Automatized integration of a contextual model into a process with data variability

Jacques Simonin, John Puentes

▶ **To cite this version:**

Jacques Simonin, John Puentes. Automatized integration of a contextual model into a process with data variability. Computer Languages, Systems and Structures, 2018, 54, pp.156 - 182. 10.1016/j.cl.2018.06.002 . hal-01836388

HAL Id: hal-01836388
https://hal.science/hal-01836388

Submitted on 9 Dec 2020

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.

Automatized Integration of a Contextual Model into a Process with Data Variability

Jacques Simonin*, John Puentes

IMT Atlantique, Lab-STICC, UBL - CS 83818 F29238, Brest Cedex 3, France

ARTICLE INFO

Article history:

Keywords:
Data variability
Contextual data
Model transformation
Substitution transformation
Enhancement transformation

ABSTRACT

Existent process models can hardly cope with the emerging issue of modelling exponential variable data volumes in systems' workflow, from specifications to operation. Given the strong relation between data context and data variability, this paper considers the automated integration of contextual models for processes with data variability. The proposed approach extends methodologically a platform independent model process, using a contextual data model, to obtain automatically the corresponding platform specific model. Contextual data are thus integrated to a process as a model, within a process. Two particular cases of contextual data models are studied in detail: substitution, when the contextual data model defines generated code, and enhancement, when learned data descriptions constitute the contextual data model. The feasibility and value of integrating a contextual model into a process to handle data variability are shown in detail describing these two use cases. Contextual model integration by substitution to include automatically variable ready to use application services to generate code, and contextual model integration by enhancement applied to supervised image classification based on variable descriptors. Results show that relating data variability and its context by means of automated integration of a designed system component model, simplifies variable data processing of system process models.

1. Introduction

Modelling of variable data – documents, software modules, images, signals, videos, multimedia content, etc. – for system processes, is an emerging issue in model-based system design. The exponential and permanent generation of variable data makes unfeasible to extend or adapt existent system process models to cope with the variability of data production. Moreover, this variability often relates to a data context [1], which is either ignored [2] or partially modelled with multiple constraints [3] because of the complexity to represent it.

Usually, context awareness is defined through included conditional relations of a process model. In that case, only if a context parameter has a predefined value, then a specific operation can be triggered. Nevertheless, whenever data production variability is addressed, context awareness has to be defined through integration, by means of substitution or enhancement of a process model. The interest of a contextual model can be illustrated by three examples – that could be modelled – of data context strongly related to production variability as article writing support, application development, and pattern recognition:

- Article writing support: The context is defined by thousands of previously published articles, from which, the notion of cited articles (variable generated data) characterizes part of the article's content, written with a text editing process (integration with substitution).
- Application development: The context is composed by thousands of available enterprise services, related to a process to identify and instantiate adapted application services (variable produced data) in the development of an information system software (integration with substitution).
- Pattern recognition: The context is represented by thousands of content description values, associated to statistical representations of numerous classes (variable defined data), through a process of supervised classification (integration with enhancement).

In the aforementioned examples the processing of data created in various manners could be simplified, if a ready to use context data model is applied. By analogy with off-the-shelf software components [4], such context data model is understood as an optimized guideline of related published articles, available developed services, or performant calculated content descriptors, respectively, in the form of a system component

* Corresponding author. Tel.: +33 (0)229001428
E-mail address: jacques.simonin@imt-atlantique.fr

model. Several categories are associated to contextual data [5]. Among those, the use of contextual data done by a process is part of the *relations* category and its sub-category of *functional relations* – between contextual data and the process. To develop modeling suitably aimed at considerable volumes of data, we consider the automated integration of contextual models for processes with variable produced data. It is consequently a problem of defining the transformation of available data, into data that influence a process, i.e. knowledge. For this specific problem, data variability stems particularly from the different modes of data production. In the previous introductory examples, a quoted article can be produced by queries to different publishers and comply with one of various specific standards (APA, Harvard, ISO, MLA, etc.). On the other hand, the variability of a reusable application service, results from the development of an information system application or library linked to multiple programming languages (C++, Java, C, etc.). As for the statistical representation of a class in pattern recognition, variability results from the use of different descriptors and the choice of one or several out of multiple mathematical distances between the class components, to identify that class.

Different data production modes constitute a factor of variability not previously taken into account for modeling purposes. Hence, we address the question of how to define a contextual model for data produced not in just one but variable manners, generate automatically platform specific programs from that contextual model, and integrate it in a process. Our main assumption is that modelling a variable data context is independent of modelling a process. Furthermore, a meta-model can be defined if it is known which task generated data and in what manner. So, besides handling different modes to generate data, the resulting contextual model provides knowledge with direct impact on the given process. The main contribution of this work is allowing the architect to specify appropriate rules to integrate data into the process, according to variable production modes.

To facilitate the integration of variable contextual data into a process, the essential modelling hypothesis is that a MDA-conform approach produces the data, and thus defines a particular production mode. This implies that the concepts at the input and output of models transformations, as well as relationships describing this mode of contextual data production, are modeled in a meta-model. Therefore, the input concepts of a contextual data production transformation are defined first, instead of adapting a model separately to a process for each production mode. Our approach extends weaving as a result, which is restricted to data generated by a unique production mode.

Among known model-driven system architecture approaches, context data modeling appear to be feasible applying Model Driven Engineering – MDE [6]. The main reasons are because the Object Management Group supports it, generated models are platform independent, MDE is compatible with some modeling standards, it has a largely generic design spectrum, and it is widely used [7]. Additionally, a possibility in the MDE framework could be to apply directly a relational transformation specified as [8]: "*an association between the elements or parameters of two models of a system that induces a further mapping between the relationships in the models*". However, there are not MDE context data models compliant with variability of data production, designed specifically to be seamlessly integrated in a process, but with conditional filtering [9]. Moreover, the automatic generation of platform specific programs obtained from those models applying Model Driven Architecture – MDA [10], has not been defined either, for context integrations related to data variability.

This paper reports on an original approach to generate context data models, integrated to MDA compatible processes, within a MDE framework. The proposed approach extends methodologically a platform independent model (PIM) design process, making use of a contextual data model, before the corresponding platform specific model (PSM) is automatically obtained. In this manner contextual data are integrated to a process as a model, reinforcing complementary knowledge. Two particular cases of contextual data models are studied in detail, substitution – when the contextual data model defines generated code – and enhancement – when learnt data descriptions constitute the contextual data model. The feasibility and practical value of integrating a contextual model into a process for variable data production are shown in this paper through a step-by-step demonstration of both cases. The first case illustrates the feasibility of reusing an information system element to develop an application of that system, while the second case illustrates the suitability for pattern recognition, a process in which the application of model engineering is unconventional.

The paper is organized as follows. Background and related work regarding variable data and context modeling, as well as systems engineering based on models and MDA are described in Section 2. The proposed approach is defined in Section 3, considering how to integrate a contextual model to treat variable generated data by substitution or enhancement. A use case of contextual model integration by substitution is presented in Section 4, developing the automatic integration of ready to use application services to generate code. A use case of contextual model integration by enhancement is presented in Section 5, applied to supervised image classification. In Section 6 findings, lessons learned from both examples, as well as risks for validation are discussed. Conclusions and perspectives are summarized in Section 7.

2. Related work

This section examines previous works on modeling of data variability and context. It then summarizes research initiatives associated to model-based engineering that have made use of these elements.

2.1. Data variability and context modeling

With the extensive complexity of systems, a significant problem in model-based techniques is data consistency, i.e. to seamlessly include variable ways of data production in system architecture modeling, for both system architect and system user. Data production variability in system modeling is challenging because it is complex to represent, constraint, integrate, and trace in the workflow, from specifications to operation. Moreover, there are multiple definitions, sources, and viewpoints, which make very difficult to model and integrate data production variability.

Few works have studied variability in systems architecture, although out of the modeling scope, to comply with fixed or changing requirements. Variability in systems architecture has been analyzed in software product families [11], to manage the complexity introduced in UML models [12], to define features or decision models [13], to be described and shared by groups of systems [14], as data-centric to improve model consistency [15], or model transformation rules [16]. Otherwise, few works have examined the problem of variability in context modeling. An ontology-based model was proposed to pilot electricity generation using wind turbines depending on the operational context [17]. Also, it was determined that even if contextual data constrain a process,

context representation is essential to dynamically adapt a system before process execution conditions are taken into account [18].

2.2. Contextual model and model-driven engineering

The application of constraints to define models is well known in the literature. Such approach can be for instance context-related to design user-centered models of web services [19]. A language like ContextUML [20] and recent multi-agent models [21] on which user behavior is crucial, are based on the same principle. Nevertheless, the approach to define contextual models must be extended when a process, not user behavior, defines the constraints. MDE is appropriate to implement such extension, given its generalization possibilities [22]. Besides, it is a prevailing solution to define system architecture applying gradual constraints, by refining the initial system specifications [23]. Since a model oriented system architecture definition can be based on refinement in accordance to MDA [24], MDE includes an architecture activity considered as specifications refinement, applied up to code generation [25]. Refinement is basically applied to produce sets of structured and connected modules or applications. These represent coherent and stable decision rules, to achieve a given data or information processing task [6]. Refinement is fundamental to define multi-model-driven [26] and single-model-driven [27] system architectures.

MDE has been applied in industry to improve the costly automation of code generation [28], linked to software development traceability. For example, use cases were transformed by refinement into system design model [29] and specifications development [30]. MDE has also been extended to a contextual model for multi-agent system design [31]. It has remained close to code service architectures generation from business processes [32], aligning business processes and a component infrastructure based on service-oriented architecture. Yet, MDE automation has been only applied to software development of detailed architecture and the corresponding code generation, using a restrictive framework associated to an applicative layer of the system [33].

Otherwise, system architecture can be generated using pre-designed solutions, defining a contextual model consistent with the system specifications. A contextual model could contain the basic functions of an information system. These pre-designed system functions have been defined by enterprise architects viewpoint design, to compose a company's system architecture [34]. Alternatively, other process modelling techniques use so-called contextual data that add properties (tagged values) of a model element to model languages [35]. Moreover, some models integrate available data before process execution, like the definition in advance of specific services matching constraints and run upon request [36]. Alternatives using MDA have been developed to adapt system architectures to a mobile network [37] and a ubiquitous system [38]. Nevertheless, proposed approaches are neither associated to a methodology enabling software development automation, nor developed and documented in detail [19]. Besides, when a process is executed in MDE a model for contextual data differs significantly from a model to integrate available data before the process execution.

Our contribution is the definition of a methodology to integrate automatically a contextual model into processes with data production variability. Particularly, this paper illustrates with a proof of concept, how defined contextual data models are integrated to a process and automatically transformed in platform specific programs applying a MDA implementation within a MDE framework. Consequently, code generation results from an input contextual model, instead of an adaptation at the code generation level.

3. Integration of a contextual model into a process with data production variability

This section recalls first the prevailing application of MDA along with known practices to integrate models. The proposed approach is then defined in detail.

3.1. Data variability and MDA

Conventionally, MDA (Fig. 1) permits to obtain a PSM (Platform Specific Model), resulting from the application of a transformation *T* to a PIM (Platform Independent Model), under the technical constraint of a PDM (Platform Description Model) [39]. MDA is mainly used to generate code according to functional specifications of a system. Typically, a stipulated functional scenario description – PIM – is transformed into an applied coded service – PSM – for instance, by means of a Java execution environment model – PDM – and a set of rules defined to fix the environment constraints of the transformation.

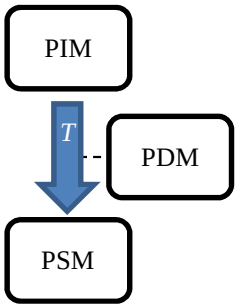


Fig. 1 - Model Driven Architecture for model transformation.

One solution to integrate a contextual model is to chain the transformations, using modeling languages to input models for business process [40], or adopting several transformation languages [41]. Such chaining takes sequentially into account the technical environments as a progression of constraints, but cannot be applied to insert contextual data in a transformation. Since constraints defined in the PDM are a generic solution implemented later by the PIM instances, it is not feasible to integrate and apply contextual data directly in the transformation if a chain of transformations is applied.

A second common model integration solution is composition [42], which can be automatic or manually defined by an expert who determines the mapping of elements to integrate [43]. Therefore, in the composition of two PIMs (PIM1 and PIM2) the respective PSMs are modified directly and the code is produced again. Otherwise, the original PSMs can be integrated using glue code (Fig. 2). This avoids the individual modification of PSMs and code regeneration, while defining the integration rules according to a composition circumscribed between the PIMs.

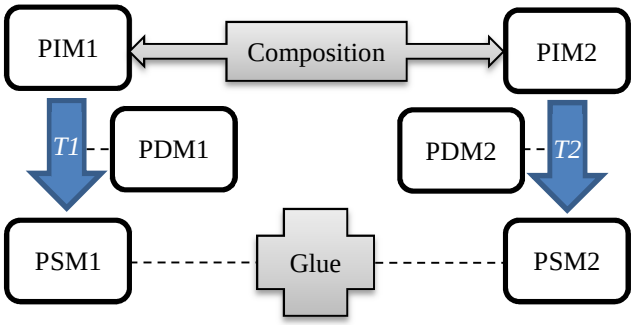


Fig. 2 - Model Driven Architecture for model integration (adapted from [42]) including PDMs.

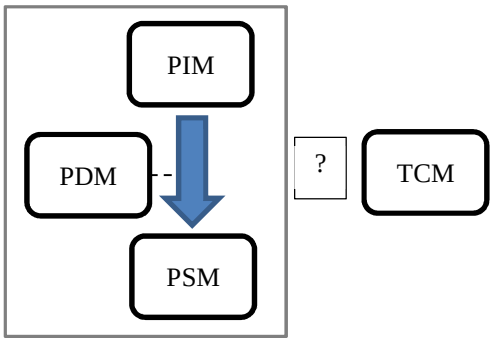


Fig. 3 – Problem to be solved regarding Model Driven Architecture and integration of the Transformation Context Model (TCM) in the global process.

The example of Fig. 2 corresponds to code integration to implement a service (PSM2) with an Enterprise JavaBeans (EJB) execution environment (PDM1), in the code of another service (PSM1) that has the same execution environment (PDM2 = PDM1). This integration specified in the composition of PIM1 and PIM2, includes the scenario description of PIM2 in the scenario description of PIM1. Making thus use of glue code, PSM1 resulting from *T1* is added to PSM2 obtained from *T2*. Glue code represents the technical constraint of PSM2 to identify its EJB via the respective Java Naming and Directory Interface (JNDI). This model integration approach is nevertheless inappropriate, due to the cost of implementing two parallel processes with one that precedes the other. Production variability, already considerable during an information system development [44], is increased by the Software as a Service distribution paradigm [45]. A large number of ready-to-use services may be available – several thousands in certain companies. Suppose that T1 is the main transformation and T2 is the transformation that produces a context for T1. To integrate those services, a very significant number (*n*) of transformations like *T2* along with the required glue code between PSM1 and the resulting *n* PSM2, imply proportional higher development complexity and costs.

Multiple *n* transformations of type *T2* and the respective glue code could be avoided, improving development significantly. Instead, *n* contextual PSM2 models that precede *T1* could be integrated. These *n* PSM2 models are grouped in a global model, identified hereafter as Transformation Contextual Model (TCM), given its contextual role with respect to *T1*. Such required enrichment of a process using a contextual model (Fig. 3), has not been previously defined in the literature.

3.2. Integration of a contextual model into a process

While the TCM represents available data that an expert exploits to influence a process, meta-modelling of the TCM must take into account the production method of the corresponding data. The extension to the MDA approach mechanisms that we propose is therefore a meta-modelling of the context, including useful concepts for the production of contextual data, in order to automate the implementation of context in a transformation. The TCM meta-model describes the production mode selected for the contextual data. As hypothesized in the research question, the data production mode is consistent with an MDA approach, on which platform-independent elements are transformed into platform-specific elements through the description of the platform. This allows integrating the data production method adequately into the process as knowledge. A contextual model must thus be studied at the early stages of system analysis before system architecture activities, to be integrated in a MDA-compliant process. Corresponding knowledge of the transformation input elements permits an expert to formulate rules associating a context with a process. The Contextual Transformation (*CT*) of data into knowledge consists on this set of rules, specified by a process expert, permitting to integrate contextual data in the process.The proposed integration of a contextual TCM results from enriching the PIM – by means of *CT* – before the MDA-compliant transformation *T*. This transformation *T* is applied to the enriched PIM (Fig. 4). A PIM enriched by a contextual model is denoted henceforth PICM (Platform Independent Contextual Model). The two use cases described in sections 4 and 5 show the feasibility of CT design making use of a TCM representing a MDA approach of the selected contextual data production.

It is important to note that the TCM specificity is to include the production mode of contextual data, without using MDA marks, enabling to associate an element of the PIM with contextual data. Additionally, marks defined in the PDM -independently from the PIM- are not a part of the PICM model, while the TCM is a part of the PICM model and associates contextual elements to the PIM. The proposed approach is structured according to systemic modelling [46] by transformations depicted in Fig. 4. Defined rules rely on knowledge about the result of transforming any relationship between two units. Every model is hence defined by a set of units (denoted *_unit) and a set of relationships (denoted *_relationship). Rules are identified with the type of transformation (*CT* or *T*), while specific rules about a unit are denoted *RU* and rules about relationships between units *RR*.

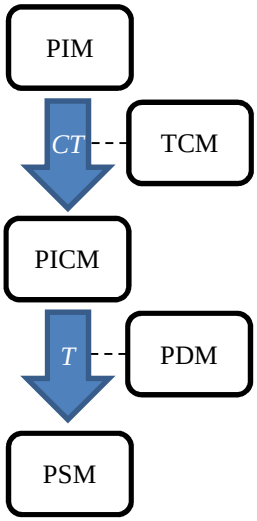


Fig. 4 - Model driven integration of a contextual model into a MDA-compliant process.

Three successive stages constitute the proposed approach:

- ***Contextual CT-Transformation design***, on which mapping between the contextual model and the input of the model transformation are defined. The PICM results from the contextual transform *CT* that maps the PIM with the TCM. A rule defines this mapping to indicate under which criterion a PIM element can be associated to a TCM element (represented by "→"). The transformation of PICM units is defined by:

  PICM_unit = CT(PIM_unit, TCM_unit)

  where *CT* connects PICM_unit and PIM_unit with a mapping link between one PIM_unit and one TCM_unit. Such ***CT-RU*** rule implemented through *CT* means that if a *pim_unit* and a contextual model unit *tcm_unit* can be mapped, then the resulting *picm_unit* is:

  picm_unit = CT(pim_unit, tcm_unit) = (pim_unit → tcm_unit),

  if not,

  picm_unit = CT(pim_unit,{Ø}) = pim_unit.

  Regarding relationships between units (represented by "–"), *CT* defines a PICM_relationship from a PIM_relationship that takes into account a TCM_relationship:

  PICM_relationship = CT(PIM_relationship, TCM_relationship)

  This ***CT-RR*** rule implemented by *CT* for the relationships between units, illustrates in a similar manner as for units that, if relationships exist between a *pim1_unit* and a *pim2_unit*, and a contextual model relationship *tcm1_unit – tcm2_unit* between a *tcm1_unit* mapped with *pim1_unit* and a *tcm2_unit* mapped with *pim2_unit*, then the resulting *picm_relationship* is:

  picm_relationship =
  CT(pim1_unit – pim2_unit, tcm1_unit – tcm2_unit)
  = (pim1_unit – pim2_unit → tcm1_unit – tcm2_unit),

  if not,

  picm_relationship = CT(pim1_unit – pim2_unit, {Ø})
  = pim1_unit – pim2_unit.

- ***T-Transformation design***, on which a MDA-compliant technical constraint and the result model of CT are defined. Model transformation *T* provides a PSM that results from a PICM and a technical constraint specified in the PDM. Transformation *T* for units is defined by:

  PSM_unit = T(PICM_unit, PDM)

  where *T* applies constraints defined in the PDM to a *PICM_unit* giving as result a *PSM_unit*. The same applies to a relation between PSM units and is defined as:

  PSM_relationship = T(PICM_relationship, PDM)

  where *T* applies constraints defined in the PDM to a *PICM_relationship* giving as result a *PSM_relationship*.

  Rules implemented in model transformation *T* for integration by substitution are different than rules for integration by enhancement. Therefore, specializations *ST* (*Substitution Transformation*) and *ET* (*Enhancement Transformation*) of *T* are necessary.

  For units, the ***ST-RU*** rule implemented by *ST* distinguishes among PICM_unit transformations with and without substitution (*pim_unit* by *tcm_unit*) as follows (using the same ***CT-RU*** notation):

  picm_unit = CT(pim_unit, tcm_unit) = (pim_unit → tcm_unit)
  ⇒ psm_unit = ST(tcm_unit, pdm)

  and,

  picm_unit = CT(pim_unit, {Ø}) = pim_unit
  ⇒ psm_unit = ST(pim_unit, pdm)

  In the case of *ET* the ***ET-RU*** rule changes because the process is enhanced (*pim_unit* with *tcm_unit* represented with the sign ⊕) by the contextual model as (using the same ***CT-RU*** notation):

  picm_unit = CT(pim_unit, tcm_unit) = (pim_unit → tcm_unit)
  ⇒ psm_unit = ET(pim_unit ⊕ tcm_unit, pdm),

  and,

  picm_unit = CT(pim_unit,{Ø}) = pim_unit
  ⇒ psm_unit = ET(pim_unit, pdm).

  Concerning relationships between units, the ***ST-RR*** rule implemented by ST discriminates PICM transformations with and without substitution as follows (using the same ***CT-RR*** notation).

  picm_relationship =
  CT(pim1_unit – pim2_unit, tcm1_unit – tcm2_unit)
  = (pim1_unit – pim2_unit → tcm1_unit – tcm2_unit)
  ⇒ psm_relationship = ST(tcm1_unit – tcm2_unit, pdm),

  and,

  picm_relationship = CT(pim1_unit – pim2_unit, {Ø})
  = pim1_unit – pim2_unit
  ⇒ psm_relationship = ST(pim1_unit – pim2_unit, pdm).

  For *ET* the ***ET-RR*** rule applies specifically to process enhancement using a contextual model as (using the same ***CT-RR*** notation):

  picm_relationship =
  CT(pim1_unit – pim2_unit, tcm1_unit – tcm2_unit)
  = (pim1_unit – pim2_unit → tcm1_unit – tcm2_unit)
  ⇒ psm_relationship =
  ET(pim1_unit – pim2_unit ⊕ tcm1_unit – tcm2_unit, pdm),

  and,

  picm_relationship = CT(pim1_unit – pim2_unit, {Ø})
  = pim1_unit – pim2_unit
  ⇒ psm_relationship = ET(pim1_unit – pim2_unit, pdm).

  These four rules – ***ST-RU***, ***ET-RU***, ***ST-RR***, and ***ET-RR*** – emphasize the need to apply constraints defined in the PDM to a unit or a relationship of the PICM, specified in the process as substitution or enhancement.

- ***CT and T Transformations running*** to validate the proposed approach's feasibility. The next two sections describe in detail how

transformation *CT*, and the specializations of *T*, *ST* and *ET*, implement the described approach when applied to a code generation process and to an image classification process.

Table 1 summarizes the previously defined notations for the implemented transformations and rules.

Table 1 - Meaning of the Acronyms for Transformations and Rules

Transformation		Transformation Meaning	Implemented Rule	Rule Meaning
T	***CT***	Contextual transformation	***CT-RU***	Rule for unit
			CT-RR	Rule for relationship
	ST	Substitution transformation	***ST-RU***	Rule for unit
			ST-RR	Rule for relationship
	ET	Enhancement transformation	***ET-RU***	Rule for unit
			ET-RR	Rule for relationship

The next two sections present two use cases on which the pertinence of the proposed approach is examined. In each case we consider, the feasibility of defining a suitable contextual model, how can automatic platform specific programs be generated, and how to integrate these programs to the respective process.

4. Integration of a contextual model into a process with substitution: the case of code generation

This section illustrates the integration of a contextual model into a business process of code generation, formed by a large variety of activity sequences in a company. According to the proposed approach in section 3, ready-to-use services available in a company's information system are the contextual data of the code generation process. Moreover, the link between business activities and these ready-to-use services by means of mapping permits to associate contextual data, i.e. an applied service, to one activity. Every business activity associated or not to contextual data, is then implement by code representing calls to specified applied services. These calls are integrated when mapping is required or programmed if that is not the case. Generated code is constrained by a technical environment like Java Enterprise Edition (JEE). Table 2 summarizes the code generation process models that are modified when a contextual model with substitution is integrated.

Table 2 – Process models modified by a contextual model with substitution.

Model	Integration of a contextual model into a code generation process
PIM	Model of activities
TCM	Model of existing applicative services of the information system
PICM	Model of activities enhanced by existing services
PDM	Model of JEE environment for an applicative service
PSM	Model of generated code

The *CT* transformation permits to associate a contextual applied services model – ready to be substituted through the TCM modelling – to a PIM activities model. This association forms pairs of one activity and one existing applied service (PICM). Coupling is conditioned by a mapping, implemented in *CT*, between the PIM activity name and the name of the activity supported by the ready-to-use service. Additionally, *ST* transforms an activities model in a code of applied services (PSM), enabling the substitution of available application services of the information system (PICM). Such coding is constrained by a technical environment (PDM).

4.1. Contextual CT-Transformation design

The *CT* transform allows to associate external services modelled in a TCM to a PIM business process, composed by progressive sequences of business activities.

4.1.1. Unit CT-Transformation

Mapping between one PIM unit and one TCM unit is defined by the name of the PIM's business activity and the name of the activity carried out by the TCM's application ready-to-use service, which must be identical. To illustrate the contextual model of a code generation process two applied services are defined: "*cASReadAProduct*" to implement the "*Read a product*" activity; and "*cASCreateAnOrderOfAProduct*" to implement the "*Create an order of a product*" activity. The relation between business activities and ready-to-use application services defined by the ***CT-RU*** rule (see section 3.2) is underlined in Fig. 5. Applied services and their respective interface are represented here by operations in UML classes (considering a business activity like a use case analogously as in a context UML diagram).

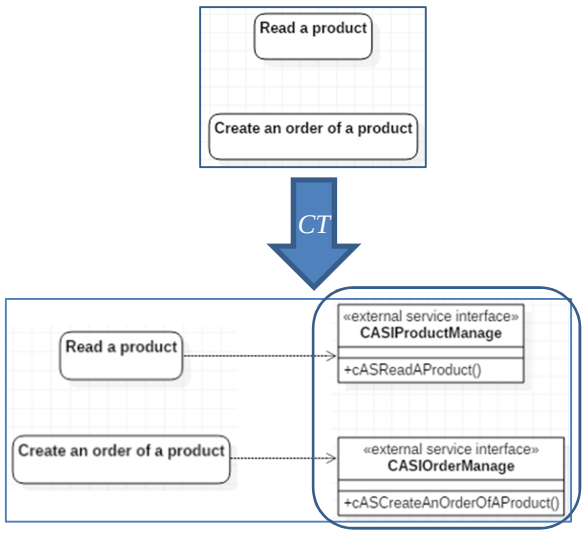


Fig. 5 - Illustration of the unit contextual CT-Transformation for the code generation process.

Variable data production is reflected in this first illustration of feasibility by the development method with which the external services are produced (function implementation, business activity production, programming language, etc.). Moreover, variable data production is represented by the fact that these services could be replaced in the TCM by application data provided by these services (the physical data *Product* and *Order* in this case). These application data are characterized by the generated logical data and the selected technical environment (for example, relational database management system or object oriented). For the TCM illustrating context by replacement, the service production mode is a conventional service-oriented architecture framework. In this case, the service encapsulated in an interface, carries out a business activity and is deployed on a technical environment.

Note that the PICM's independence property with respect to the platform defined by the PDM is maintained, since the mapping is independent of the technical environment used for the deployment of these two external services (a JEE development in this case).

The PIM meta-model concept for a unit and its attribute (described by one instance excerpted from Fig. 5) are defined as:

- "BusinessActivity":
  - "name" (*Read a product*).

The TCM meta-model contains a description of an external service interface that encapsulates a ready-to-use application service (instances are extracted from Fig. 5):

- "ExternalApplicativeService" describes an external available service:
  - "name" represents the name of the external available service (*cASReadAProduct*).
  - "externalServiceInterface" describes the interface that encapsulates the application service (*CASIProductManage*).
  - "supportedActivityName" refers to the name of the business activity supported by the external service (*Read a product*).
  - "deploymentEnvironment" specifies the technical environment on which the service is deployed (*JEE*).

The resulting PICM meta-model of *CT* for a unit is outlined in Fig. 6.

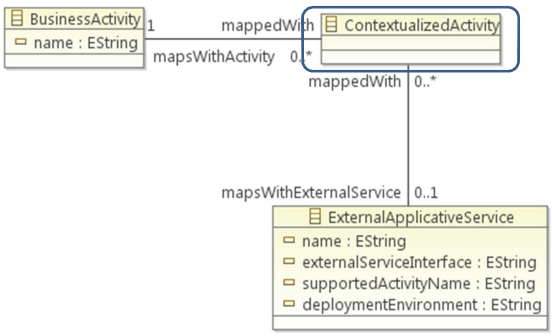


Fig. 6 - PICM meta-model (enclosed) of the code generation case including the mapping between PIM (BusinessActivity) and TCM (ExternalApplicativeService) meta-models applied to a unit.

The PICM and its attributes are instantiated from the example in Fig. 5:

- "ContextualizedActivity" describes an activity enriched by the contextual model of existing application services:
  - "mapsWithActivity" indicates the enriched business activity (*Read a product*)
  - "mapsWithExternalService" indicates the external service mapped with the activity from the business activity name and the business activity name supported by the external application service (*cASReadAProduct*). This link does not exist when external services can't be mapped to the business activity (cf. rule ***CT-RU*** in 3.2)

4.1.2. Unit Relationship CT-Transformation

Concerning relations between units, a PIM relationship is a temporal sequence of business activities. The rule ***CT-RR*** (see 3.2) becomes thus for the code generation process: if an activity A1 precedes an activity A2 and if the service cAS1 mapped with A1 precedes in an external orchestration the service cAS2 mapped with A2, then this orchestration enriches the sequence.

CT({A1 – A2}, {caS1 – cAS2}) = {A1 – A2} → {cAS1 – cAS2}.

The PIM is represented by a control process consisting of an ordered sequence of activities "*Read a product*" and "*Create an order of a product*". On the other hand, the TCM is represented by an ordered sequence, or orchestration, of the "*cASReadAProduct*" and "*cASCreateAnOrderOfAProduct*" services. The instantiation of the ***CT-RR*** rule is such that:

{Read a product → cASReadAProduct =
CT(Read a product, cASReadAProduct)} ∧
{Create an order of a product → cASCreateAnOrderOfAProduct =
CT(Create an order of a product, cASCreateAnOrderOfAProduct)}
⇒ CT({Read a product – Create an order of a product},
{cASReadAProduct – cASCreateAnOrderOfAProduct}) =
{Read a product – Create an order of a product} →
{cASReadAProduct – cASCreateAnOrderOfAProduct}

The result is therefore an integration of the application services orchestration resulting from business activity mapping. A simplified implementation of the transformation is illustrated in Fig. 7.

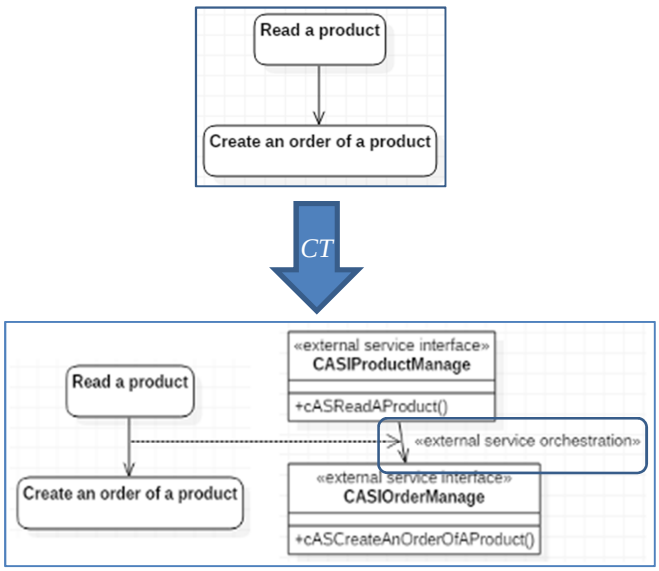


Fig. 7 - Illustration of the contextual CT-Transformation (enclosed) of a relationship for the code generation process.

The PICM meta-model for a relationship between units is outlined in Fig. 8 (instances are extracted from Fig. 7). Mapping between external orchestration of services and sequence of business activities is based on the ***CT-RU*** rule.

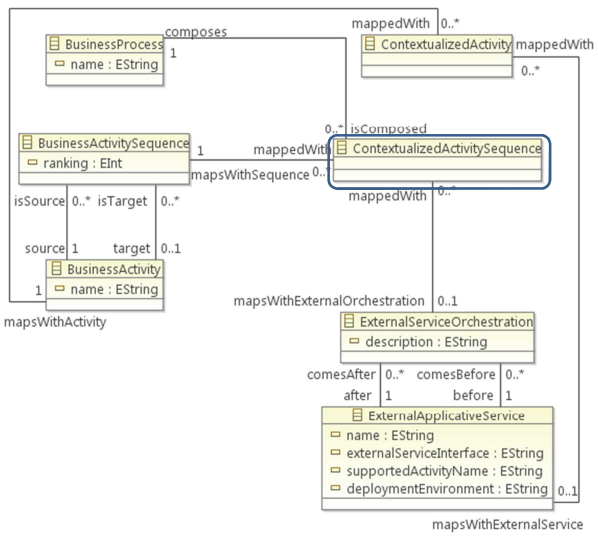


Fig. 8 - PICM meta-model (enclosed) of the code generation case including the mapping between PIM and TCM meta-models applied to a unit relationship.

Consequently, the PIM meta-model concepts in Fig. 8, represented as relationships between units and attributes (described with instances from Fig. 7) are defined as:

- "BusinessProcess" characterizes a response of a company to a customer's request:
  - "name" (*Order process*).
- "BusinessActivitySequence" represents a time sequence of two business activities that make up a process (*Read a product → Create an order of a product*):
  - "composes" identifies the process to which the sequence belongs (*Order process*).
  - "source" designates the business activity that initiates the sequence (*Read a product*).
  - "target" specifies the business activity completing the sequence (*Create an order of a product*). This link to the target activity is null when the process consists of only one business activity.
  - "ranking" specifies the number of the sequence in the process (*1*: the *1*st sequence of the process).

The TCM of Fig. 8 describes an orchestration of services based on the following concept (instances are from Fig. 7):

- "ExternalServiceOrchestration" describes an external sequence of calls to external services:
  - "description" designates the sequence of external services (*cASReadAProduct – cASCreateAnOrderOfAProduct";*).
  - "before" refers to the first service of the sequence (*cASReadAProduct*).
  - "after" refers to the second service of the sequence (*cASCreateAnOrderOfAProduct*).

For relations between units, the PICM concept (outlined in Fig.8) and its attributes (with instances extracted from Fig. 7) are:

- "ContextualizedActivitySequence" describes a sequence of activities enriched by the contextual model of existing orchestrations of services:
  - "mapsWithSequence" designates the business activity sequence (*Read a product – Create an order of a product*).
  - "mapsWithExternalOrchestration" identifies the sequence of two calls to external application services (*cASReadAProduct – cASCreateAnOrderOfAProduct*). This link does not exist when there isn't a sequence of external services to be mapped with the sequence of business activities.
  - "composes" provides the link of the business process that contains the sequences of activities (*Order process*).

4.2. ST- Transformation design

When a contextual model is integrated with substitution, the *ST* MDA-compliant transformation is no longer applied to the PIM, but to the PICM. The PDM specifies the rules to be technically implemented to integrate services developed with a Java runtime environment or to program new services. These rules complete the definition of a three-layer application architecture synchronized by an application server:

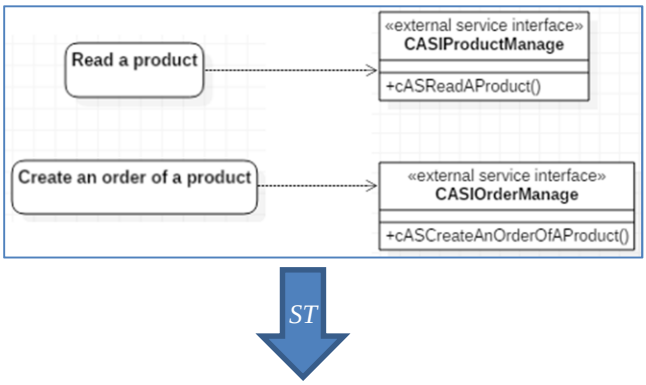
- The "data access" layer which allows manipulating data with SCRUD-type operations (Search, Create, Read, Update, Delete).
- The "service" layer where each service supports a business activity in this particular case and uses components of the "data access" layer.
- The "presentation" layer that represents the Human Machine Interface (HMI) and uses components of the "service" layer depending on the user's activity.

4.2.1. Unit ST-Transformation

For units, *ST* implements the rules to transform a business activity into an application service (to be developed or ready to be substituted by code generation of a service). Application services are declared in a Java interface and coded (if development is necessary) in a Java class with the appropriate formalism. While the PSM resulting from *ST* is associated with the "service" layer of the system (Fig. 9), the ***ST-RU*** rule (see section 3.2) is applied to units individually substituted by a TCM unit (resulting code is represented by a *code* function):

ST(cASReadAProduct, {JavaInterface,JavaCode}) =
{*code*(cASReadAProduct)}
ST(cASCreateAnOrderOfAProduct, {JavaInterface,JavaCode}) =
{*code*(cASCreateAnOrderOfAProduct)}

Additionally, the PSM is only composed by Java interfaces code obtained from the PICM transformation (code of the Java classes implementing the interfaces in an external system). A rule implemented in *ST* checks the consistency between the external service deployment environment and the execution environment of the system to be coded. In the examined case, the hypothesis of a JEE environment for external services allows to integrate them into the generated code.




```
// service interfaces (used existing interfaces)
public interface CASIProductManage
{
  public Product cASReadAProduct
     (String productName);
}
public interface CASIOrderManage
{
  public Order cASCreateAnOrderOfAProduct
     (Date dateOrderProduct,Product product);
}
```

Fig. 9 - Example of the ST-Transformation of a unit for the code generation process, including the PSM and the PICM.

PICM and PSM meta-models defined for the units and relations between units resulting from *ST* (Fig. 10) are explained below.

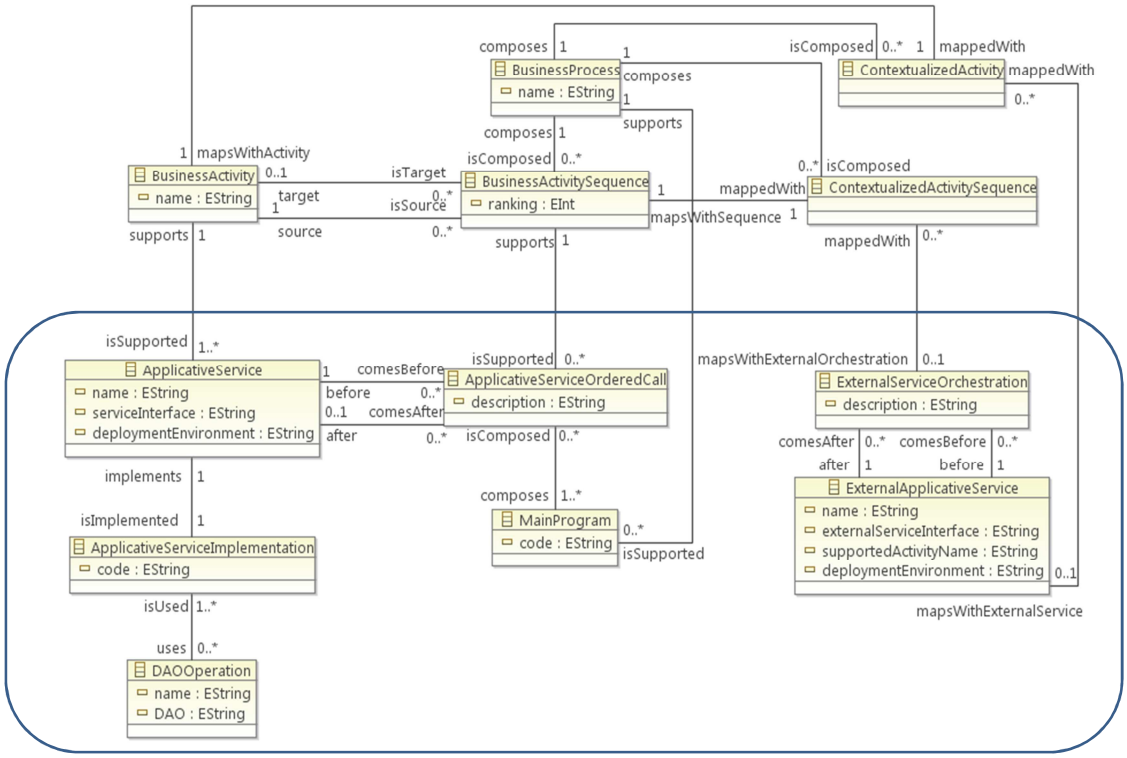


Fig. 10 - PSM meta-model (enclosed) of the code generation case, associated to the PICM meta-models (unit and relationship), satisfying the PDM meta-model.

The PDM meta-model specifying the technical constraints of an *ST* transformation, as described for the units, is outlined in Fig. 11. Concepts specific to the definition of architecture proposed in the introduction to section 4.2 – "ApplicationServer", "ServiceLayer", and "DataAccessLayer" –, are useful for units and complete the meta-model.

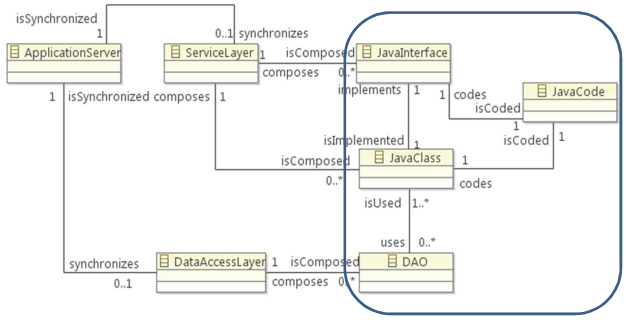


Fig. 11 - PDM meta-model (enclosed) of the code generation illustration with a MDA approach applied to a unit.

Consequently, the PDM meta-model concepts and attributes are defined as:

- "JavaCode" represents Java code associated with code generation.
- "JavaInterface" represents a Java interface of the service layer (the "ServiceLayer" concept complements the PDM meta-model):
  - "isCoded" associates the interface with its Java code.
- "JavaClass" represents a Java class of the service layer ("ServiceLayer" concept):
  - "isCoded" associates the interface to its Java code.
  - "implements" refers to the Java interface implemented in the class.
- "DAO" represents a Data Access Object of the "data access" layer (concept "DataAccessLayer" that completes the PDM meta-model):
  - "isUsed" refers to the Java class that makes use of this DAO.

PSM meta-model is completed for service coding (not instantiated in Fig. 9, but this case is rather common in today's information technology) using the MDA approach for code generation):

- "ApplicativeService" represents the signature of the application service, defined as an operation of the Java interface encapsulating the signature of the service, as recommended in the PDM:
  - "supports" designates the business activity of the PIM transformed by this service.
  - "name" provides the name of the application service.
  - "serviceInterface" is the name of the interface containing the service signature.
  - "deploymentEnvironment" specifies the technical constraint of the PDM.
- "ApplicativeServiceImplementation" represents the service implementation in a PDM-compliant manner:
  - "implements" refers to the implemented Java interface as specified in the PDM.
  - "code" represents the code of the interface operation.
- "DAOOperation" represents the operations for accessing PDM-compliant data:
  - "isUsed" refers to the code of the service using a call to the DAO operation.
  - "name" indicates the name of the operation.
  - "DAO" is the name of the data access object that encapsulates the data access operation.

4.2.2. Unit Relationship ST-Transformation

For relations between units, the *ST* transformation targets the "presentation" layer. Each sequence of two business activities is transformed by *ST* into a coded sequence of calls to application services (external or internal). Furthermore, *ST* satisfies a technical constraint to execute a Java program. The Java program is designed on the "presentation" layer, which communicates with the "service" layer previously generated from units of the PICM.

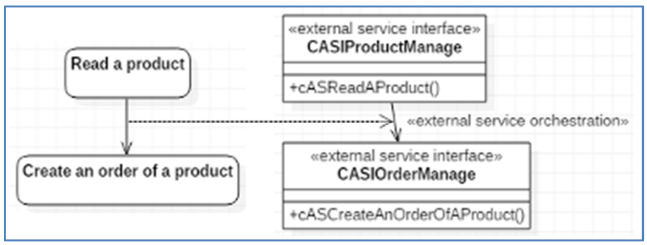
The ***ST-RR*** rule (see section 3.2) is applied to a single relationship that is mapped with a TCM orchestration. Consequently, the ***ST-RR*** rule becomes: if an activity A1 precedes an activity A2, then the service cAS1 mapped with A1 precedes, with an external service orchestration defines in the PDM, the service cAS2 mapped with A2:

if (CT({A1 – A2}, {cAS1 – cAS2}) = {A1 – A2} → {cAS1 – cAS2})
then
ST({cAS1 – cAS2}, ExternalServiceOrchestration) = {cAS1;cAS2;}
else
checking of input and output parameters of cAS1 and cAS2 services to design their orchestration
endif

The ***ST-RR*** rule is instantiated for the sequence transformation, from the consultation activity of a product to the order creation activity, constrained by a Java service orchestration (resulting code is represented by a *code* function as previously for ST illustration for units):

ST({cASReadAProduct – cASCreateAnOrderOfAProduct}, Java:ExternalServiceOrchestration) = {*code*(cASReadAProduct;cASCreateAnOrderOfAProduct;)}

An implementation of the ST transformation is illustrated in Fig. 12. The sequence of activities with the orchestration of services (outlined on the top) mapped to it, "{*Read a product – Create an order of a product*} → "{*cASReadAProduct;cASCreateAnOrderOfAProduct*}", is the input of the *ST* transformation. At the output of *ST*, the coded sequence is modeled by calling these two services in the *main*, as well as the methods "*presentation_readProduct()*" and "*presentation_orderProduct()*".




```
// get existing service interfaces
public static CASIProductManage
getManageProduct()
{
InitialContext ctx;
CASIProductManage manageProduct = null;
ctx = new InitialContext();
manageProduct = (CASIProductManage)
  ctx.lookup(CASIProductManage.JNDI_NAME);
```

```
return manageProduct;
}
public static CASIOrderManage getManageOrder ()
{
InitialContext ctx;
ASIOrderManage manageOrder = null;
ctx = new InitialContext();
manageOrder = (CASIOrderManage)
  ctx.lookup(CASIOrderManage.JNDI_NAME);
return manageOrder;
}
// methods associated to services
public static Product presentation_readProduct()
{
return getManageProduct().
  cASReadAProduct ("product_name");
}
public static void presentation_orderProduct
  (product : Product)
{
Date date = new Date(System.currentTimeMillis());
getManageOrder().cASCreateAnOrderOfAProduct
  (date, product);
}
//Java program
public static void main(String[] args)
{
Product product = presentation_readProduct();
presentation_orderProduct (product);
}
```

Fig. 12 - Illustration of the ST-Transformation of relationship for the code generation process.

The PDM meta-model specifying technical constraints of the *ST* transformation as described above for the relations between units, is outlined in Fig. 13. Concepts of the three-layer architecture – "ApplicationServer", "PresentationLayer", and "ServiceLayer" – complete the PDM meta-model.

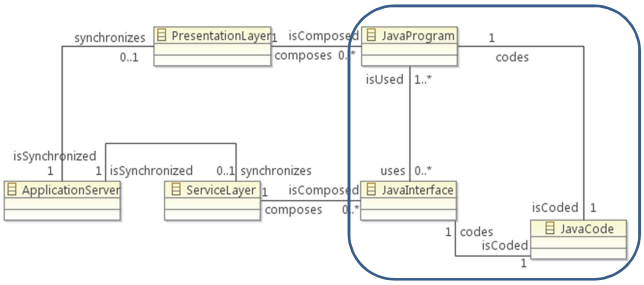


Fig. 13 - PDM meta-model (enclosed) of the code generation illustration with a MDA approach applied to a unit relationship.

The PDM meta-model is complemented by a concept for relations between units:

- "JavaProgram" represents the Java program on the "presentation" layer (being "PresentationLayer" a concept that completes the PDM meta-model) to which the system user has access:
  - "isCoded" associates the program to its Java code.
  - "uses" indicates the Java interfaces used by the program.

The resulting PSM meta-model (Fig. 10) concepts (from the MDA-compliant ST transformation) and attributes (described by one instance of Fig. 12) stipulated for these unit relationships are:

- "ApplicationServiceOrderedCall" designates two successive calls to application service interfaces:
  - "description" (*aSReadAProduct; asCreateAnOrderOfAProduct;*);
  - "before" specifies the first called applicative service (*aSReadAProduct*).
  - "after" specifies the second called applicative service (*asCreateAnOrderOfAProduct*). When the second service is null then the process is composed of only one business activity and therefore supported by a single application service.
- "MainProgram" characterizes the support of a business process in accordance with the PDM:
  - "code" (*public static void main (String[] args) {Product product = presentation_readProduct(); presentation_orderProduct (product);}*).
  - "supports" refers to a supported business process (*Order process*).
  - "isComposed" refers to sequences of calls to application services (*aSReadAProduct; aSCreateAnOrderOfAProduct;*)

The *ST* transformation implements rules for assembling call sequences to application services that support a business process. Such assembling follows the previous consistency rule between the service call structure and the activity structure in a particular sequence. Also, generation of code linked to an external service call or the orchestration of external services, are taken into account by the *ST* transformation.

The specific addition to the PICM at the input of the ST transformation is the use of JNDIs of external service interfaces, namely "*CASIProductManage*" and "*CASIOrderManage*" (cf. enclosed parts of Fig. 12) instead of using service interfaces developed in the system.

4.3. CT- and ST-Transformations Execution

The execution of these transformations is linked to the adaptation of meta-models to a development process within the framework of a real enterprise. Generating code from the analysis of a system is split in two sub-processes. For instance, in the Enterprise Architecture for Unified Process approach, each sub-process integrates a specific contextual model [47]:

- Functional architectural model of a system according to a functional analysis of the system and a contextual model representing the functional EA of the information system.
- Application architecture and code model based on the functional architecture of the system previously modeled with a ready for use contextual model of services (external or internal to the system).

The *ST* transformation for the code generation process can be implemented. One application is, for example, a rule generating a component of the service layer. Rules specific to the information system in which the system is developed are satisfied by the creation of a service

layer component. One of the rules for creating such a component is for instance:
ServiceLayerComponentDesignRule. One applicative service (*AS*) results from the transformation of one business activity (*BA*). An applicative service interface providing *AS* results from the transformation of the business process that contains *BA*.

This rule is implemented by the operational-QVT (Query / View / Transformation) [48] code in Fig. 14, which is compliant with the PSM meta-modeling in Fig. 13.

```
//creation of a service of a component of the service layer
mapping BusinessActivity::createApplicativeService() :
ApplicativeService
{
        name := "aS" + self.name.replace(" ","");
        serviceInterface := "ASI" + self.mappedWith.composes.name.
    replace(" ","");
  deploymentEnvironment := "JEE";
        result.supports := self;
}
```

Fig. 14 - Illustration of the ST-transformation coding of the *ServiceLayerComponentDesignRule* for the code generation process.

Coding of the rule *ServiceLayerComponentDesignRule* is completed by coding of rules deferring JEE technical constraints. The contextual model linked to this code generation process allows capturing the variability related to the model of application services. It can replace code generation of a service to be developed. This integration is shown in the following section with the application of a contextual model to the image classification process.

5. Integration of a contextual model into a process with enhancement: the case of supervised classification

Image classification is another case on which context variability occurs [49]. In image classification it is assumed that visual image features can be uniquely associated to statistically learned and semantically meaningful classes. Context variability is produced by the multiple possible image content descriptors. An image assigned to a visual class is quantitatively represented by numerous descriptors composed of several hundred elements. These are basically low (contours and texture), medium (regions), and high (values analysis) level image features automatically calculated.

Besides the need to handle millions of images and thousands of classes, process complexity arises from the fact that image features differ in size, classification performance, and reliability. As a consequence, different performance results are obtained depending on the corpus. Image features also relate differently to supervised (learnt examples) and unsupervised (grouping criteria) classification. For these reasons, numerous approaches have been implemented for image classification in large databases, with variable sets of descriptors and classes [50] [51] [52] [53].

Image classification is particularly difficult given that knowledge owned by experts of the application domain is required. Data production variability increases in this case because experts need to combine optimally different image descriptors to improve classification results [54]. A descriptor is a low level mathematical characterization of visual image content, for example a geometrical moment of a set of pixels [55]. Applied image content low and medium descriptors to illustrate the classification process in this paper are:

- Zernike descriptor (49 elements from $\mathbb{C}$, i.e. a vector in $\mathbb{C}^{49}$), composed by orthogonal complex moments of an image that are invariant to rotation and robust to noise [56].
- Angular partition descriptor (32 elements from $\mathbb{N}$, i.e. a vector in $\mathbb{N}^{32}$), formed by 32 elements that represent the distribution of segmented edges in a symmetric angular partition of the image [57].
- Angular radial transform descriptor (25 elements from $\mathbb{C}$, i.e. a vector in $\mathbb{C}^{25}$), calculated applying an angular radial transformation that projects image gray levels on an orthogonal radial space [58].

Although descriptors have different representations and complementary information, a ranking of descriptors is necessary before classification. Then, a classification process could be learnt based on selected descriptor elements, instead of the whole descriptors set. Consequently, evaluating the quality of descriptors' elements extends the classification process from a predefined amount of descriptor elements, to a reduced and selected set of descriptor elements.

To examine the pertinence of MDE, the contextual model integration approach presented in section III is applied to the process of image classification. It is assumed that previously learnt image classes, independently of the learning algorithm, are available for image classification. These learnt image classes enrich the process. Without such enrichment, a *MDA-compliant* transformation would just provide the closest images to a given query image, as post-classification result. Otherwise, the design of transforms *CT* and *ET* only targets units, without defining relationships between images. Since the integration of a contextual model outputs a classified image, a model including image content descriptors' elements is transformed into a model of a classified image. Several process models are modified by the integration of a contextual model into a process with enhancement in the case of image classification (Table 3).

Table 3 – Process models modified by a contextual model with enhancement.

Model	Integration of a contextual model into an image classification process
PIM	Model of image
TCM	Model of characteristic image class
PICM	Model of image enhanced by characteristic image class
PDM	Model of Euclidean distance
PSM	Model of a classified image

The *CT* transformation allows associating a contextual model of image class to an image (PIM), making use of characteristic images of a class (TCM). Pairs defined in this manner relate a query image to an image class (PICM).

The design condition of these pairs is a mapping between the name of the image collection to which the image to be classified belongs and the name of the image collection to which the classes are associated. This condition is implemented in *CT*. Alternatively, the *ET* transformation

targets an image enriched by mapped image classes (PICM) and results into a classified image (PSM). Classification is the assessment of the closest class in the sense of a Euclidean distance (PDM), between certain elements of the descriptors quantifying the image and the same elements quantifying the characteristic image of each class. Selected descriptor elements and calculation of the smallest distance between the image and all the characteristic images, are coded in the *ET* transformation.

5.1. Contextual CT-Transformation Design

To demonstrate the feasibility of the proposed approach, exhaustive testing was conducted using the ALOI (Amsterdam Library of Object Images) collection [59]. For instance, using a schematic representation of the *CT* transformation (Fig. 15), query image "*_C36_l6c2*" is associated to classes "*C36*" and "*C461*" according to a subset of learnt images:

$$CT(\_C36\_l6c2,C36) = \_C36\_l6c2 \rightarrow C36$$

$$CT(\_C36\_l6c2,C461) = \_C36\_l6c2 \rightarrow C461$$

With respect to the code generation case, the input image "*_C36_l6c2*" is equivalent to a business activity. The contextual model, which in the previous case consisted on existing application services for code generation, is represented here by classes illustrated with a picture of a characteristic image: a cup (C36) and a roll of paper (C461). The two image-class oriented associations indicate that these classes constitute the contextual image data because they belong to the same corpus. Such association to the same corpus specifies the mapping between an element of the PIM and one or more elements of the TCM. Although for simplicity reasons the mapping implemented in *CT* permits to select image classes from ALOI, TCM can also include other image databases.

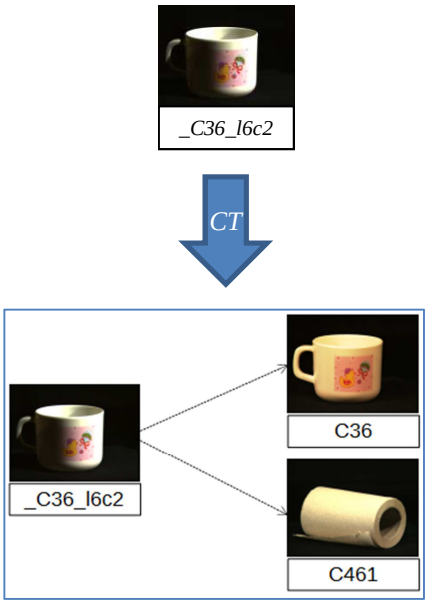


Fig. 15 - Illustration of the contextual CT-Transformation for the image classification process.

A description of an image is provided by the PIM meta-model. It is defined based on the content of image descriptors (Fig. 16). As indicated in the introduction of section 5, quantitative image content description is defined by vectors in $\mathbb{N}^n$, $\mathbb{R}^m$, or $\mathbb{C}^p$.

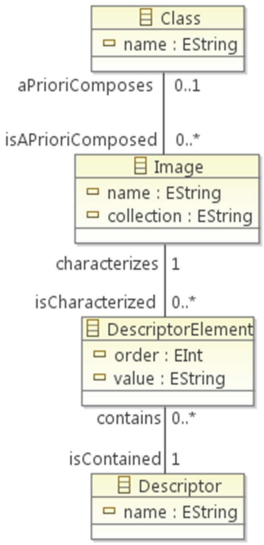


Fig. 16 - PIM meta-model of the image classification illustration.

Consequently, the PIM meta-model concepts and attributes (described by the example of one instance) are defined as:

- "Image":
  - "name" (*_C36_l6c2*).
  - "collection" (*ALOI*).
- "Descriptor":
  - "name" (*Zernike*).
- "DescriptorElement", consisting of:
  - "order" of the descriptor element (10^{th}).
  - "value" of the element (*-5656.085-1218.4637i*).
  - "isContained", by the given descriptor (*Zernike*).
  - "characterizes", denoting the image represented by the element of the descriptor (*_C36_l6c2*).
- "Class"
  - "name" (*C36*)
  - "isAPrioriComposed" (*_C36_l6c2*).

Alternatively, an a priori classification done by the expert is extended via the "aPrioriComposes". It refers to the class that a priori includes content of images described in the classification corpus. Although the "aPrioriComposes" association permits to calculate a classification error rate, it is not computed during image classification that results in a posteriori composition.

Data production variability is illustrated in this second feasibility example by the static evaluation method of the selected classes for pattern recognition. Furthermore, data production variability could be extended whenever the previous evaluation is completed by the assessment of conditional probability of a class knowing another class (for example, a class representing a particular shape in an image, knowing the class representing another shape that appears associated to the previous one in some sets of images). For the TCM exemplifying context by enrichment, the class production mode is characterized by minimum and maximum values, as well as the mathematical expectation of each descriptor's elements. Previous statistical scores are then completed by the rank of each element and descriptor in the classification result, corresponding to their intrinsic quality (defined to be better if the minimum and maximum are comparatively closer).

Available classes are described by the TCM meta-model (see Fig. 17). For the image classification illustration, a set of ranked descriptors as well as a set of ranked descriptors' elements complete the TCM. A training phase allows exhaustive search for the best descriptors and for the best descriptors' elements, with respect to classification performance. Boundary conditions to delimitate image classes of similar data are applied [60]. Different data are generated whenever classes' definition and/or training are modified to suit other classification requirements.

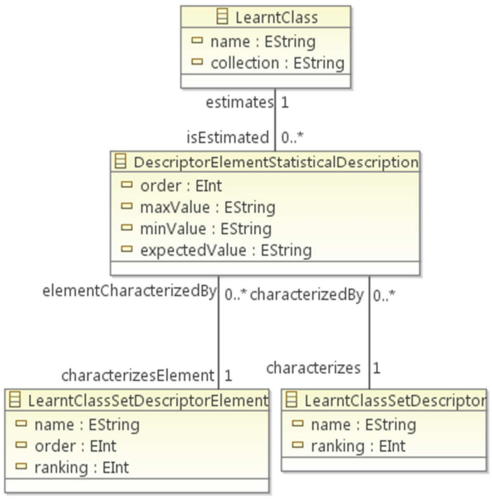


Fig. 17 - TCM meta-model of the image classification process.

The TCM meta-model concepts and attributes (each described by one instance example) are defined as:

- "LearntClass" representing a class identified according to statistical learning:
  - "name" (*C36*).
  - "collection" (*ALOI*).
- "LearntClassSetDescriptor" representing a descriptor, associated to the learnt class set, identified according to statistical learning:
  - "name" (*Zernike*).
  - "ranking" of the descriptor (1^{st}).
- "LearntClassSetDescriptorElement", representing a descriptor element, associated to the learnt class set, identified according to statistical learning:
  - "name", representing the descriptor associated to the element (*Zernike*).
  - "order" of the descriptor element having a value between 1 and *n*, when the amount of descriptor elements is *n* (10^{th}).
  - "ranking", a descriptor element rank of descriptor (64^{th}).
- "DescriptorElementStatisticalDescription", designating the statistical description of a descriptor element, associated to one class and characterizing an element of a descriptor:
  - "order" of the descriptor element having a value between 1 and *n*, when the amount of descriptor elements is *n* (10^{th}).
  - "maxValue" is the maximum value of an element of a descriptor for a set of images belonging to the same class (*8511.2867 + 2374.859i*).
  - "minValue" is the minimum value of an element of a descriptor for a set of images belonging to the same class (*-6017.3437 - 1097.7585i*).
  - "expectedValue" is the average of the values of an element of a descriptor for a set of images belonging to the same class (*1497.495359 + 396.052725i*).
  - "estimates" refers to the learnt class (*C36*).
  - "characterizes" indicates the descriptor associated to the learning result (*Zernike*).
  - "characterizesElement" indicates the descriptor element associated to the learning result (10^{th}).

The PICM resulting from the *CT* transformation is based on mapping between an image and an image class (see ***CT-RU***), respectively excerpted and learnt from the same collection of images (mapping is outlined Fig. 18).

The new concept resulting from the mapping is:

- "ContextualizedImage" representing an association class between the image to classify and one class learnt with an excerpt of the same image collection:
  - "mapsWithImage" refers to the mapped image (*_C36_l6c2*).
  - "mapsWithClasss" refers to the mapped learnt class (*C36*).

CT implements the creation of the "ContextualizedImage" based on the mapping between the collections of the image and of one class, if exists, and the associated concepts of the PIM and of the TCM.

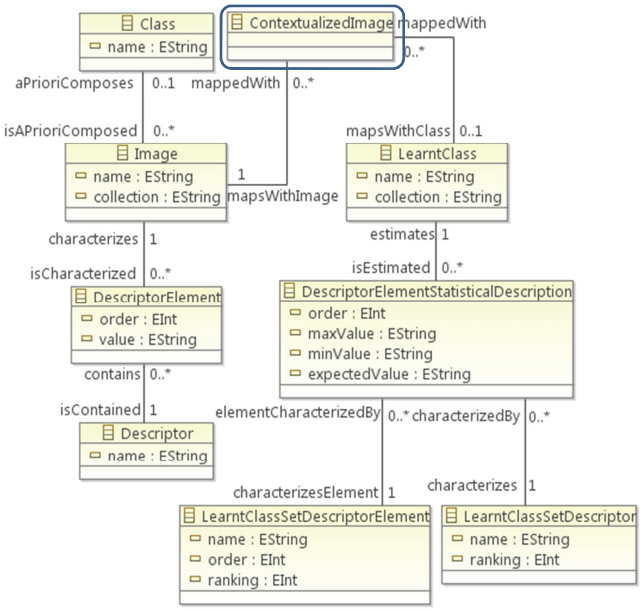
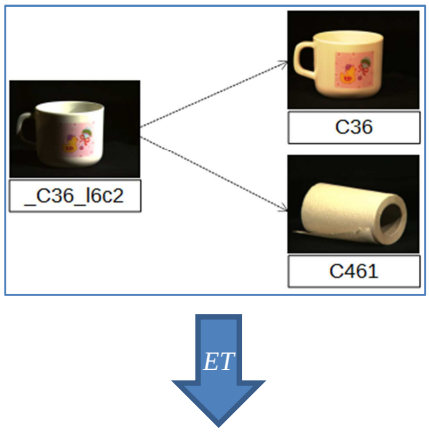


Fig. 18 - PICM meta-model (enclosed) of the image classification process, including the mapping between PIM and TCM meta-models.

5.2. ET-Transformation Design

The *ET* transformation containing the application of the ***ET-RU*** rule with the previous input PICM and a PDM constraint defined by a Euclidean distance is illustrated in Fig. 19. Each line shows the result of one classification process step.




```
Classification of the image _C36_l6c2 (class C36) impossible
Classification of the image _C36_l6c3 (class C36) failed in dans C461 with the Estimate the ART_5x5_gray descriptor activity
Classification of the image _C36_l7c1 successful in C36 with the Estimate the AP_8x3_gray descriptor activity
Classification of the image _C36_l7c2 successful in C36 with the Estimate the AP_8x3_gray descriptor activity
Classification of the image _C36_l7c3 successful in C36 with the Estimate the Zernike descriptor activity
Classification of the image _C36_l8c1 successful in C36 with the Estimate the Zernike descriptor activity
Classification of the image _C36_l8c2 successful in C36 with the Estimate the Zernike descriptor activity
```

Fig. 19 - Three possible results – impossible, failed, and successful – of the proposed ET-Transformation example for the image classification process.

The first line of Fig. 19 indicates that the *_C36_l6c2* image was not classified by any of the three defined descriptors. An unattainable classification like this one means that the decidability threshold is too high. This line represents an instance of the PSM on which "Distance" "evaluates" a pair composed only by an "Image" and no "LearntClass" (*_C36_l6c2*, *null*) with a *true* "classification". *ET* does not return any "LearntClass" for this "Image":

ET(_C36_l6c2 ⊕ C36, EuclideanDistance) = null

ET(_C36_l6c2 ⊕ C461, EuclideanDistance) = null

The second line shows a classification error, because the *_C36_l6c3* image was incorrectly classified in the *C461* class by the *ART_5x5_gray* (angular radial transform) descriptor. This classification error means that the classification is only decidable for the *ART_5x5_gray* descriptor (the lowest quality descriptor), although the obtained result is wrong. This line represents an instance of the PSM on which "Distance" "evaluates" a pair composed by an "Image" and a "LearntClass" (*_C36_l6c3*, *C461*) with a *true* "classification". Failing is deduced from the comparison, implemented in *T*, between the *C461* "LearntClass" and the *C36* "aPrioriComposes" "Class" of the *_C36_l6c3* "Image". *ET* returns an error for this "LearntClass" when comparing it with the "Class" that "aPrioriComposes" the "Image":

ET(_C36_l6c2 ⊕ C36, EuclideanDistance) = null

ET(_C36_l6c2 ⊕ C461, EuclideanDistance) = false

The third line represents also an instance of the PSM where "Distance" "evaluates" a pair composed by an "Image" and a "LearntClass" (*_C36_l7c1*, *C36*) applying the *AP_8x3_gray* (angular partition) descriptor with a *true* "classification". Success is deduced from the comparison between the *C36* "LearntClass" and the *C36* "aPrioriComposes" "Class" of the *_C36_l7c1* "Image". *ET* returns success for this "LearntClass", which is the same "Class" that "aPrioriComposes" the "Image":

ET(_C36_l6c2 ⊕ C36, EuclideanDistance) = true

ET(_C36_l6c2 ⊕ C461, EuclideanDistance) = null

An equivalent result is obtained for image *_C36_l7c2* in the fourth line. In these two cases, although the classification is not decidable with the *Zernike* descriptor, it is not necessary to test the *ART_5x5_gray* descriptor. The other lines of the running logs represent additional successful classifications. As indicated, *_C36_l7c3*, *_C36_l8c1*, and *_C36_l8c2* images were properly classified by the *Zernike* descriptor. These results indicate that classification is decidable according to the best quality descriptor.

To calculate a Euclidean distance between two vectors of $\mathbb{N}^n$, $\mathbb{R}^m$, ou $\mathbb{C}^p$ formed by descriptor elements, the PDM (Fig. 20) is designed to represent required useful concepts.

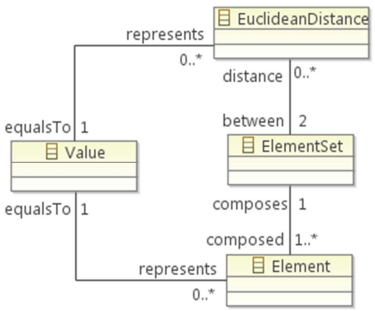


Fig. 20 - PDM meta-model of the image classification process.

The PDM meta-model is therefore composed by the following concepts to compute a Euclidean distance:

- "Value" is a value of $\mathbb{N}$, $\mathbb{R}$, or $\mathbb{C}$.
- "ElementSet" represents a set of elements or a vector.
- "Element" is an element or a coordinate of a vector:
  - "composes" refers to the set containing the element.
  - "equalsTo" refers to the value of the element.
- "EuclideanDistance" is the Euclidean distance between two vectors:
  - "between" refers to the two vectors representing two sets of elements taken to calculate the distance.
  - "equalsTo" refers to the real value of the Euclidean distance.

Additionally, the *ET* transformation contains the implementation of Euclidean distance calculation rules in $\mathbb{N}^n$, $\mathbb{R}^m$, or $\mathbb{C}^p$.

The implementation of PDM constraints on the PICM targets a distance between the selected descriptor elements, defines quantitatively the image to classify, and a class of images mapped with it. The PSM (Fig. 21) results from the implementation of this constraint.

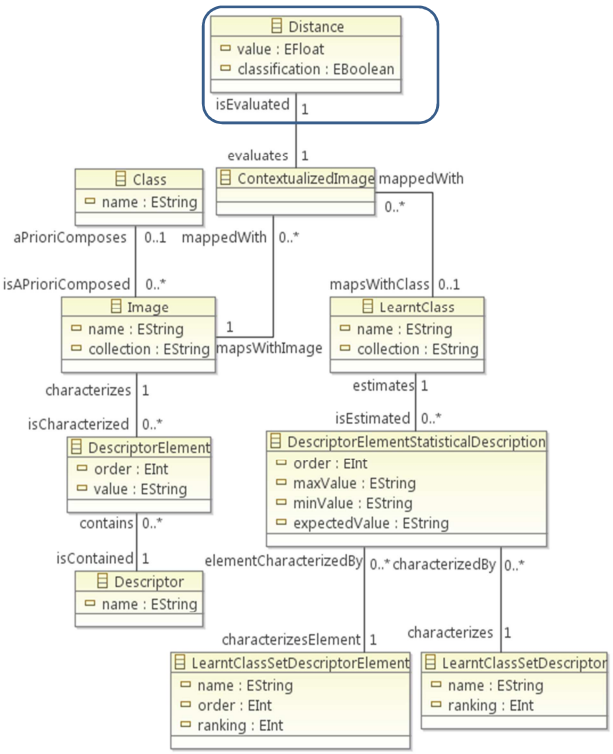


Fig. 21 - PSM meta-model (enclosed) of the image classification process associated to the PICM meta-model and satisfying the PDM meta-model.

The concept of distance (Fig. 21) between descriptor elements can thus be added to the PICM to define the PSM:

- "Distance" representing a distance characterizing an element of a descriptor defining quantitatively the image to classify and a class:
  - "value" is the float value of the Euclidian distance (*73.71026*).
  - "classification" is a Boolean that indicates if the pair, composed by the image and the mapped class, is, or is not, the result of the classification (*true*).
  - "evaluates" refers to a pair composed by an image and a class (*_C36_l6c2*, *C36*).

5.3. CT- and ET-Transformations Execution

Taking as input query images to be classified, the *ET* transformation is constrained by the description of each image class, resulting from the preceding *CT* transformation. An a posteriori classification is obtained and compared to the a priori classification. This comparison provides an error rate and a non-classification rate (when an a posteriori class does not result from the *ET* transformation).

Because of its originality for the MDA approach, the algorithm implemented in the *ET* transformation is detailed below. Decidability of a classification is the *ET* transformation basis, i.e. an image can be classified by means of a descriptor or not. Thereafter, an axiom of decidability to classify an image in a class *C* according to a descriptor *D*, is: classification of an image in *C* is decidable if the distance between the image and the minimum value, the maximum value, and the expected value of *C*, is *significantly* greater than the distance between the image and the minimum value, the maximum value and the expected value of the other classes for descriptor *D*. A "τ decidability threshold" determines quantitatively how significant are compared distances in the transformation. This decidability axiom conforms to the previous one, regarding the quality indicator.

Fig. 22 represents an extract of the proposed algorithm based on the image classification illustration, and implemented in operational-QVT. The algorithm is applied to the η best descriptors' elements. An operational-QVT helper dedicated to image classification and an operational-QVT mapping for creating the PSM concept "Distance" composing the *ET* transformation, fulfill the image classification algorithm and comply with meta-modeling (Fig. 21).

```
//Mapping of distance calculation associated to a pair composed by an
//image and a mapped class
mapping ContextualizedImage::createDistance (dist : Real) : Distance
{
        result.value := dist;
        result.classification := false;
        result.evaluates := self;
}
//Helper for image classification
helper Image::classifyImage (η : Integer) : ContextualizedImage
{
        var i := 1;
        //image to classify
        var image : Image := self;
        //class resulting from the a priori classification
        var classImage : Class := image.aprioriComposes;
        //class resulting from the a posteriori classification
        var classTest : ContextualizedImage := null;
        while (i<=number of ranked descriptors)
        {
                var descript : Descriptor := the ith descriptor;
                var j := 1;
                var seqCI : Sequence(ContextualizedImage)
                        := image.mappedWith->
                        select(e : ContextualizedImage |
                        e.mapsWithClass<>null)->asSequence();
                while (j<= seqCI->size())
                {
                        seqCI->at(j).map createDistance
                                (dist(D(descript , I, η), D(descript, Cj, η));
                        j := j + 1;
                };
                var bestIndex := 0;
                if (seqCI->size()>1)
                then
                {
                        var k := 1;
                        while (k<=seqCI->size() and bestIndex=0)
                        {
```

```
                        var testIndex := true;
                        var m := 1;
                        while (m<=seqCI->size() and testIndex)
                        {
                                // τ decidability threshold
                                if (k<>m and
                                  ((seqCI->at(k).isEvaluated.value /
                                  seqCI->at(m). isEvaluated.value) > τ))
                                then
                                        testIndex := false
                                endif;
                                m := m + 1;
                        };
                        if testIndex
                        then
                                bestIndex := k
                        endif;
                        k := k + 1;
                };
        }
        endif;
        classTest :=
                if seqCI->size()=1
                then
                        seqCI->at(1)
                else
                        if (seqCI->size()>1 and bestIndex<>0)
                        then
                                seqCI->at(bestIndex)
                        else
                                null
                        endif
                endif;
        i := i + 1;
};
if (classTest<>null)
then
{
        if (classImage.name=classTest.mapsWithClass.name)
        then
                log (“image correctly classified”)
        else
                log (“image not correctly classified”)
        endif;
        classTest.isEvaluated.classification := true;
}
else
        log (“image not classified”)
endif;
//return represents the result of the classification
return (classTest);
}
```

Fig. 22 - Example of the ET transformation coding with operational-QVT for the image classification process.

Mathematical functions (PDM for the image classification process) constraining the *ET* transformation, represent various operations:

- *D(descript, I, η)* defines the value of the elements of the *descript* descriptor, selected from the η best elements of the set of descriptors, applied to image *I*.
- *D(descript, Cj, η)* is the value of the *descript* descriptor, selected from the η best elements of the set of descriptors, applied to the *Cj* class depending on the minimum value, on the maximum value, and on the expected value for *Cj*.
- “*Dist*” is the Euclidean distance on the space $\mathbb{N}^m$, $\mathbb{R}^n$ or $\mathbb{C}^p$, corresponding to the space on which are defined the values of the selected elements of the *descript* descriptor.

Integration of a contextual model is therefore feasible for the proposed image classification process, since it takes into account the variability of data associated to image content descriptors and their elements.

6. Discussion

Heterogeneous data produced in variable manners are complex to model. Such complexity is significantly increased when the model is intended to have a direct impact on the associated process. The proposed approach and its first experimental results indicate that contextual models of variable produced data can be automatically generated and integrated into a corresponding process. To handle variable means to generate data, it is required to know the task that generates data and how such data generation is carried out. Additionally, modelling the data context should be independent of modelling the process. On the other hand, the resulting contextual model provides knowledge with direct impact on the concerned process. This is made possible by analyzing the selected data production mode and its representation by model transformation, considering that input parameters are the fundamental integration keys. In the two described use cases these input parameters are, respectively: activity supported by the external service to integrate external services and image class to integrate patterns to be recognized. The whole approach relies on three successive stages, on which an expert defines:

- The mapping between the contextual model and the input of the model transformation.
- A transformation based on a technical constraint and the contextual transformation model.
- The transformations running.

Meta-models parametrized using the concepts of the proposed approach were developed for an information system and an image classification system, to which a suitable contextual model was integrated. Although there is a difference at a conceptual level between these two processes, both are structured according to the three stages of the approach. Code generated for a feature is instantiated during its execution by a user. For instance, code generated from a functional requirement is instantiated to order an instance of a commercial product during its execution by a vendor. Classification grounded on enhancement, is at the instance level of code generation constructed on substitution. Otherwise, pattern recognition targets directly image instances, like the recognition of a commercial product in an image, for example. These two example cases make it possible to underline the difference between mapping with a contextual model using *CT* and a technical constraint with transformations such as *ST* or *ET*. Meta-models associated to transformations define a model fusion role for the first constraint and a constraint pattern role applied to a model for the second constraint. *CT* makes it possible to

produce a PICM from a mapping between the PIM and the TCM, whereas *ST* or *ET* allow, in a classical way for the MDA, to deduce a PSM from a PIM with a technical constraint described in the PDM.

It is relevant to consider that whereas the TCM makes possible to associate an element of the PIM with contextual data, the PDM only describes how the PSM transforms this association. Therefore, a mark defined in the PDM represents a concept in the PSM, and is applied to an element of the PICM, indicating how that element is transformed. Marking is replaced in the PDM, indicating by means of rules how a PICM concept should be transformed, depending on whether it is associated with an element of the TCM or not. Furthermore, instead of a marked PIM, the proposed model uses the PICM constrained by the PDM rules to make the PSM transformation. Since the PIM marked by the TCM cannot generate the PSM, this task is done by the PDM. Correspondingly, the TCM does not require structured marks to define mutually exclusive alternative mappings, permitting to include the contextual data production mode. Likewise, only the context data instances of the TCM that perform certain functions of the PIM are automatically selected. Therefore, several instances of contextual TCM data can be seamlessly associated to the same element, while the PDM filters technically compatible elements, according to a chosen transformation.

The example cases show two data production variability frameworks for which our approach is relevant. The first framework relates to a dataset that can be integrated as a part into the process by substitution. An application service to be reused, replaces the transformation of an activity into an interface and a Java class encoding an application service, during code generation. The second framework concerns a dataset that can be integrated into the process by aggregation. A class of images is aggregated during the classification to search the closest images in the sense of a Euclidean distance. As a result, our work contribution is to relate variable generated data and its context by means of automated integration of a designed system component model, in order to improve the processing of data generated in variable manners. Described results show the proposed approach feasibility in two specific cases and also provide some helpful insights on implementation aspects.

The PDM generates marks applied to the corresponding elements of the PICM. In this way, the PIM is associated with the contextual data of the TCM. However, this marking model cannot take into account the reuse of already produced contextual data and the corresponding modes of production. To this end, knowledge represented in the TCM refines the PIM by associating contextual data to be reused in different manners, without adapting the underlying process. It is the PDM that defines how to process and reuse contextual data, refraining from process adaptation to each data set, as a conventional MDA approach. The extension of weaving to the complete examination of a given contextual data production mode, provides the variability related to the production modes of these contextual data. It does not imply in any way however, that such integration is optimal by definition or design. This could be considered as an aspect to improve in the proposed approach. Still, it is useful to remember how the approach should be applied. Integration rules should be set by the process expert and this work should be facilitated by taking into full account the concepts manipulated by the production method.

From a practical point of view, the proposed contextual model integration approach may be conditioned by the fact that it is designed closely associated to the process development. This aspect is penalizing when the processing of *ST* or *ET* related to process transformations is significant, due to large data variability. Nevertheless, these transformations could be adapted to take into account context changes. The integration of a new information system application service or a new image class could be equivalent to services reuse or image classifications already performed.

Another restriction is the quality of process characterization, on which meta-model concepts related to the MDA approach depend. To properly take context into account requires adding a link between a meta-model describing the context (TCM) and a meta-model describing the input element of the MDA transformation (PIM), in order to design the meta-model of the PICM. The accuracy of the PIM relation with the process characterization should therefore be verified. Similarly, the PDM consistency, as designed for the PIM transformation with the PICM, should be confirmed and, if necessary completed.

7. Conclusion

Modelling of data production variability in systems architecture and engineering is a complex and rarely addressed task. Taking into account the data production context, our work proposes a model to integrate variable generated data as a system component. Context data models are generated within a MDA compatible process in a MDE framework. These models permit to conceive automatically platform specific programs, integrated thereafter in a process. The TCM model extends the MDA approach, by representing the models involved in the production of contextual data, both at the input and output. This TCM model makes possible therefore to automate context integration into a process, independently of the data source and without using MDA marks at that level.

The proposed approach based on contextual data mapping, was tested on two example MDA compliant processes to investigate its applicability. Data production variability that characterizes these two processes was addressed through the generic level offered by the MDE. Such strategy permits automatization applying meta-modelling of the PIM, TCM, PICM, PDM and PSM, along with the *CT*, *ST*, and *ET* transformations coded with operational-QVT, parametrized using the concepts of these meta-models. Consequently, the development of an information system and an image classification system is enriched by the integration of a contextual model, in the form of a ready-to-use services model and a model of image classes, respectively.

Meta-models supporting integration of a contextual model into a process enable useful transformations to automate rules specifying a process solution. Interestingly, model mapping for a real case of data variability is added to MDA and illustrates the various potential application domains of integration of a contextual model into a process. Future works encompass the approach optimization for considerably variable contextual datasets, the dynamic adaptation to context changes, and the extension to new use cases.

References

[1] Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4-7.

[2] Yeo, G. (2013). Trust and context in cyberspace. *Archives and Records*, vol. 34, no. 2, pp. 214-234.

[3] Snidaro, L., García, J., & Llinas, J. (2015). Context-based information fusion: a survey and discussion. *Information Fusion*, vol. 25, pp. 16-31.

[4] Majchrowski, A., & Deprez, J. C. (2008). An operational approach for selecting open source components in a software development project. *European Conference on Software Process Improvement*, Springer Berlin Heidelberg, pp. 176-188.

[5] Zimmermann, A., Lorenz, A., & Oppermann, R. (2007). An operational definition of context. in *Proceedings of the 6th International and Interdisciplinary Conference on Modeling and using Context (CONTEXT)*, Springer Press, pp. 558-571.

[6] Kent, S. (2002). Model Driven Engineering. In *Proceedings 3rd international conference on Integrated Formal Methods*. LNCS vol. 2335, pp 286-298.

[7] Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model-driven engineering. *IEEE Software*, vol. 31, no. 3, pp. 79-85.

[8] Schmidt, D. C. (2006). Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25-31.

[9] Vale, S., & Hammoudi, S. (2008). Context-aware model driven development by parameterized transformation. *Proceedings of the 1st International Workshop on Model Driven Interoperability for Sustainable Information Systems*, pp. 167-180.

[10] Frankel, D. S. (2003). *Model Driven Architecture – Applying MDA to Enterprise Computing*. Wiley Publishing Inc.

[11] Sinnema, M., & Deelstra, S. (2007). Classifying variability modeling techniques. *Information and Software Technology*, vol. 49, no. 7, pp. 717–739.

[12] Tessier, P., Servat, D., & Gérard, S. (2008). Variability Management on Behavioral Models. In *Proceedings 2nd ACM International VaMos Workshop*, pp. 121-130.

[13] Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., & Wąsowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings 6th ACM International VaMos Workshop*, pp. 173-182.

[14] Dumitrescu, C., Tessier, P., Salinesi, C., Gérard, S., Dauron, A., & Mazo, R. (2013). Capturing variability in model based systems engineering. In *Proceedings Complex Systems Design & Management (CSD&M) Conference*, pp. 100-115.

[15] Zhan, G., Ge, B., Li, M., & Yang, K. (2015). A data-centric approach for model-based systems engineering. *Journal of Systems Science and Information*, vol. 3, no. 6, p. 549–560.

[16] Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G. & Plöger, J. (2016). RuleMerger: automatic construction of variability-based model transformation rules. In *Proceedings International Conference on Fundamental Approaches to Software Engineering*, pp. 122-140.

[17] Murguzur, A., Capilla, R., Trujillo, S., Ortiz, Ó., & Lopez-Herrejon, R. E. (2014). Context variability modeling for runtime configuration of service-based dynamic software product lines. In *Proceedings of the 18th International Software Product Line Conference Workshops*, vol. 2, pp. 2-9.

[18] Jaouadi, I., Djemaa, R. B., & Ben-Abdallah, H. (2016). A model-driven development approach for context-aware systems," *Software & Systems Modeling*, pp. 1-27.

[19] Alegre, U., Augusto, J. C., & Clark, T. (2016). Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, vol. 117, pp. 55-83.

[20] Sheng, Q. Z., & Benatallah, B. (2005). ContextUML: a UML-based modeling language for model-driven development of context-aware web services. *IEEE International Conference on Mobile Business (ICMB)*, pp. 206-212.

[21] Ashamalla, A., Beydoun, G., & Low, G. (2017). Model Driven Approach for Real-time Requirement Analysis of Multi-Agent Systems. *Computer Languages, Systems & Structures*, vol. 50, pp. 127-139.

[22] Szvetits, M., & Zdun, U. (2016). Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime, *Software and Systems Modeling*, vol. 15, no. 1, pp. 31-69.

[23] Davies, J., Gibbons, J., Milward, D., & Welch, J. (2012). Compositionality and Refinement in Model-Driven Engineering. In *Proceedings 15th Brazilian Symposium on Formal Methods: Foundations and Applications*, LNCS 7498, pp. 99-114.

[24] Wymore, A. W. (1993). Model-based systems engineering: An introduction to the mathematical theory of discrete systems and to the tricotyledon theory of system design. CRC Press, 1993.

[25] France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap," in *Proceedings IEEE Future of Software Engineering (FOSE)*, pp. 37-54.

[26] Perovich, D., Bastarrica, M., & Rojas, C. (2009). Model-driven approach to software architecture design," in *Proceedings of the IEEE ICSE Workshop on Sharing and Reusing Architectural Knowledge*, pp. 1-8.

[27] Davies, J., Gibbons, J., Welch, J., & Crichton, E. (2014). Model-driven engineering of information systems: 10 years and 1000 versions. *Science of Computer Programming*, vol. 89, p. 88–104.

[28] Hutchinson, J., Rouncefield, M., & Whittle, J. (2011). Model-driven engineering practices in industry. In *Proceedings 33rd IEEE International Conference on Software Engineering (ICSE)*, pp. 633-642.

[29] Aizenbud-Reshef, N., Nolan, B. T, Rubin, J., & Shaham-Gafni, Y. (2006). Model traceability. *IBM Systems Journal*, vol. 45, no. 3, pp. 515-526.

[30] Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., & Aho, P. (2010). Knowledge based quality-driven architecture design and evaluation. *Information and Software Technology*, vol. 52, no. 6, pp. 577-601.

[31] Gascueña, J. M., Navarro, E., & Fernández-Cabal, A. (2012). Model-driven engineering techniques for the development of multi-agent systems. *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 159-173.

[32] Dahman, K., Charoy, F., & Godart, C. (2010). Generation of component based architecture from business processes: model driven engineering for SOA," in *Proceedings 8th IEEE European Conference on Web Services (ECOWS)*, pp. 155-162.

[33] Mattsson, A., Fitzgerald, B., Lundell, B., & Lings, B. (2012). An approach for modeling architectural design rules in UML and its application to embedded software. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 2, p. 10.

[34] Buckl, S., Matthes, F., & Schweda, C. M. (2009). A viable system perspective on enterprise architecture management. In *Proceedings IEEE International Conference on Systems, Man and Cybernetics*, pp. 1483-1488.

[35] Weilkiens, T., Lamm, J. G, Roth, S., & Walker, M. (2015). *Model-based process*. John Wiley & Sons.

[36] Kallel, S., Tramoni, B., Tibermacine, C., Dony, C., &. Kacem, A. H. (2017). Generating reusable, searchable and executable “architecture constraints as services”. *Journal of Systems and Software*, vol. 127, pp. 91-108.

[37] Zachariadis, S., Mascolo, C., & Emmerich, W. (2006). The SATIN component system-a metamodel for engineering adaptable mobile systems. *IEEE Transactions on Software Engineering*, vol. 32, no. 11.

[38] Guinea, A. S., Nain, G., & Le Traon, Y. (2016). A systematic review on the engineering of software for ubiquitous systems. *Journal of Systems and Software*, vol. 118, pp. 251-276.

[39] Gervais, M.-P. (2002). Towards an MDA-oriented methodology. In *Proceedings IEEE International Computer Software and Applications Conference (COMPSAC)*, pp. 265-270.

[40] Recker, J. C., & Mendling, J. (2006). On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In *Proceedings 18th International Conference on Advanced Information Systems Engineering*, pp. 521-532.

[41] Rivera, J. E., Ruiz-Gonzalez, D., Lopez-Romero, F., Bautista, J., & Vallecillo, A. (2009). Orchestrating ATL model transformations. In *Proceedings International Workshop on Model Transformation with ATL (MtATL)*, vol. 9, pp. 34-46.

[42] Bouzitouna, S., Gervais, M.–P., & Blanc, X. (2005). Model Reuse in MDA. In *Software Engineering Research and Practice*, pp. 354-360.

[43] Burger, E., Henss, J, Küster, M, Kruse, S., & Happe, L. (2016). View-based model-driven software development with ModelJoin. *Software & Systems Modeling*, vol. 15, no. 2, pp. 473-496.

[44] Ren, M., & Lyytinen, K. J. (2008). Building enterprise architecture agility and sustenance with SOA. [Online]. Available: http://works.bepress.com/kalle_lyytinen/3/. [Accessed 07 06 2017].

[45] Liu, F., Guo, W., Zhao, Z. Q., & Chou, W. (2010). SaaS integration for software cloud. *3rd IEEE International Conference on Cloud Computing*, pp. 402-409.

[46] Eriksson, D. (1997). A principal exposition of Jean-Louis Le Moigne's systemic theory. *Cybernetics & Human Knowing*, vol. 4, no. 2-3, pp. 35-77.

[47] Simonin, J., Alizon, F., Deschrevel, J.-P., Le Traon, Y., Jézéquel, J.-M., & Nicolas, B. (2008). EA4UP: an enterprise architecture-assisted telecom service development method," in *IEEE Enterprise Distributed Object Computing Conference*, pp. 279-285.

[48] Object Management Group, (2009). MOF 2.0 Query / View / Transformation, OMG Technical Report. [Online]. Available: http://www.omg.org/spec/QVT. [Accessed 05 08 2016].

[49] Dinçer, B. T., Ounis, I., & Macdonald, C. (2014). Tackling biased baselines in the risk-sensitive evaluation of retrieval systems. *Advances in Information Retrieval, Proceedings 36th European Conference on IR Research*, vol. LNCS 8416, pp. 26-38.

[50] Lu, D., & Weng, Q. (2007). A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823-870.

[51] Datta, R., Joshi, D., Li, J., & Wang, J. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, vol. 40, no. 2, article 5.

[52] Chatzichristofis, S., Arampatzis, A., & Boutalis, Y. (2010). Investigating the behavior of compact composite descriptors in early fusion, late fusion and distributed image retrieval. *Radioengineering*, vol. 19, no. 4, pp. 725-733.

[53] Andres, S., Arvor, D., & Pierkot, C. (2012). Towards an ontological approach for classifying remote sensing images. In *Proceedings IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS)*, pp. 825-832.

[54] Zhang, Q., & Izquierdo, E. (2006). Optimizing metrics combining low-level visual descriptors for image annotation and retrieval. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. II 405-II 408.

[55] Yang, L., & Albregtsen, F. (1996). Fast and exact computation of Cartesian geometric moments using discrete Green's theorem. *Pattern Recognition*, vol. 29, no. 7, pp. 1061-1073.

[56] Liao, S., & Pawlak, M. (1998). On the accuracy of Zernike moments for image analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1358-1364.

[57] Chalechale, A., Naghdy, G., & Mertins, A. (2005). Sketch-based image matching using angular partitioning. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 35, no. 1, pp. 28-41.

[58] Hwang, S.-H., & Kim, W.-Y. (2006). Fast and Efficient Method for Computing ART. *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 112-117.

[59] Geusebroek, J. M., Burghouts, G. J., & Smeulders, A. W. (2005). The Amsterdam Library of Object Images. *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103-112.

[60] Barroso, R., Ponciano-Silva, M., Traina, A., & Bueno, R. (2013). Using boundary conditions for combining multiple descriptors in similarity based queries. In *Iberoamerican Congress on Pattern Recognition, Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, LNCS 8258, pp. 375-382.