From Requirements to Code: A Conceptual Model-based Approach for Automating the Software Production Process

Oscar Pastor[*,a], Marcela Ruiz[b]

[a] PROS Research Centre, Universitat Politecnica de Valencia, Spain
[b] Utrecht University, the Netherlands

Abstract. *Conceptual Models are part of an increasing number of engineering processes. The model driven development approach considers conceptual models as first-class entities and also considers tools, repositories, etc. as models. In order to take full advantage of these ideas, model transformation is a main activity. A sound software production process, conceptual-modelling based, must go from the initial requirements model to the final application code through a well-defined set of conceptual models and transformations between them. Model transformation aims at supporting the production of target models from a number of source models, while keeping a full traceability support. The current paper presents a practical application of these ideas using the Model Centred Architecture contributed by Heinrich C. Mayr. In this line, we present our research efforts on the integration of requirements and executable conceptual models. We reflect on the integration of Communication Analysis (a communication-oriented business process modelling and requirements method) and the OO-Method (an object-oriented model-driven development method).*

Keywords. Conceptual Modelling • Conceptual Programming • OO-Method • Communication Analysis • Model-Centred Architecture • Model-Based Software Production

1 Introduction

An essential component of a conceptual model-based software production approach is an executable conceptual model. This is how we start this paper, presenting the OO-Method approach in section 2. To avoid moving from an Extreme Programming perspective to a kind of Extreme Conceptual Modelling point of view, where a particular conceptual model comes from must be clearly determined. This is why we need a concrete requirements modelling strategy (presented through the Communication Analysis perspective in section 3), and a concrete model transformation from the requirements model to its corresponding executable conceptual schema. This is what we discuss in section 4 using a model centred architecture. Concluding remarks and references complete the work.

2 The OO-Method approach

OO-Method is an approach for automatic software generation based on the specification of object-oriented conceptual models (Pastor and Molina 2007). It is supported by Integranova [1] , a model-driven tool that provides an OO-Method modelling environment, a conceptual model compiler, and automatic code generation. OO-Method uses four conceptual models partial views that conform all together a complete information systems specification: i) Object model, ii) Dynamic model, iii) Functional model, and iv) Presentation model.

* Corresponding author.
E-mail. opastor@pros.upv.es

[1] http://www.integranova.com/

The conceptual models are platform independent, i.e., they do not involve platform-dependent characteristics.

Object model

This model specifies the structure and static relationships between the classes of a software system. It provides a graphical notation that can be considered equivalent to a UML class diagram where only a delimited set of primitive constructs are selected as relevant. It basically includes classes, attributes, services (events (simple) and transactions or operations (complex)), and the relationships between the classes.

The basic graphical notation is complemented with the specification of textual integrity constraints written according to a well-defined first-order logical language. As an example, a rent-a-car system will include in its object model classes as car, customer, etc., each one with its corresponding set of attributes and services.

Dynamic model

This model specifies the dynamic and behavioural aspects of the classes of the object model. The dynamic model can be considered equivalent to UML's state transitions diagram. The valid life-cycles of the classes are represented in this model, as well as the possible interactions between different objects (i. e., instances of different classes).

Functional model

This model specifies the semantics of the change of an object state as a result of event executions. For example, a change in the state of an object car (from available to rented) when the rent service is activated. A set of services, preconditions, and post-conditions are specified to define changes in object states. OO-Method provides a declarative language to indicate the constraints that are related to each object state.

The functional model provides the facilities to specify domain dependent restrictions. The declarative language of the functional model is aligned with the object and functional models.

Presentation model

This model specifies the characteristics of the user interface of a software system and how the users will interact with the system; the model is created by means of a pattern-based graphical model through three levels of detail, from more general to more specific characteristics.

Applications generated from conceptual modelling with the OO-Method follows a three-layer software architecture. The presentation layer contains the software components responsible for presenting users the application interface to interact with the software system. The application layer provides services that implement the functionality. The persistence layer provides the services that manage data persistence, in order to store and obtain the pieces of data necessary for the execution of an application.

The software process of the OO-Method consists of two stages. First, system analysts (modellers) create a conceptual schema, which corresponds to a representation of the problem space (i.e., the application domain). A UML-based notation and textual specifications are used according with the four model views commented before. Second, the code of an application is generated on the basis of the Execution Model of Integranova, which corresponds to a representation of the solution space and can be targeted at different technologies.

When comparing the OO-Method with other approaches for software modelling and development, it deals with the static (data-oriented), the dynamic and functional (behaviour-oriented) and the presentation views of an Information System (IS). All together they constitute a complete IS modelling and development approach. In addition, it relies on an underlying formal model OASIS; (Lopez et al. 1992) and it provides a conceptual model compiler intended to make true the conceptual programming goal that states that "the conceptual model is the code" (instead of "the code is the model") (Embley et al. 2011). This strategy allows the generation of complete and

ready-for-running applications by precisely specifying an IS through its conceptual model.

In relation to MDA (Model Driven Architecture; (Miller and Mukerji 2003)), a detailed description of its correspondence with OO-Method can be found in (Pastor and Molina 2007). The main points are that: 1) an OO-Method conceptual schema corresponds to a Platform-Independent-Model; 2) the execution model corresponds to a Platform-Specific-Model, and; 3) the code generated corresponds to an Implementation Model.

3 The Requirements Perspective: Communication Analysis

This section presents the Communication Analysis (CA) method (España et al. 2009). The CA-based approach extends the platform-independent view provided by the OO-Method with the upper Computation-independent model, where requirements modelling is going to be managed. To better understand the underlying ideas from a practical point of view, we are going to use a lab demo to illustrate the use of the Communication Analysis method.

The case presented in this paper is an adaptation of The SuperStationery Co. case. SuperStationery Co. is a company that provides stationery and office material to its clients. The company acts as an intermediary: the company has a catalogue of products that are bought from suppliers and sold to clients. This case is presented in full detail in (España et al. 2011). In this paper, we focus on the part of the sales manager business process (acronym SALE).

3.1 Concepts of Communication Analysis requirements models

To facilitate understanding of the illustrative example, this subsection presents a brief explanation of the concepts used for Communication Analysis (Communicative Event Diagrams and Message Structures).

Concepts of Communicative Event Diagrams

The Communicative Event Diagram (CED) is a business process modelling technique that adopts a communicational perspective and facilitates the development of an IS that will support those business processes (España 2011) (González et al. 2009). A communicative event is a set of actions that are related to information (acquisition, storage, processing, retrieval and/or distribution), which are carried out in a complete and uninterrupted way.

The unity criteria allows communicative events to be identified. Each communicative event is represented as a rounded rectangle and is given an identifier, a number and a descriptive name (e.g. SALE 1 in Figure 1).

For each event, the actors involved are identified. The primary actor triggers the communicative event and provides the input information. For instance, the client is the primary actor of the communicative event SALE 1.

The interface actor is in charge of physically interacting with the IS interface. Interface actors are specified at the bottom of the event. For instance, the salesman is the interface actor of the communicative event SALE 1. The receiver actors are those who need to be informed of the occurrence of an event. The sales manager is the receiver actor of the communicative event SALE 1.

An ingoing relationship is a communicative interaction that feeds the IS memory with new meaningful information. The main direction of the ingoing communicative interaction is from the primary actor to its related communicative event. For instance, the relationship named ORDER between the primary actor client and the communicative event SALE 1 is an ingoing communicative interaction. An outgoing relationship is a communicative interaction that consults the IS memory.

The main direction of the outgoing communicative interaction is from the communicative event to its related receiver actor. For instance, the relationship named ORDER that is between the communicative event SALE 1 and the receiving

actor SALES MANAGER is an outgoing communicative interaction.

The precedence relationships are represented as arrows among communicative events (e.g. SALE 1 requires the previous occurrence of PROD 2 and CLIE 1).

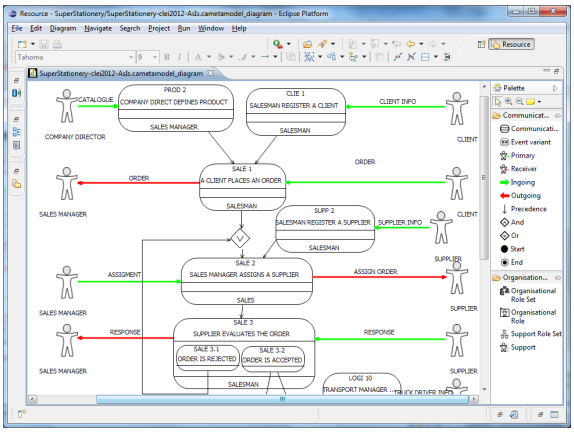

Figure 1: CED of the Sale process of SuperStationery Co.

Concepts of Message Structures

Message Structures is a specification technique that allows the message that is associated to a communicative interaction to be described (González et al. 2011). A substructure is an element that is part of a message structure. This way, LINE, Client and Payment type are substructures that are part of the substructure ORDER (See Figure 2).

There are two classes of substructures: fields and complex substructures. A field is a basic informational element of the message that is not composed of other elements. A data field is a field that represents a piece of data with a basic domain. For instance, payment type is a data field. A reference field is a field whose domain is a type of business object. For instance, Client refers to a client that is already known by the IS.

A complex substructure is any substructure that has an internal composition. An aggregation substructure specifies a composition of several substructures. It is represented by angle brackets < >. For instance, LINE=<Product+Price+Quantity> specifies that an order line consists of information about a product, its price, and the quantity. An iteration substructure specifies a set of repetition of the substructures it contains. It is represented by curly brackets { }. For instance, an ORDER can have several lines for each product requested.

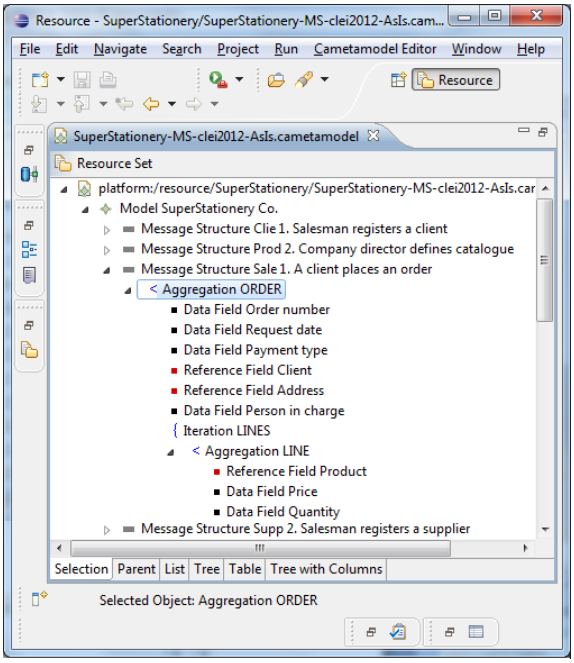

Figure 2: Message Structure of the communicative event Sale 1

The Communication Analysis Method is supported by means of the GREAT Process Modeller (Rueda et al. 2015). GREAT allows creating communicative event diagrams (i.e. business process models), specifying message structures (which describe the messages associated to each communicative event), and automatically generating a class diagram (representing the data model of an information system that would support organisational communication).

4 Generating an Executable Conceptual Model

España (2011) integrates the Communication Analysis and the OO-Method. The integration strategy is based on an ontological alignment of the concept

definitions in which the two methods are grounded. As a result, the ontological alignment provides precise transformation guidelines to transform Communication Analysis models into OO-Method object models (see Figure 3). The systematic derivation of OO-Method conceptual models from Communication Analysis requirements models is offered in two flavours: a set of rules to be manually applied by a human analyst, and an ATL model transformation that automates this task (Jouault and Kurtev 2006).

This is where the link between requirement modelling and conceptual model execution is established in a precise way through the connection between a CA model and its associated OOM model, the result of the transformation.

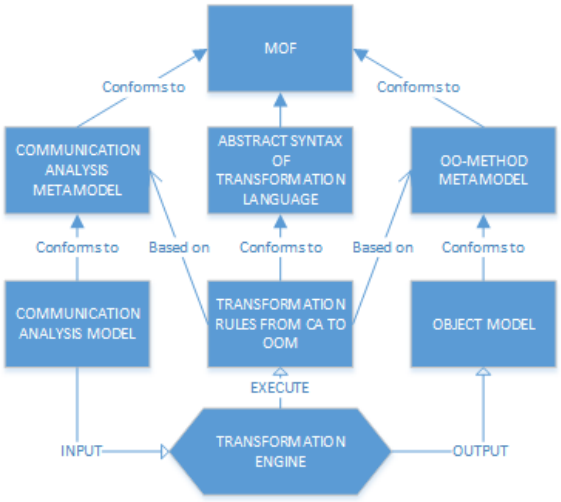

Figure 3: Model derivation strategy from (Jouault and Kurtev 2006)

4.1 Model Centred Architecture of Communication Analysis and OO-Method

Our research line applies the core concepts of the Model Centred Architecture paradigm (Mayr et al. 2017). As a result, the requirement and conceptual models of Communication Analysis and OO-Method can be seen as the core of information systems. Figure 4 presents an overview of the Model Centred Architecture paradigm applied to the Integration of Communication Analysis and OO-Method.

The Model Centred Architecture for the integration of Communication Analysis and OO-Method ensures a complete support for domain-specific applications. Thanks to model-driven tools like GREAT and Integranova, it is possible to give the power to system users and metamodel authors to specify and maintain information systems.

5 Conclusions

The current paper reflects on our research efforts for automating the software production process. The practical application of the Model Centred Architecture is reflected in the integration of the model-driven methods Communication Analysis and OO-Method. The application of the Model Centred Architecture benefits the role of system users by facilitating the maintenance and usage of model-driven tools.

In the near future we plan to explore the application of round trip model transformations. The idea is to involve system users when applying the transformation engines from requirements to code. The involvement of end users in the transformation process will ensure the implementation of domain specific requirements that should be specified during the transformation process. In addition, we plan to integrate a tool support for goal modelling to our current software production process. With the intentional perspective, the entire Model Centred Architecture acquires different views that will enrich the final software product.

We plan to improve the metamodel and model exchange interfaces support. The idea is to facilitate the connection among the different requirement specifications (processes, goals, and conceptual models) and software code. In this line, one big challenge will be the design of a modelling environment in which the traceability links become explicit assets.

In the long term, we intend to implement an evolution support for metamodels, models, and domain models. In this future scenario, the system

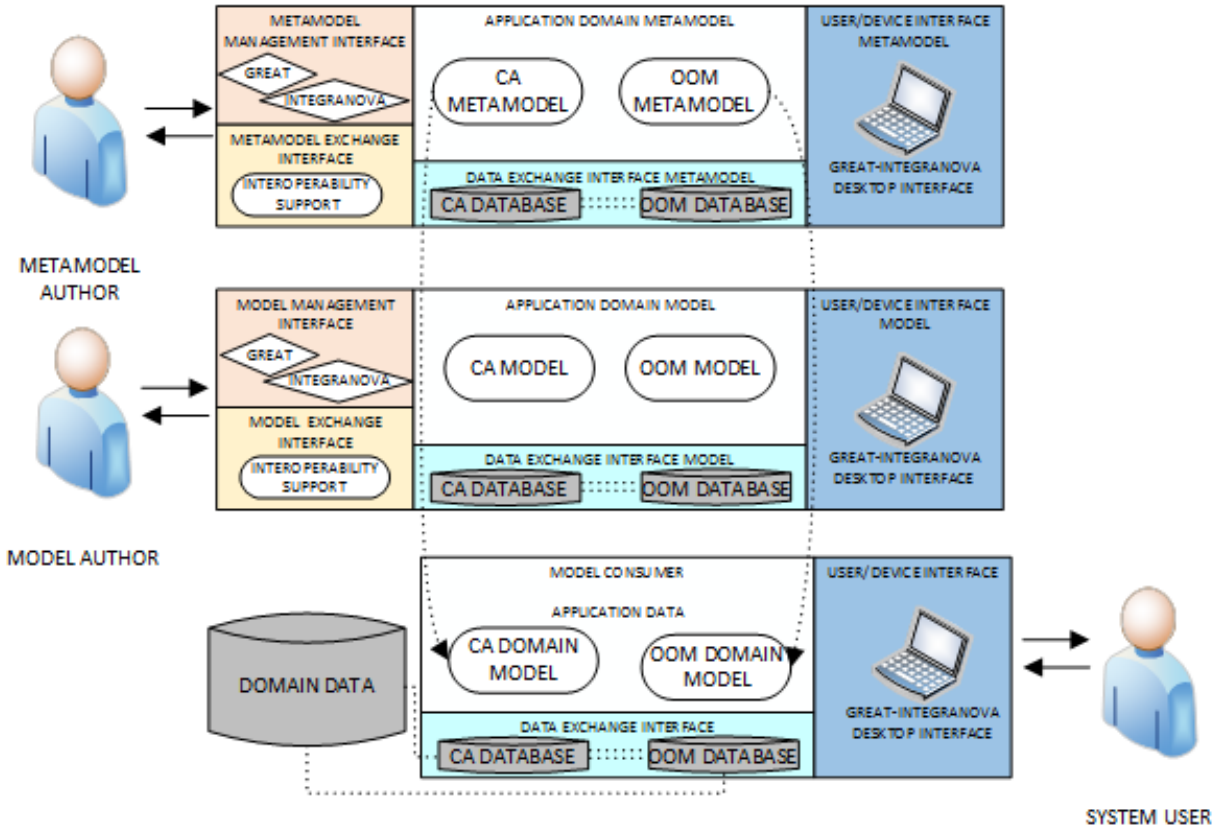

Figure 4: Model Centred Architecture for Communication Analysis and OO-Method

users will be in charge of the maintenance of the three layers of the Model Centred Architecture.

References

Embley D. W., Liddle S. W., Pastor O. (2011) Conceptual-Model Programming: A Manifesto In: Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges Embley D. W., Thalheim B. (eds.) Springer, pp. 3–16 https://doi.org/10.1007/978-3-642-15865-0_1

España S. (2011) Methodological Integration of Communication Analysis into a Model-Driven Software Development Framework. PhD thesis, Universitat Politecnica de Valencia http://hdl.handle.net/10251/14572

España S., González A., Pastor O. (2009) Communication Analysis: A Requirements Engineering Method for Information Systems In: Advanced Information Systems Engineering: 21st International Conference, CAiSE 2009 van Eck P., Gordijn J., Wieringa R. (eds.) Springer Berlin Heidelberg, pp. 530–545

España S., González A., Pastor O., Ruiz M. (2011) Integration of Communication Analysis and the OO Method: Manual derivation of the Conceptual Model. The SuperStationery Co. lab demo. In: CoRR abs/1101.0105 http://arxiv.org/abs/1101.0105

González A., España S., Pastor O. (2009) Unity criteria for Business Process Modelling: a theoretical argumentation for a Software Engineering recurrent problem. In: IEEE, p. 10

González A., Ruiz M., España S., Pastor O. (2011) Message Structures a modelling technique for information systems analysis and design. In: p. 12

Jouault F., Kurtev I. (2006) Transforming Models with ATL. In: Bruel J.-M. (ed.) Satellite Events at the MoDELS 2005 Conference. Springer Berlin Heidelberg, pp. 128–138 https://doi.org/10.1007/11663430_14

Lopez O. P., Hayes F., Bear S. (1992) Oasis: An object-oriented specification language. In: Loucopoulos P. (ed.) Advanced Information Systems Engineering. Springer Berlin Heidelberg, pp. 348–363 https://doi.org/10.1007/BFb0035141

Mayr H. C., Michael J., Ranasinghe S., Shekhovtsov V. A., Steinberger C. (2017) Model Centered Architecture In: Conceptual Modeling Perspectives Cabot J., Gómez C., Pastor O., Sancho M. R., Teniente E. (eds.) Springer International Publishing, pp. 85–104 https://doi.org/10.1007/978-3-319-67271-7_7

Miller J., Mukerji J. (2003) MDA Guide Version 1.0.1.. Object Management Group (OMG)

Pastor O., Molina J. C. (2007) Model-Driven Architecture in Practice. Springer-Verlag Berlin Heidelberg, Upper Saddle River, NJ, p. 302 https://doi.org/10.1007/978-3-540-71868-0

Rueda U., España S., Ruiz M. (2015) GREAT Process Modeller user manual. In: ArXiv e-prints http://adsabs.harvard.edu/abs/2015arXiv150207693R