A PROCESS-DRIVEN MODELLING OF INFORMATION SYSTEMS

Andrea Gualtieri
Exeura srl/University of Calabria
Department of Electronics, Informatics and System (DEIS)
Edificio Polifunzionale
87036 Arcavacata di Rende (CS) - Italy
agualtieri@unical.it

Domenico Saccà
ICAR CNR/University of Calabria
Department of Electronics, Informatics and System (DEIS)
Via Bucci – Cubo 41c
87036 Arcavacata di Rende (CS) - Italy
sacca@unical.it
ABSTRACT

In this paper we propose a process-driven methodology for continuous information system modelling. Our approach supports the whole information system cycle, from planning to implementation and from usage to re-engineering. The methodology includes two different phases: firstly we produce a scenario analysis adopting a Process-to-Function approach to capture interactions among organization, information and processes; then we produce a requirement analysis adopting a Function-for-Process and package oriented approach. Finally we deduce an ex-post scenario analysis deduced by applying process mining techniques on processes execution traces. The whole methodology is supported by UML diagrams organized in a Business Model, a Conceptual Model and an Implementation Model.
1. INTRODUCTION

Our methodology for software planning and development is configured as an approach alternative to traditional phases based on waterfall model [Royce]. It adopts iterative procedures as in “lightweight” and “agile” programming [Cockburn] and it is characterized by short recurrent steps, that are target-oriented and able to an adaptive evolution.

The whole macro-process of software planning and development is modelled by specifying two phases, directly connected to the concepts of “process” and “function”. From this point of view, the well known Feature Driven Development [Coad et Al.] approach proposes a planning and implementation incremental procedure oriented to homogeneous groups of functions. Our methodology integrates this approach with an analysis of the processes that constitute the scenario in which the functions are required. The software planning and development path is so composed by two macro-phases:

- in a first phase, we produce a scenario analysis adopting a P2F (Process To Function) approach: our goal is to capture the interactions among organization, information and processes. We model the decomposition of processes in subprocesses and activities and connect them to the required information contents and to the specific portion of the business organization involved in it;

- in the second phase, we produce a requirement analysis for the information system adopting a F4P (Function For Process) approach, in which system development is modelled, planned and dynamically reported with respect to a package-oriented criteria: the features of the system are in fact grouped in packages and integrated with processes that characterizes scenario.

After the implementation and the enactment of the information system, the logs of execution are stored and analyzed by process mining techniques. This way we are able to deduce path of process execution and so we can produce an ex-post analysis of scenario, highlighting differences due to the adopting of the information system realized. This ex-post analysis of scenario should be used as input of a new iteration of the whole methodology.
The whole methodology is supported by UML diagrams that are able to offer a meaningful expressivity to developers and consultants and an immediate interpretation to customer [Booch et Al]. In particular:

- in the P2F phase, we adopt an approach based on Usecase and Activity Diagram, to obtain a “Business Model” characterized by a process-oriented representation.

- In the F2P phase we still adopt Usecase Diagrams to model the logical structure of the functions, but progressively we introduce Deployment Diagrams and Class Diagrams to model the architecture of the information system and the physical elements, like interfaces, databases and data control modules. As result of this phase we obtain a “Conceptual Model” to define the functional analysis and an “Implementation Model” to formalize the requirement for the realization of the information system.

2. THE ANALYSIS OF SCENARIO AND THE BUSINESS MODEL

The definition of the business processes characterizing the scenario in which the system will operate is a milestone of the planning phase. A business model is so an essential input to a subsequent individuation of functions able to manage information that is useful in a specific context.

The analysis is obtained as result of the study of the organization, interviews to the members of the organization, reading of documents, individuation of relevant procedures. All these elements are referred in the business model that is a formalization of the organization processes, actors and information.

A business model is so characterized by a “Process Schema” section, an “Actor Schema” section and an “Archive Schema” section.

Actor and archive representation
Actors and archives are a formalization of active and passive entities that interact with processes. We represent them by adopting stereotypes of the UML actor element. We consider as “actor” all the operators (human or automatic) that activate or enact a process. An “archive” is instead every information source useful for the process execution.
In the archive and actor sections, we represent taxonomies and ontologies to allow a contextualization of information and of organization elements.

**Process representation**

The processes are modelled with a top-down approach. We analyze processes and then individuate the subprocesses that characterize each of them. We consider:

- **process**: is considered as a set of procedures that are finalized to obtain a goal, starting from a set of input; it involves a number of actors and it requires a set of information; it is composed by “subprocess”;

- **subprocess**: is an element of the process, more restricted than a process, but with the same level of formalization; it individuates the whole path required for the release of a sub-product (or sub-service); it should be structured (i.e. composed itself by other subprocesses) or atomic (in this case it is named “activity”);

- **activity**: is an atomic element that constitutes a specific portion of work and represents a logic step inside a process; a process gives a structure to the activity flow to obtain a goal.

To obtain a better visual representation, we adopt a notation that requires a different color to indicate usecases corresponding to structured processes and atomic activities.

The top-down analysis considers the high level processes characterizing the scenario. We introduce a package for every macro-process: every package contains a UML Usecase Diagram, in which the usecase element corresponding to the process is connected with the usecase elements corresponding to every subprocess. To model these connections we use “include”, “extend” and “specialize” associations provided by UML. In particular we assume that:

- a process P “includes” a subprocess P1 if in every instance of P is required to execute an instance of P1;

- a subprocess P1 “extends” a process P if in every instance of P, an instance of P1 is executed only if is verified a condition expressed by the “extension point” element;
- a subprocess P1 “specializes” a process P if it involves all the subprocesses involved in P and it involves other specific activities.

A generic UML association is adopted to connect a usecase representing a process or a subprocess and an actor or archive element. We are so able to express that an actor execute a process (or a subprocess) and the process requires or modifies information contained in an archive.

For each process, we then model the path of execution of its subprocesses, associating an UML Activity Diagram to the structured process. So in the Usecase Diagram we represent a first analysis about the composition of the process and in the Activity Diagram we are able to formalize the sequence of execution of the activities and to express pre-condition and post-condition by adopting join, fork, merge constructs.

![Figure 1: From an Usecase Diagram to an Activity Diagram](image)

Then we replicate this decomposition for every subprocess that is itself a structured process: we individuate its subprocesses and connect to it by “include”, “extend” or “isa” relations, so we model the dynamic of the evolution by linking it to a specific Activity Diagram. This way, for each process P we introduce an Activity diagram containing just subprocesses P_i, directly connected to P: if a
subprocess P₁ itself involves subprocesses P₁₁, their sequence of execution should be represented in another Activity Diagram connected to P₁.

If a subprocess is too much articulated, to model it we introduce a sub-package containing another usecase diagram, to obtain a modular and incremental process view. In the main Usecase Diagram we represent the sub-package and its main process and we connect it to the process that contains it. This way, we are able to obtain a modular representation of processes, that should be modified in a particular portion without conditioning the whole structure of the model.

![Figure 2: A modular representation of processes implemented in Enterprise Architect](image)

3. THE ANALYSIS OF FUNCTION AND THE CONCEPTUAL MODEL

The analysis of scenario describes the context in which the system should operate. The following step is to analyze the functions that the system should provide to
facilitate the execution of the processes inside the organization. The output of this phase is a conceptual model, able to offer:

- a formal schema of functions and users;
- a formal schema of data;
- a formal schema of the interaction between function and data.

Our approach provides a Conceptual Model that represents the functional blocks and the views on data structures: the functional blocks are modelled by usecases packages and taxonomy of actors, by adopting an approach similar to the one used for the processes in the Business Model; the data views are instead represented by using UML Class Diagram. A Conceptual Model is so characterized by:

- a dynamic analysis represented by a “Function Schema” section and an “User Schema” section
- a static analysis represented by a “View Schema” section and a “Data Schema” section.

The representation of functions and users
The representation of the User Schema in the Conceptual Model has the same syntax of the one relative to the Actor Schema in the Business Model. If the actors are the entities (human or automatic) that activate or enact the process, the users are instead the entities (human or automatic) that will interact with the system.

Also the functions are modelled in the Conceptual Schema by adopting a syntax analogous to the one adopted in the Business Model to represent the processes. The archives are substituted by the “views” that are physical representations of a portion of database.

The representation of data and views
The Data Schema contains a UML Class Diagram that represents a conceptual model of the database of the system. We use "Table" and "Key" stereotypes to adapt classes and attributes and represent a data schema. The foreign keys and cardinality constraint are instead represented by modelling UML associations between classes. At this level, we use “composition” and “aggregation” associations and taxonomies to represent logical relations between entities.
Figure 3: A view and its documentation implemented in Enterprise Architect

The View Schema is a package that contains a class diagram named “View catalogue”. A view is a portion of database useful in a specific functional context. Every view is represented by a package containing a class diagram in which the specific classes of the database are shown. In every package we represent also an actor with the stereotype “view” that is used to represent the view in the Function Schema. The documentation of the actor contains also additional information as: the name of the view; for each class the list of the specific attributes – a selection of the whole set of attributes - that are useful in the specific context; the way used in the specific context for navigate of the associations between classes.

4. THE IMPLEMENTATION MODEL AND THE DEVELOPMENT

Once the requirement analysis is completed and the Conceptual Model is defined, a physical planning of the system is necessary. The guidelines defined in the Conceptual Model are mapped on the architecture designed for the system. In relation to the specific system, it is possible to choose different architectural
approaches, but every approach should be characterized by three levels of implementation:

- a database level to represent data source of the system;
- a control level to represent classes that manages the logic of the system procedures;
- an interface level to represent the forms that manage the interaction between the system and the users (human or automatic).

The Implementation Model is so composed by an “architecture” section, containing a representation of the physical elements of the information system, an “interface” section, a “control” section and a “database” section to model respectively the level of interaction with the system, the level that manages the logic of the system and the level that manages the data.

Every section is modelled by a package. The “Architecture” session contains an UML Deployment Diagram in which is possible to define node and components characterizing the system. It is also possible to use sub-package to obtain a modular representation of the system.

The “Database” section contains a UML Class Diagram enriched by stereotypes, named Database Schema, to represent the schema of the data contained in the database of the system. With respect to the Data section of the Conceptual Model, we now adopt just general associations and cardinality constraints to obtain a linear description of the database.

The “Control” section contains a UML Class Diagram, named Control Schema, in which a control class’s catalogue is represented. Every control class (a UML class with stereotype <<Control>>) contains methods used by the interface level to manage the access to the data level. Every control class should refer a view that is represented by inserting in the Control Schema a package containing the classes of the Database Schema useful in the specific context. The methods of each control class are expressed in the specific area of the UML class and they are explained in the documentation of the class.
Figure 4: A Control Schema implemented in Poseidon for UML

The control level is so invoked in the “Interface” section that contains a UML Class Diagram with a schema of the interaction with the system. The path of interactions is expressed in the Activity Diagrams of the Conceptual Model: we use that path to model a sequence of form, that are classes enriched by specific stereotypes. A form is characterized by three elements to interact with it:

- an “entry unit” is an area in which the system receives input elements; it should be implemented in different ways with respect to the needs of the programmer;
- a “data unit” is an area in which are shown information derived from database;
- a “display unit” is an area in which are shown static information;

To express each attribute transmitted by units we add an attribute to the specific class: the kind of attribute transmitted is coherent with the kind of the attribute of the UML class in the form.
A form should be a plain form, a list form or a recursive form: the second one is modeled by a stereotype “FormL” and is used to represent a form in which is shown a set of tuples; the recursive form is modeled by a stereotype “Form*” and is used to represent a form that is shown many times, one for each tuple corresponding to a specific parameter.

To obtain the path of interface, the forms are connected by directional associations: we can also use a stereotype <<LinkP>> to express that a parameter connected to a form is transmitted to the following.
5. THE APPLICATION SCENARIOS

This methodology is actually experimented in Exeura Srl, a spin-off company of the University of Calabria that operates in the IT area.

With respect to different application scenarios, the methodology should be executed in a complete or in a partial way, with respect to the P2F and the F4P phases.

The P2F phase is executed by an “agile” approach, characterized by short and adaptive iterations, each of them contains an analysis and a modelling step. The interactions with customers are so frequent and allow reviewing and validating the analysis, eventually enriching it by elements useful to start new iterations.

The F4P phase is instead executed in an incremental way: a first architectural and functional model is delivered, so every iteration is composed by a specific functional analysis and an implementation phase, in which is allowed a high level of concurrency and feedback. In particular, the Implementation Model is structured to enable reverse engineering operations from the source code.
If the whole software planning and implementation process should be executed to realize a novel system, the P2F and the F2P phases are both required. By adopting process mining techniques to analyze the traces of execution of the process [Greco et Al.] is then possible to deduce also an ex-post evaluation of the process scenario that generate a new iteration of the whole methodology. If a re-engineering of an existing system is required, the analysis of scenario should be derived by the analysis of the process logs generated by process mining tools as ProM [is.tm.tue.nl/~cgunther/dev/prom/].

Actually, to support the UML modelling and the direct and reverse engineering, we adopt the EA Architect Platform, provided by SparxSystem [www.sparxsyste.com]. Other tools as Poseidon for UML are available.

6. REFERENCES


