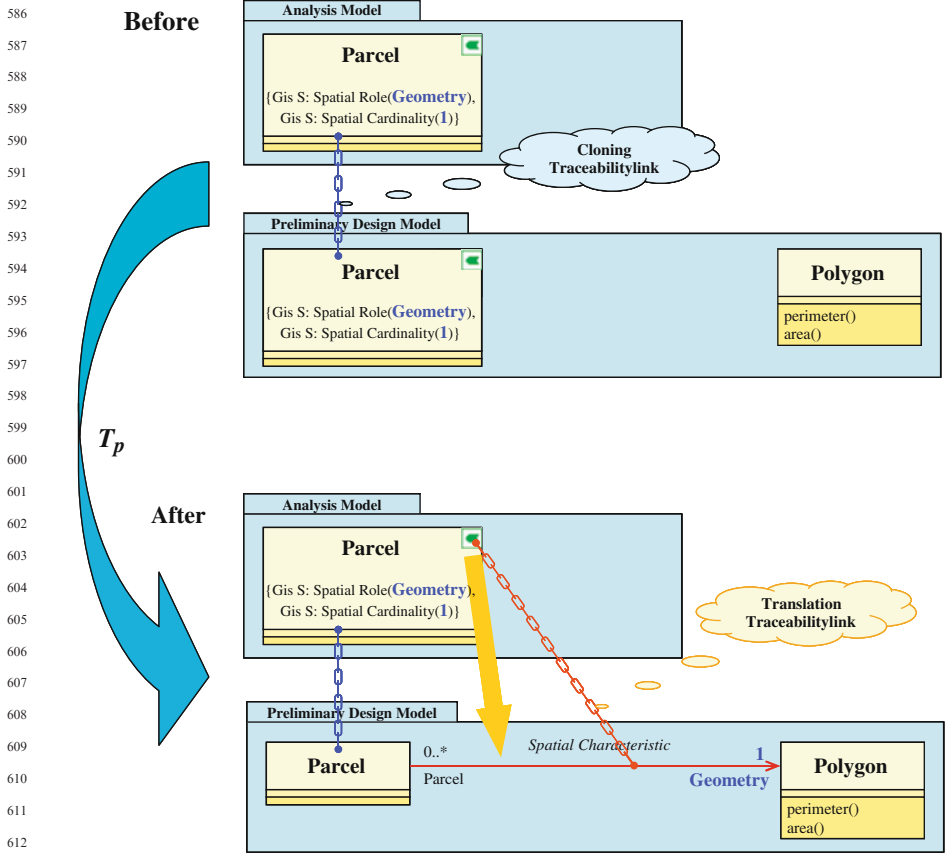


Fig. 10 The pictogram translation transformation T_p

(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.

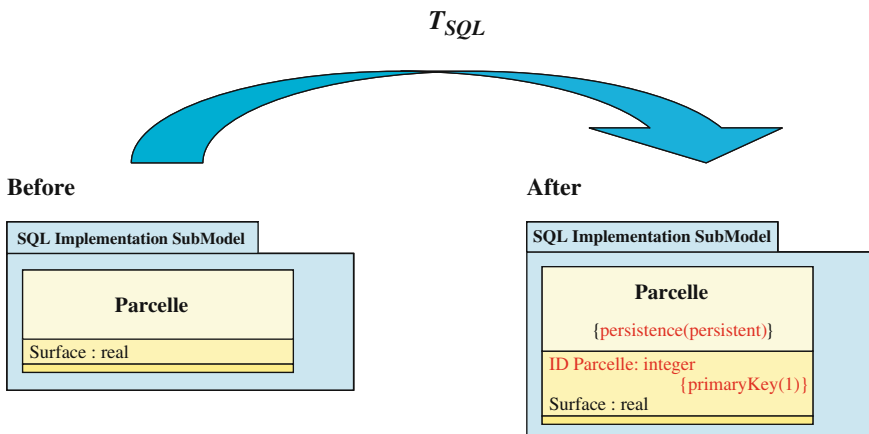


Fig. 11 The SQL transformation T_{SQL}

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

721 information to all the concepts of a hierarchy even though this is not necessary.
 722 In this case, the coherence of the model deteriorates. In the software develop-
 723 ment process approach, nothing forbids the designer to add these SQL proper-
 724 ties in the design or analysis phase. The capitalization is then affected. The
 725 implementation of the transformation T_{SQL} on the SQL implementation model
 726 avoids this problem. The quality of the SQL implementation model is better and
 727 the productivity is increased. It is the same for all the other transformations
 728 described in Section 4.4. These thoughts are corroborated by the study con-
 729 ducted by The Middleware Company [29]. This consulting business has been
 730 in charge of conducting a productivity analysis between two teams with
 731 an equivalent competency level: the first one was to develop an application
 732 in a traditional way, and the second one had to create the same application
 733 according to the MDA approach. The team working according to the MDA
 734 approach completed the application with a time savings of 35% and a gain in
 735 quality as the team did not have to correct development bugs. The analysts of
 736 the consulting business attribute this gain in quality to the automated transfor-
 737 mations of the models.
 738
 739
 740

741 References

- 742 1. Beck K. 2000. eXtreme Programming Explained – Embrace Change. Addison-Wesley.
 743 190 pp.
- 744 2. Bédard Y, Larrivée S, Proulx M-J, Nadeau M. 2004. Modeling Geospatial Databases
 745 with Plug-ins for Visual Languages: A Pragmatic Approach and the Impacts of 16 Years
 746 of Research and Experimentations on Perceptory. Presented at ER Workshops 2004
 747 CoMoGIS, Shanghai, China.
- 748 3. Bénard J-L. 2001. Méthodes agiles (1) – Panorama. Développeur Référence. [http://www.
 749 devreference.net/devrefv205.pdf](http://www.devreference.net/devrefv205.pdf). Last access: September 2004.
- 750 4. Bénard J-L, Bossavit L, Médina R, Williams D. 2002. Gestion de projet eXtreme
 751 Programming. Eyrolles. 298 pp.
- 752 5. Booch G, Rumbaugh J, Jacobson I. 2000. Guide de l'utilisateur UML. Eyrolles. 500 pp.
- 753 6. Cros T. 2001. La conception dans l'eXtreme Programming. Développeur Référence.
<http://www.devreference.net/devrefv201.pdf>. Last access: September 2004.
- 754 7. Desfray P. 1994. Object Engineering – The Fourth Dimension. Addison-Wesley. 342 pp.
- 755 8. Fayet E. 2002. Forum Utilisateurs Rational – Le discours de la méthode. Développeur
 756 Référence. <http://www.devreference.net/devrefv220.pdf>. Last access: September 2004.
- 757 9. Gamma E, Helm R, Johnson R, Vlissides J. 2001. Design patterns – Elements of Reusable
 Object-Oriented Software. Addison-Wesley Professional. 416 pp.
- 758 10. Guimond L-E. 2005. Conception d'un environnement de découverte des besoins pour le
 759 développement de solutions SOLAP. Thèse. Université Laval, Québec. 124 pp.
- 760 11. Jacobson I. 2003. Use Cases – Yesterday, Today, and Tomorrow. [http://www.ivarjacob
 761 son.com/html/content/publications_papers.html](http://www.ivarjacobson.com/html/content/publications_papers.html); [http://www.ivarjacob
 762 son.com/publications/uc/UseCases_TheRationalEdge_Mar2003.pdf](http://www.ivarjacobson.com/publications/uc/UseCases_TheRationalEdge_Mar2003.pdf). Last access: August 2005.
- 763 12. Jacobson I, Booch G, Rumbaugh J. 1999. The Unified Software Development Process.
 Addison-Wesley. 463 pp.
- 764 13. Kleppe A. 2004. Interview with Anneke Kleppe. Code Generation Network. [http://www.
 765 codegeneration.net/tiki-read_article.php articleId=21](http://www.codegeneration.net/tiki-read_article.php%20articleId=21). Last access: August 2006.

- 766 14. Kleppe A, Warmer J, Bast W. 2003. MDA Explained: The Model Driven Architecture—
767 Practice and Promise. Addison-Wesley Professional. 170 pp.
- 768 15. Kruchten PB. 1999. The Rational Unified Process: An Introduction. Addison-Wesley
769 Professional. 336 pp.
- 770 16. Larman C. 2002. Applying UML and Patterns: An Introduction to Object-Oriented
771 Analysis and Design and the Unified Process. Prentice Hall PTR. 627 pp.
- 772 17. Larman C. 2002. UML et les Design Patterns. CampusPress. 672 pp.
- 773 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.
- 774 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.
- 775 20. Miralles A. 2006. Ingénierie des modèles pour les applications environnementales. Thèse
776 de doctorat. Université Montpellier II, Montpellier. <http://www.teledetection.fr/ingenierie-des-modeles-pour-les-applications-environnementales-3.html>. 322 pp.
- 777 21. Muller P-A, Gaertner N. 2000. Modélisation objet avec UML. Eyrolles. 520 pp.
- 778 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.
- 779 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.
- 780 24. Région Wallonne. 2004. Le prototypage: Définition et objectifs. Portail Wallonie.
- 781 25. Roques P, Vallée F. 2002. UML en Action – De l’analyse des besoins à la conception en
782 Java. Eyrolles. 388 pp.
- 783 26. Royce WW. 1970. Managing the Development of Large Software Systems. Presented
784 atIEEE Westcon, Monterey, CA.
- 785 27. Schramm WL. 1954. How communication works. In: The Process and Effects of Communication. University of Illinois Press. pp. 3–26.
- 786 28. Softeam. 2005. Formation sur les Modèles Objet et UML.
- 787 29. The Middleware Company. 2003. Model Driven Development for J2EE Utilizing a
788 Model Driven Architecture (MDA) Approach – Productivity Analysis.
- 789
- 790
- 791
- 792
- 793
- 794
- 795
- 796
- 797
- 798
- 799
- 800
- 801
- 802
- 803
- 804
- 805
- 806
- 807
- 808
- 809
- 810

AQ6

AQ7

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

814 Query No.	Line No.	Query
815 AQ1	109	816 Clarify sense/ rewrite phrase: “and only 28% of the 817 developments have to finish in delay and in the projected 818 budget”; are 28% delayed? do they finish within budget?
819 AQ2	134	820 Define acronym SCRUM.
821 AQ3	150	822 Check legend of Fig. 3; identical to that of Fig. 2.
823 AQ4	235	824 In footnote 2, clarify phrase: “or failing by a member of the 825 team”; how does this relate to term customer on-site?
826 AQ5	505	827 Define acronym SQL.
828 AQ6	782	829 In Ref. 28, provide name of publisher.
830 AQ7	783	831 In Ref. 29, provide name of publisher.

824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855