Towards a CMMI-Compliant Goal-Oriented Software Process through Model-Driven Development

Alexandre Vasconcelos, Giovanni Giachetti, Beatriz Marín, Oscar Pastor

Towards a CMMI-compliant Goal-Oriented Software Process through Model-Driven Development

Alexandre M. L. de Vasconcelos[1], Giovanni Giachetti[2], Beatriz Marín[2], and Oscar Pastor[2]

[1]Centro de Informática - Universidade Federal de Pernambuco
Av. Jornalista Anibal Fernandes s/n, 50740-560, Cidade Universitária, Recife-PE, Brazil
amlv@cin.ufpe.br
[2]Centro de Inv. Métodos de Producción de Software - Universidad Politécnica de Valencia
Camino de Vera s/n, 46022, Valencia, Spain
{ggiachetti, bmarin, opastor}@pros.upv.es

Abstract. The *i** framework is a Goal-Oriented Requirement Engineering (GORE) approach that is widely applied at academic level. However, its application to industrial scenarios is limited. For the application of *i** in concrete software development process, an alternative is to transform the defined requirements models into initial input models to be used by Model-Driven Development (MDD) approaches. However, this does not assure that the resultant development process will be sound enough to motivate real development companies to adopt this GORE solution. To tackle this issue, we propose the alignment of GORE and MDD solutions with software process maturity models, which are strongly adopted and applied by industry. In particular, we have considered an approach that integrates the *i** framework into an industrially-applied MDD solution to obtain a development process (that goes from requirements to the final software code), which is compliant with the CMMI-DEV maturity model.

Keywords: Goal-Oriented Requirements Engineering, *i** framework, Model-Driven Development, Software Process Quality, CMMI.

1 Introduction

Requirement modeling plays a relevant role in software development, since the quality of the requirements has a direct impact on the success of software development projects [10]. Among several approaches for defining requirements, the *Goal-Oriented Requirement Engineering* (GORE) [28][31][32] is one that has a wide application spectrum. In general terms, GORE focuses on obtaining the "why" of the intended systems through the analysis of organizational scenarios. It is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements.

However, as stated in [5], a Requirements Engineering (RE) approach is not useful per se, it must be appropriate for the software process into which it is integrated.

Thus, a GORE approach must be properly integrated into a full software process (from requirements to the final code). An alternative to achieve this integration is to automatically transform the requirements models into an initial model [1][8] to be used as input in the context of a Model-Driven Development (MDD) approach [26][33]. Then, this initial model can be refined to automatically generate code through a model compilation (transformation) process. An additional advantage of integrating GORE with MDD is that, through the automatic generation of code from models, MDD allows lower development costs, higher productivity, portability, interoperability, ease of software evolution, and software quality improvement [16].

Among the existing GORE approaches, the *i** framework [35] is one of the most widespread and used at research level [36]. However, there is a gap between the vast application of *i** in academy in relation to its application to real (industrial) development scenarios [34]. An alternative to obtain a suitable support for the application of the *i** framework into real scenarios is to align *i**-based development processes with a software process maturity model [22][30], such as the CMMI-DEV (Capability Maturity Model Integration for Development) [26]. In this way, this kind of GORE solutions become more attractive for the companies that are using those maturity models as the basis to improve their development processes in order to become more competitive in terms of quality and maturity of their processes.

As GORE, MDD, and process maturity models are focused on achieving/increasing the quality of the software product, we believe they are rather complementary. In this context, the following research question is proposed: *How can be designed a GORE-based MDD process to fulfill the requirements of a software process maturity model?* Since this challenge has not been properly addressed by any software process yet, research into this area is relevant and necessary.

Towards answering the research question, in this paper we propose a Goal-Oriented software process (hereafter called GO-MDD) based on the *i** framework and on OO-Method (an industrially applied MDD approach) [20], which is compliant with the requirements development (RD) process area (PA[1]) of CMMI[2]. We have focused on the compliance with RD because, similarly to *i**, the main objectives of this process area are the elicitation and/or specification of system requirements.

The contribution of this paper is twofold. First, practitioners that follow or plan to follow a maturity model and at the same time want to combine GORE and MDD can adapt this proposal instantiating it to their specific needs. Second, the paper can be useful in academia as reference for further research on combining different instances of GORE, MDD, and software process maturity models.

The rest of this paper is organized as follows: Section 2 presents relevant background. Section 3 describes the GO-MDD process. Section 4 presents the analysis of GO-MDD in regard to the RD process area. Section 5 discusses relevant related works. Finally, Section 6 shows the conclusions and proposes future work.

---

[1] A cluster of related practices that, when implemented collectively, satisfies a set of goals for making improvements in an area.

[2] In this paper, the terms CMMI and CMMI-DEV are used as synonyms.

2 Background

The reasons for choosing *i**, OO-Method, and CMMI as the basis for the GO-MDD proposal are the following: *i** is currently one of the most widespread GORE modeling and reasoning frameworks [28][32]; OO-Method is a MDD approach that has been successfully applied in the software industry [19] and; finally, CMMI is the most frequently adopted software process maturity model [22][30]. This section provides a brief explanation of these software development approaches.

2.1 The *i** Goal-Oriented Requirements Framework

The *i** framework [35] emphasizes the analysis of strategic relationships among organizational actors to capture intentional requirements. An actor generically refers to any unit for which intentional dependencies can be ascribed. Actors are intentional in the sense that they do not simply carry out activities and produce entities, but also have desires and needs. The framework offers two types of models: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model.

The SD model focuses on external relationships among actors. It includes a set of nodes and connecting links, where nodes represent actors (*depender* and *dependee*) and each link indicates a dependency (*dependum*) between two actors. There are four possible *dependum* elements: goal, resource, task, and *softgoal*. A goal is a condition or state of concerns that an actor would like to obtain. A resource is a physical or informational entity that must be available for an actor. A task specifies a particular way of doing something and can be decomposed into small sub-tasks. Finally, a *softgoal* is associated to non-functional requirements.

The SR model is a detailed view of the SD model that shows the internal actor relationships. In addition to the dependencies that are present in the SD model, the SR model incorporates three new types of relationships: (i) task-decomposition links, which describe what should be done to perform a certain task; (ii) means-end links, which suggest that a task is a means to achieve a goal; (iii) contribution links, which suggest how a model element can contribute to satisfy a *softgoal*.

2.2 The OO-Method MDD Approach

OO-Method [20] is an object-oriented method that allows the automatic generation of the final application code from a conceptual model. It is supported by the industrial tool *OlivaNova* [19] and provides a precise UML-like notation, which is used to specify a Conceptual Schema that describes a system at the problem space level. The development process suggested by OO-Method has two phases (Fig. 1): *Development of a Conceptual Schema* and *Generation of a Software Product*.

The first phase consists of eliciting and representing the essential properties of the information system under study, thereby creating the corresponding conceptual schema. In the second phase, a precise execution model, conformed by a set of compilation patterns, indicates the correspondences between the conceptual schema

and pieces of code in a target implementation platform. Thus, the application code is automatically generated for an input conceptual schema.

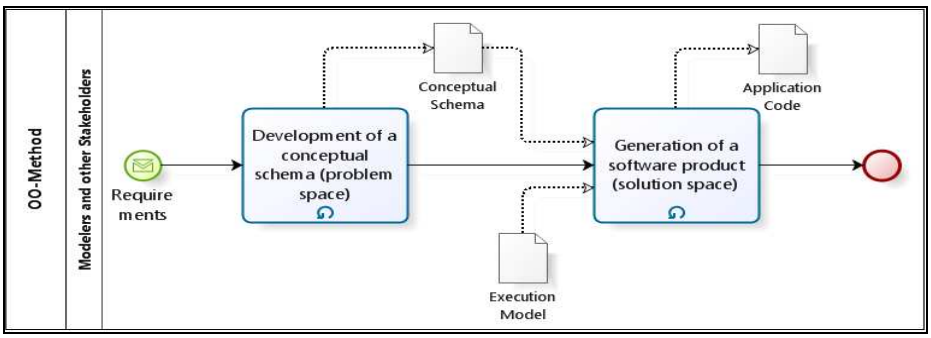

Fig. 1. Phases and artifacts of the OO-Method MDD approach.

2.3 CMMI-DEV

CMMI-DEV [26] is a guide to implement a continuous process improvement for developing products and services. For accomplishing this task, it provides two representations: ***Staged***, which assesses the maturity level of a whole development process from an organization; and ***Continuous***, which assesses the capability level of individual process areas (PAs), selected based on the organization's business goals. The process framework described in this paper (see Section 3) is related to the continuous representation, since it is focused on only one process area (PA): requirements development (RD). It complies with the capability level 1 of RD, which is considered, according to CMMI, the basis for improvement initiatives in a specific PA. A meta-model for the continuous representation is presented in Fig. 2.

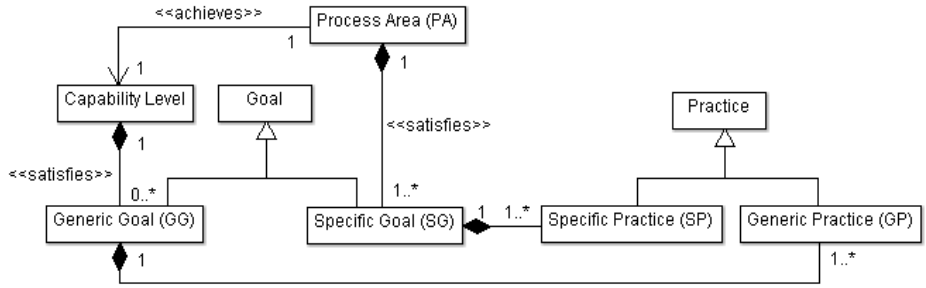

Fig. 2. CMMI Continuous Meta-model, adapted from [17][26].

In the continuous representation, the achievement of a capability level depends on goals and practices (decomposition of goals) of two types: **1)** specific goals (SGs) and specific practices (SPs), which are applied only to a particular PA; and **2)** generic goals (GGs) and generic practices (GPs), which are applied equally to all PAs that achieve a specific capability level. From the assessment of practices and goals, which

is performed on a bottom-up way (from the practices up to the goals), it is possible to classify the capability level of a PA on a scale from 0 to 3 (for details see [26]).

3 The Proposed Process Framework: GO-MDD

The process framework, GO-MDD, extends the works published in [1][2][8][9][21]. It is composed of six stages that are performed through an iterative and incremental development cycle (Fig. 3). Thus, after performing all the stages, the cycle can re-start for a new iteration if the product being developed is not finished yet.

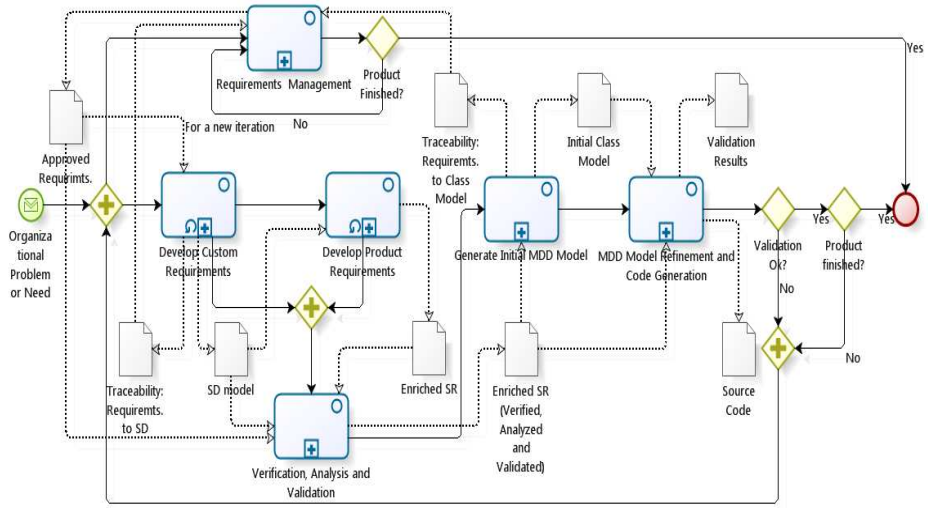

Fig. 3. Stages and artifacts of GO-MDD.

We call GO-MDD a process framework and not simply a process because it has a character most descriptive rather than prescriptive, meaning that it focuses on the "what" rather than on the "how" (a detailed discussion on the differences between descriptive and prescriptive processes is presented in [23]). Hence, GO-MDD can be instantiated for each organization prior to its use (e.g., the way of doing the requirements elicitation or their validation).

In particular we apply the CMMI perspective to create a process framework that automatically integrates the *i** framework into a concrete MDD processes (OO-Method). This process framework presents specific transformation guidelines, *i** extensions, and verification mechanisms to assure the correct generation of initial MDD models from *i** models. Then, by means of refinement of this MDD model, a fully executable application that is aligned with the stakeholder requirements is generated. Since the proposed integration of *i** and MDD is based on the class model generation, the results presented in this paper can be used as reference for other object-oriented MDD processes, such as UML-based proposals.

An example, related to the management of work requests in a Photography agency, extracted from the experiment presented in [9], is used to explain the process

framework GO-MDD. The Photography agency is dedicated to the management of photo reports and their distribution to publishing houses. This agency operates with freelance photographers, which must present a work request to its production department. Due to space constraints, only those aspects that are not part of *i** and are more relevant to the paper's objectives are illustrated.

First stage: Develop Custom Requirements. Requirements from various stakeholders are consolidated, prioritized (according to stakeholder needs and constraints) and detailed to be implemented in the current iteration. These requirements can be elicited in the current iteration or come from a backlog (i.e., a list) of previously approved requirements (see the description of the *Second Stage*). The requirements backlog and the iterative development cycle were inspired from *Scrum* [25]. As the result of this stage, an SD model is produced and the traceability (i.e., a mapping) from requirements to the SD model is created/updated.

Second stage: Requirements Management. This stage is responsible for monitoring the requirement requests from several stakeholders and performing initial requirements elicitation and analysis (of adequacy and impact) in order to decide whether the requests will be approved (or not) to be developed in some iteration. It is executed in parallel with the first stage, until the product is satisfactorily produced. As the result of this stage, a backlog of approved requirements is created/updated.

Third stage: Develop Product Requirements. In this stage, detailed in [2][21], an initial SR model is produced from the refinement of the SD model. The goals defined in the SR model are analyzed to decide the intentional elements that must be considered as requirements of the system to be. These elements are highlighted by means of specific stereotypes, which introduce information to automatically perform the corresponding MDD model generation. Thus, an enriched SR model is produced.

Fig. 4 shows an example of an *i** SR model, related to the Photography agency, extended with the stereotypes defined for the integration with OO-Method. This SR model shows that the *production department* depends on the reception of *work requests* (i.e., job applications), which are produced by *photographers* that want a *work opportunity*. The *work requests* are comprised by the photographer's *personal data*. The *production department* is responsible for *refusing* or *accepting* the *received work requests* by indicating the final *work request status*. For the accepted requests, a *photographer level* is assigned according to the information provided by the *Commercial Department*. The stereotypes that extend the *i** SR model introducing specific information to generate the corresponding MDD (class) model according to the OO-Method approach, and their main application to the transformation process are briefly described as follows (further information can be found in [2]):

- **SActor:** Indicates that an *i** actor must be maintained by the corresponding system. This actor will be represented by means of a class in the generated MDD model.
- **SPhysicalR:** Indicates that an *i** resource is considered as a physical resource that must be maintained in the system as a class in the generated MDD model.
- **SInfoR:** Indicates that an *i** resource is considered as an informational resource that must be maintained in the system as class attribute in the generated MDD model.
- **STask:** Indicates that an *i** task must be considered for the system behavior. This will be represented as a class service in the generated MDD model.

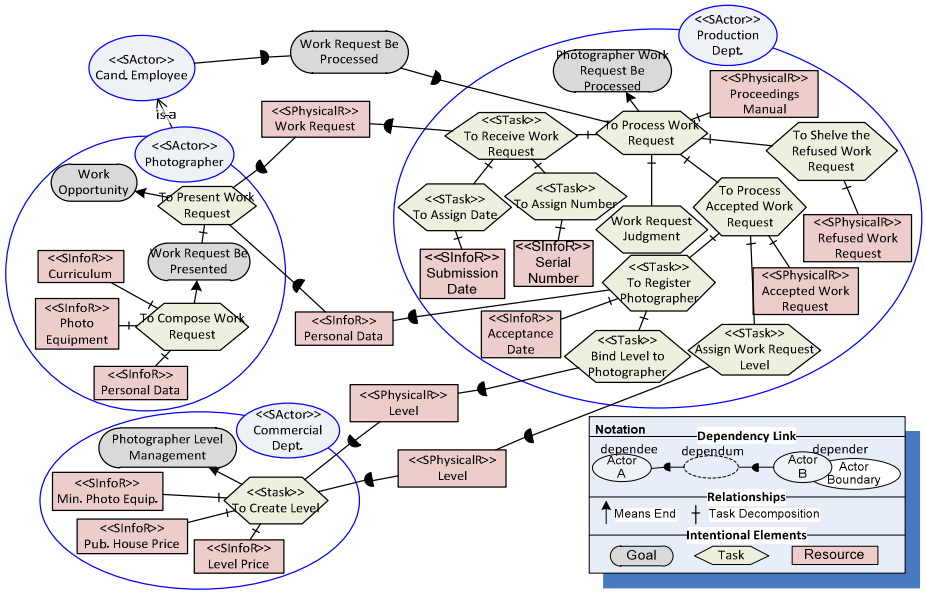

Fig. 4. Example of extended *i** SR model.

Fourth stage: Verification, Analysis, and Validation: The models defined in the previous stages are used as input for this stage. An analysis is performed to guarantee that the requirements defined in the SR model are necessary and sufficient to meet the organizational goals and to balance stakeholder's needs and constraints. Then, the requirements are validated with the stakeholders to guarantee their correctness and completeness and, if any problem is detected in the analysis or in validation, it must be fixed. Later, the resultant SR model is verified by means of a set of measures (detailed in [9]), which evaluate the elements extended. These verification measures are formally specified by means of OCL rules [18] and guarantee the completeness of the MDD model generation in relation to the requirements indicated in the *i** model. This has been demonstrated by means of the controlled experiment presented in [9]. The measures not only identify the modeling issues, but also provide fixing guidelines to improve the SR model and the MDD model generation. Hence, if some problem is detected during the model verification, it must be fixed before getting to the next stage. For instance, the measure *Wrong Attribute Generation* (Table 1) specifies that an *i** resource stereotyped as an informational resource (SInfoR) must be related to a system actor (SActor) or to a physical resource (SPhysicalR), which are transformed into a class. Otherwise, the informational resource cannot be transformed into a class attribute due to the lack of a class that contains it.

Table 1: Some characteristics of the measure *Wrong Attribute Generation*

Characteristic	Definition
Measurement Scale	Ratio scale
Attribute to be measured	Informational resources not related to a physical resource or to an actor.
Measurement principle	This kind of informational resource corresponds to a wrong attribute generation in the MDD model.
Measurement procedure	The attributes to be measured must be counted to obtain the number of informational resources that cannot be transformed into attributes.

Fifth stage: Generate Initial MDD Model: Once the *i** SR model has been verified and improved according to the corresponding verification measures, it is transformed into an initial MDD model (class model) by means of a set of model-to-model transformations (detailed in [2] and [21]). We refer to an initial MDD model and not a complete one because there are aspects related to specific system functionality that cannot be obtained from requirements models.

Traceability from requirements to the MDD model is also produced in this stage. It is important to point out that the class model is the central model in the OO-Method approach. The rest of the models that are necessary to completely specify the OO-Method conceptual model (such as the presentation or the functional model) are derived from this central model. Thus, the traceability from requirements to the other OO-Method models can be obtained from the association of requirement elements to the corresponding class model elements. Fig. 5 shows the class model obtained from the extended *i** model presented as example in Fig. 4.

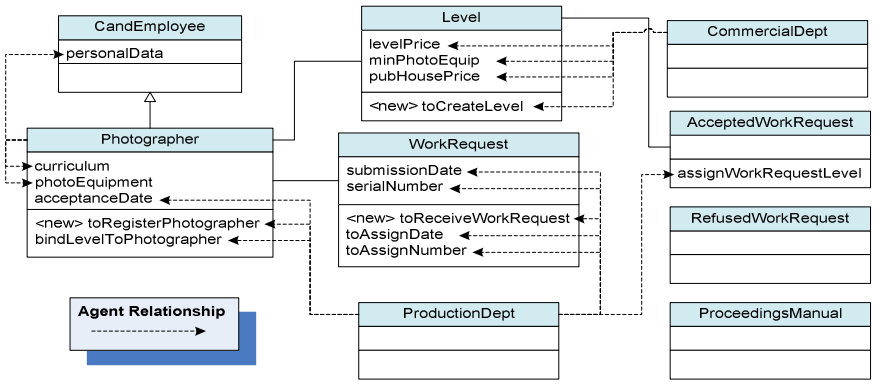

Fig. 5. Initial class model generated from the extended *i** SR model.

A specific OO-Method construct that differs from the traditional (UML-like) class model notation can be observed in the generated class model. This is the agent relationship, which indicates the visibility that a class has over attributes or services of other classes of the model. Agent relationships are defined between a class generated from an *i** actor and the elements generated from *i** elements that are inside the boundary of the actor transformed (e.g., the agent relationship that is defined from the class *ProductionDept* to the service *toReceiveWorkRequest*). These relationships provide relevant information for defining presentation models related to the specification of users' interactions with the final system.

Sixth stage: MDD Model Refinement and Code Generation. The initial MDD model generated is refined to introduce those design aspects that cannot be obtained from the transformation of the enriched *i** SR model. Some of the refinements that must be performed are the specification of additional class services, association of cardinalities, or specific system constraints. The refined model is verified by means of a facility provided by the OO-Method modeling tool [19], which guarantees the correct specification of the MDD model for the automatic code generation. The system source code is generated by applying the OO-Method model compilation

technology. Finally, the generated source code must be compiled and executed in order to be validated against the corresponding requirements.

3.1 Research Method for the Design of GO-MDD

The design of GO-MDD was done according the following research method. First, based on previous works of our research group, regarding the integration of *i** and OO-Method [1][2][8][9][21] (that propose the use of stereotypes, transformation guidelines and verification measures), an initial version of the process framework was specified. The compliance of this version was analyzed against the RD process area and, based on the gaps identified in the analysis, new characteristics, activities and artifacts were included into GO-MDD to make it fully compliant with this process area. Finally the compliance mapping, described in the next section, was produced. No exclusions were done from the original process, and the inclusions were mainly related to the consolidation, prioritization and traceability of requirements (*First Stage*); the whole Requirements Management (*Second Stage*); the analysis and validation of requirements (*Fourth Stage*); and the definition of an interactive/incremental cycle (performed along the whole process framework).

4 Compliance Mapping from the RD Process Area and GO-MDD

According to CMMI-DEV, the purpose of RD is to elicit, analyze and establish customer, product, and product component requirements. A compliance mapping between the capability level 1 of RD and the proposed process framework (GO-MDD) is presented in this section. For each SG, its purpose is described, and for all the corresponding SPs, a mapping relating stages, activities, and artifacts of GO-MDD to each SP is produced. To comply with the capability level 1, a process must satisfy the generic goal (GG) associated to this level (GG 1), which has only one generic practice (GP 1.1) that requests all the SGs associated to the PA to be satisfied (if at least one of the SGs is not satisfied, the PA is considered to have capability level 0). For evaluating capability levels higher than 1, a PA must satisfy the GG associated to the specific level, which imposes other requirements, and all the GGs associated to the lower levels. Fig. 6 instantiates the meta-model presented in Fig. 2 to represent the elements involved in the compliance mapping for level 1.

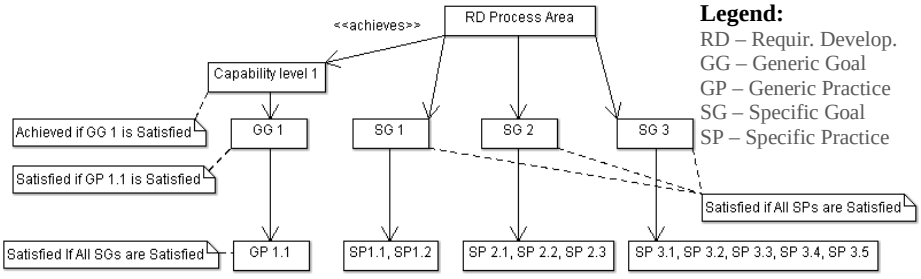

Fig. 6. The capability level 1 for the RD process area

SG 1 Develop Customer Requirements: This goal addresses the collection of stakeholder needs, expectations, constraints and interfaces, and their translation into customer requirements.

SP 1.1 Elicit Needs: requirements should be elicited and a requirements specification (e.g., a textual document or a model) should be produced. The requirements specified (functional and non-functional) express stakeholder needs, expectations, constraints, and interfaces for all the phases of the product lifecycle.

Compliance mapping: A preliminary requirements elicitation is performed in the *Requirements Management* stage and a list of approved requirements is produced. Then, in the *Develop Custom Requirements* stage, the requirements to be implemented in the current iteration are detailed, and an initial *i** model (SD) is produced with the definition of different organizational actors (stakeholders) and their dependencies.

SP 1.2 Transform Stakeholder Needs into Customer Requirements: requirements elicited from various stakeholders (including business and technical functions) should be consolidated, analyzed regarding missing information and presence of conflicts, and prioritized according to some criteria. Requirements specific to verification and validation (V&V) for the system to be can also be elicited.

Compliance mapping: The SD model produced in the *Develop Custom Requirements* stage is a consolidated and prioritized specification of the needs from various stakeholders.

SG 2 Develop Product Requirements: This goal addresses the refinement and elaboration of customer requirements in order to develop product and product component requirements. Some of the practices associated to this goal can be performed during or in conjunction with a design stage.

SP 2.1 Establish Product and Product Component Requirements: product and product component requirements should be derived (identified) from customer requirements. Product requirements are functional and non-functional requirements expressed in technical terms that can be used for design decisions. Modifications on customer requirements due to approved requirements changes must be reflected in the derived requirements. Derived requirements also address the needs of other lifecycle phases (e.g., production, operations and disposal).

Compliance mapping: The enriched SR model produced in the *Develop Product Requirements* stage is a refinement of the SD model and specifies which requirements (functional and non-functional) are allocated to the products and product components to be developed. Modifications on customer requirements are captured in the *Requirements Management* stage, and their impacts on the derived requirements are analyzed based on the traceability from requirements to the SD model (produced in the *Develop Custom Requirements* stage), on the refinement relationship between the SD and the SR models, and on the traceability from requirements to class model (produced in the *Generate Initial MDD Model* stage).

SP 2.2 Allocate Product Component Requirements: the product components requirements (functional and non-functional) should be allocated to product components of the defined solution.

Compliance mapping: In the *Generate Initial MDD Model* stage, the initial class diagram generated is a first approximation for the allocation of requirements to product components. Later, this allocation can be updated when the class diagram is refined in the *MDD Model Refinement and Code Generation* stage.

SP 2.3 Identify Interface Requirements: interface requirements between functions, objects or other logical entities should be identified.

Compliance mapping: The initial class diagram, produced in the *Generate Initial MDD Model* stage, includes classes, methods, attributes, and associations. It defines the interfaces among entities that are identified from the requirements. Later, these interfaces can be updated when the class diagram is refined for code generation.

SG 3 Analyze and Validate Requirements: This goal addresses requirements analysis and validation. Its specific practices support the development of the requirements in SG 1 and SG 2. Some of the practices associated to this goal can be performed during or in conjunction with a design stage.

SP 3.1 Establish Operational Concepts and Scenarios: operational concepts and scenarios should be identified and maintained (i.e., updated when necessary). Operational concepts are general descriptions of the ways in which entities are used or operate. Scenarios are detailed sequences of events that make explicit some of the functional or quality attribute (non-functional) needs of the stakeholders.

Compliance mapping: The SD and SR models illustrate the tasks and subtasks (i.e., the way of doing something) related to the satisfaction of goals that the actors would like to achieve. In particular, the enriched SR highlights the tasks and subtasks related to processes that will be automated.

SP 3.2 Establish a Definition of Required Functionality and Quality Attributes: a definition of the required functionality and quality attributes should be established and maintained.

Compliance mapping: The enriched SR in conjunction with the initial class diagram specify the quality attributes (*softgoals*) and required functionalities (tasks and class services) that are related to the requirements elicited. Later, when the class diagram is refined, these quality attributes and functionality can be updated.

SP 3.3 Analyze Requirements: the requirements for one level of the product hierarchy should be analyzed to determine if they are necessary and sufficient to meet the objectives of higher levels (i.e., the practice analyses the consistence between requirements in different levels of hierarchy).

Compliance mapping: satisfaction of this SP is discussed together with the next one.

SP 3.4 Analyze Requirements to Achieve Balance: requirements should be analyzed to balance stakeholder's needs and constraints, such as cost, schedule, product or project performance, functionality, priorities, reusable components, maintainability, and risks.

Compliance mapping: The SP 3.3 and SP 3.4 are satisfied in the following way. Preliminary requirements analysis is performed during the *Requirements Management* stage and also in the *Develop Custom Requirements* stage, when requirements are consolidated and prioritized according to stakeholder needs and constraints. In the

Verification, Analysis, and Validation stage, the consistence among requirements from different levels of hierarchy is analyzed. Additionally, the defined measures automatically verify the *i** models in the context of the MDD process to assure the transformation completeness of the necessary requirement elements. Thus, the MDD model generated provides a complete representation (at design time) of all the artifacts defined at requirements level.

SP 3.5 Validate Requirements: requirements should be validated to ensure that the resulting product will perform as intended in the end-user environment.

Compliance mapping: Prior to the generation of the initial class model, in the *Verification, Analysis, and Validation* stage, the requirements are validated with the stakeholders in order to guarantee their correctness and completeness. Also, the iterative and incremental refinement cycle that is present in our proposal allows automatically generated class models to be used to generate prototypes, which are validated by the stakeholders to assure the correct implementation of their needs. Hence, at each iteration, an initial MDD (class) model can be automatically generated from the requirements and later refined to obtain a complete and precise description of the generated MDD elements in order to perform the model compilation into an increment towards the final software product. Thus, necessary changes in the requirements are introduced in the defined *i** models to generate a new version of the corresponding class models, and to perform a new model compilation and validation.

5 Related Work

In the literature, we have not found any work that proposes a software process based on the automatic integration of GORE with MDD and compliant with a maturity model. However, there are works which treat the pair-wise association of these software development approaches. Some of these works are described as follows.

The integration between GORE and MDD has been discussed in several works. However, most of these works (such as [12][14][24]) are not based on standards or well-defined processes, nor do they introduce automation possibilities. Therefore, the application of these proposals must be manually performed [15], which is not a suitable option since the manual translation of models is a time consuming and error prone task [13].

In relation to the integration of GORE with a software process maturity model, in [3] it is proposed an approach for requirements development and management in the context of system family engineering, which, according to the authors, complies with the RD and RM[3] process areas of CMMI. In this approach, high abstraction level *goals* (related to functional aspects) and *softgoals* (related to quality aspects) are the first means used to elicit requirements. However, as opposed to our work, an explicit mapping identifying which are all the evidences to attest its compliance with CMMI is not presented; neither the approach is integrated into a complete software process from requirements to code.

[3] The Requirements Management process area is responsible for tracking requirements changes, analyzing the impacts of the changes and maintaining the requirements traceability.

Although there are some specific works related to the compliance of MDD approaches with CMMI or its ancestor (CMM) [4][6][7][29], they fail to deal with this issue properly. These works do not explain in detail how an approach complies with the maturity model, where the approach should be adjusted for compliance, and whether/where the approach conflicts with the maturity model requirements.

Hence, unlike those previous works, we proposed a complete software process framework, which integrates GORE and MDD through automatic transformation from requirements to initial design models and compliant with a software process maturity model. To demonstrate the adherence of the GO-MDD process framework with the maturity model, a detailed compliance mapping was presented.

6 Conclusions and Future Work

In this work, based on the research question *How can be designed a GORE-based MDD process to fulfill the requirements of a software process maturity model?*, we advocated that GORE and MDD can be put together to comply with the requirements of a software process maturity model, thus supporting the application of a GORE approach into real (industry) scenarios. In order to demonstrate the soundness of this idea, we have proposed a software process framework based on *i** and OO-Method, and have described how it complies with the RD process area of CMMI-DEV. Even though the work has been based on specific instances of GORE, MDD, and a software process maturity model, we believe that this idea can be generalized to other instances, but further research need to be done in this direction.

This proposal is part of a wider work that is related to the use of GORE and MDD to define a full software process, compliant with a software process maturity model, covering the long path that goes from goal-oriented requirements modeling to a final high-quality software product. In this context, several future works can be developed: **1)** Extending the proposed process framework to be compliant with other process areas and capability levels of CMMI is necessary. Despite the framework proposed already presents some characteristics that partially meet other PAs, such as "requirements management" and "project monitoring and control", additional characteristics must be considered to be compliant with these and further PAs; **2)** We are aware that the evaluation of the proposal in real development scenarios is necessary. Hence, we consider as future work the development of empirical studies to validate the feasibility and the effectiveness of the proposed process framework, and to analyze the practical implications of its use in an industrial context; **3)** The investigation of the research question in the scope of other software development approaches (i.e., different GORE, MDD and/or maturity models approaches) needs to be done; **4)** Finally, a systematic literature review [11] should be conducted to verify in deep the existence of other related works.

Acknowledgments. This work has been developed with the support of the Brazilian Research Agency CAPES, under the grant #BEX3229/10-6; and the Spanish Government, under the projects ORCA (PROMETEO/2009/15), PROS-REQ (TIN2010-19130-C02-02) and PTA2008-1443-P (co-financed by ERDF).

References

1. Alencar, F., Marín, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.H.: From *i** Requirements Models to Conceptual Models of a Model-Driven Development Process. In: Proc. of PoEM 2009. Springer LNIBP (2009)
2. Alencar, F.M.R., Pastor, O., Marín, B., Giachetti, G., Castro, J.: Aligning Goal-Oriented Requirements Engineering and Model-Driven Development. In: Proc. of 11th International Conference on Enterprise Information Systems (ICEIS), pp. 347–350 (2009)
3. Cerón, R., et al.: A Meta-Model for Requirements Engineering in System Family Context for Software Process Improvement using CMMI. In: Proc. of PROFES 2005, LNCS, vol. 3347, pp. 173-188. Springer, Heidelberg (2005)
4. Crag Systems: The Model-Driven Development Process. http://www.cragsystems.co.uk/development_process (2008), Last access 2011/06/30
5. de la Vara, J. L., Sánchez, J.: System Modeling from Extended Task Descriptions. In: Proc. of SEKE 2010, pp. 425-429 (2010)
6. ESI. Center: Model-driven Architecture inSTrumentation, Enhancement and Refinement. IST-2001-34600, MASTER-2003-D3.2-V1.0-PUBLIC (2003)
7. Fricker, S.: Introducing Model-Driven Development for CMMI Engineering Process Areas. 18th Software Engineering Process Group Conference (SEPG 2006). http://www.secc.org.eg/sepg%202006/ingredients/Indexes/authorindex.html#f, Last access 2011/06/30
8. Giachetti, G., Alencar, F., Marín, B., Pastor, O., Castro, J.: Beyond Requirements: An Approach to Integrate *i** and Model-Driven Development. In: Proc. of XIII Conferencia Iberoamericana en Software Engineering (CibSE 2010) (2010)
9. Giachetti, G., Alencar, F., Franch, X., Marín, B., Pastor, O.: Technical Report ProS-TR-2011-07: Automatic Verification of Requirement Models for Their Interoperability in Model-Driven Development Processes. Universidad Politécnica de Valencia (2011)
10. Kamata, M.I., Tamai, T.: How Does Requirements Quality Relate to Project Success or Failure? In: Proc. of RE 2007, pp. 69–78. IEEE (2007)
11. Kitchenham, B., Charters S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering Version 2.3. Keele University and Durham University Joint Technical Report EBSE-2007-01 (2007)
12. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modeling Approach. Info Systems 29(2), pp. 187–203 (2004)
13. Maiden, N., Jones, S., Manning, S., Greenwood, J., Renou, L.: Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. In: Proc. of CAiSE 2004. LNCS, vol. 3084. Springer, pp 368–383 (2004)
14. Martínez. A., Castro, J., Pastor, O., Estrada, H.: Closing the Gap between Organizational Modeling and Information System Modeling. In: Proc. of WER2003, pp 93–108 (2003)
15. Martínez, A.: Conceptual Schemas Generation from Organizational Models in an Automatic Software Production Process. PhD Thesis. Univ. Politéc. de Valencia (2008)
16. Mohagheghi, P., Dehlen, V.: Where is the Proof? – A Review of Experiences from Applying MDE in Industry. LNCS, vol. 5095, pp. 432-443. Springer (2008)

17. Monzón, A.: AQAP-160 vs. CMMI. Slides Presentation. Mil. Transport Aircraft Division. http://www.calidaddelsoftware.com/documentos/II%20Semana%20CMMI/03-%20EADS-CASA.pdf (2006), Last access 2011/06/30
18. OMG: Object Constraint Language 2.2 Specification (2010)
19. Pastor, O., Molina, J.C., Iborra, E.: Automated Production of Fully Functional Applications with OlivaNova Model Execution. ERCIM News, nº 57 (2004)
20. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. 1st edn. Springer, New York (2007)
21. Pastor, O., Giachetti, G.: Linking Goal-Oriented Requirements and Model-Driven Development. In: Nurcan, S., Salinesi, C., Souveyet, C., Ralyté, J. (eds.) Intentional Perspectives on Information Systems Engineering, pp. 257–276. Springer-Verlag (2010)
22. Pino, F.J., García, F., Piattini, M.: Software process improvement in small and medium software enterprises: a systematic review. Soft. Quality Journal 16(2), pp. 237-261 (2008)
23. Ruhe, G.: Learning Software Organizations. Fraunhofer Institute for Experimental Software Engineering. ftp://cs.pitt.edu/chang/handbook/09.pdf, Last access 2011/06/30
24. Santander, V., Castro, J.: Deriving Use Cases from Organizational Modeling. In: Proc. of RE 2002, pp 32–42 (2002)
25. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Microsoft Press (2004)
26. SEI: CMMI for Development, Version 1.3, CMU/SEI-2010-TR-033. http://www.sei.cmu.edu (2010), Last access 2011/06/30
27. Selic, B.: The Pragmatics of Model-Driven Development, IEEE Software, vol. 20, no. 5, pp. 19-25 (2003)
28. Shuichiro, Y., Haruhiko, K., Karl, C., Steven, B.: Goal Oriented Requirements Engineering: Trends and Issues. IEICE – Trans. Inf. Syst. E89-D, 2701–2711 (2006)
29. Steinhau, R., et al.: Guidelines for the Application of MDA and the Technologies covered by it. Deliverable 3.2, MODA-TEL Consortium, IST-2001-37785, Interactive Objects Software GmbH (2003)
30. Unterkalmsteiner, M., et al.: Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review. IEEE TSE (2011)
31. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: a Guided Tour. In: Proc. of RE 2001. Springer (2001)
32. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: a Roundtrip from Research to Practice. In: Proc. of 12th IEEE Joint International Requirements Engineering Conference. IEEE Computer Science, pp 4–8 (2004)
33. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley (2007)
34. Xavier, F.: Fostering the Adoption of *i** by Practitioners: Some Challenges and Research Directions. Intentional Perspectives on Information Systems Engineering, pp. 177–193. Springer (2010)
35. Yu, E.: Modelling Strategic Relationships for Process Reengineering, PhD Thesis, University of Toronto, Toronto, Canada (1995)
36. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering (2011)