ANNALS of the ORADEA UNIVERSITY.
Fascicle of Management and Technological Engineering, Volume VI (XVI), 2007

USING BUSINESS GRAPHICS SOFTWARE IN SYSTEM ANALYSIS CASE STUDY: SMART DRAW DIAGRAMS FOR SALARY CALCULATION

Prof. PhD. Eng. Ec. Ioan BOJAN
Technical University of Cluj-Napoca

Key words: structured analysis, unified process, use-case model, iterative

Abstract
The fast development of business and the competitiveness on the market has made business software performance a key to the success of organizations. Practice has proven that software development is most efficient when following a structured model. In this paper we focus on structured analysis as part of system requirements management. We provide an example of the way business graphics software can be used for analyzing the functionality of a salary calculation application.

1. Structured Analysis and the Unified Process:

There is no business nowadays that strives to be competitive on the market and does not rely, even on a minimum level, on software for communication, product development, internal administration, report generation and many more.

But in order to be effective and efficient, the software development process has to be structured, as there have been too many projects failing because they didn't meet functionality, cost or delivery schedule requirements.

Therefore, several models have been suggested: the "traditional" waterfall model, the spiral model, the star lifecycle model, the scenario based model and so on. In one way or another they are all adaptation of a basic scenario, which consists of the following steps:

- software elements analysis
- elaboration of specification
- software architecture
- implementation
- testing
- documentation
- maintenance

The pursuit of a methodology for optimizing and reducing the risk of software development has led to the creation of Unified Process (UP) [1], an iterative and incremental software process development framework. Iterative software implies that the system be divided in a sequence of small, fixed length, projects (called "iterations"). The result of each iteration is an implementation of the "miniproject", tested and integrated with the previous iterations. Therefore, the development process is also incremental, with each step refining and building up more of the final project. The success keys of this approach are feed-back and adaptation and they provide important advantages:

- eliminating the main risks of the project in an initial stage
- user oriented
- complexity management improvement
- continuous learning

There are six best practices considered by UP and RUP (Rational Unified Process) to lead to successful software development [1, 3]. The basic condition is to *develop software*

iteratively. There is also the need to *manage requirements* (with the use of scenarios and use-cases). Other recommendations are to: *use component based architectures*, *to model the software visually*, to *check* in parallel the *software quality* and to *control changes to the software.*

2. The Use-Case Model

In this paper we are interested in the requirements management field and, in particular, in the set of functionalities and environment features captured in the *use-case model*[1]. Use case modeling is directly related to business modeling, and therefore to the business use-cases. These depict specific business processes striving to evaluate the objectives and to point out the limits of the system. More specifically, we need a process model to tell us what activities are to be performed and to show the dataflow between them, that is "a formal way of representing how a business system operates" [4].

There are two main types of process models: *logical* process models, which describe *what* processes consist of without showing *how* they are developed, and *physical* process models, which point out the way processes are built into the system [5].

In *structured system analysis* the focus is on the logical view of the processes. The main modeling tools of modern systems analysis are:

- process specification
- data flow diagrams
- data dictionary
- state-transition diagrams
- entity-relationship diagrams [5]

In the following section we will show how business graphics software can be used for system analysis and we will exemplify by modeling a business application for salary calculation. Due to the space constrains, the focus will be on presenting the data-flow diagrams, data dictionary and state-transition diagrams.

3. Case Study: SmartDraw Diagrams for Salary Calculation

SmartDraw is a very popular business graphics software, that provides a graphical interface for rapid diagrams, business forms, flowcharts, organization charts (etc.) creation. It has a very wide symbols library, extracted from various domains: education, engineering, science, office, network design, software design and many more.

The application we are modeling is a salary calculation program for a small company. The program uses data collected from various external (Ministry of Finance, other public or private institutions) and internal (management, accounting, human resources) sources. The data are centralized and processed in the salary calculation module, which will also enable report generation and flyer printing for each employee.

One of the most effective ways of illustrating system functionality is the *data flow diagram*, a graphical representation of the data flow between processes or business functions. It is also an efficient tool for communicating with team members, managers and users issues regarding the functional requirements.

[1] The use-case model captures all the contexts of the system and the desired functionalities.

There are several different types of data flow diagrams: *context* diagrams and *level* 0, *level 1*… *level n* diagrams. The context diagram defines the scope and the boundaries of a system, while representing the entire system as one process. It also includes the external entities and, of course, the data flows linking them to the system. For our salary calculation application, a context diagram example is shown in Fig.1:

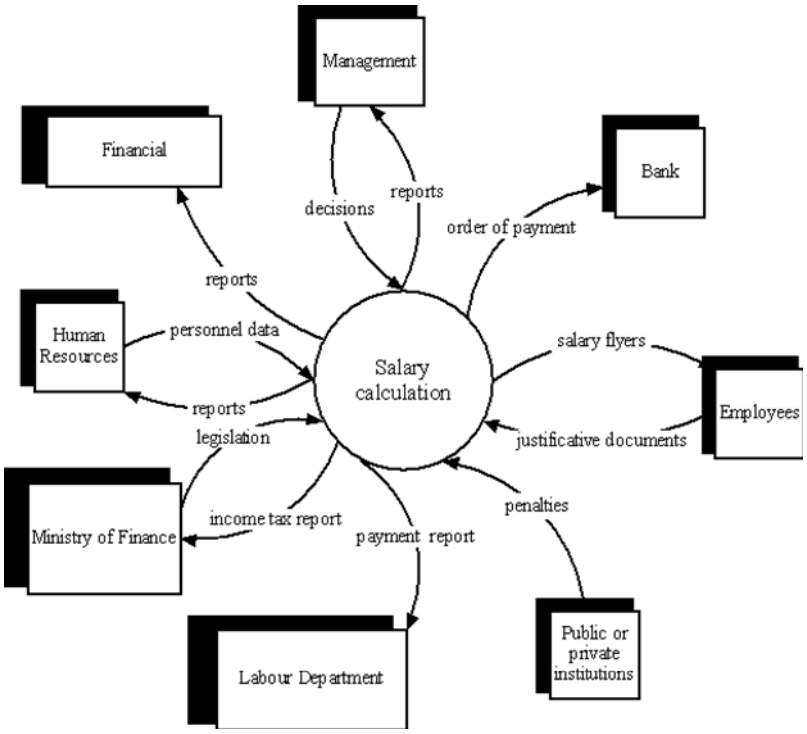

Fig. 1 Context diagram for the salary calculation application

The level 0 diagram derives from the context diagram. The system is divided into the main constitutive processes and data stores are added to the schema (see *Cards*, *Work* hours *registration* in Fig.2).

The next levels diagrams are iterative refinements of the level 0 diagram, which means that one process in a given chart is described by splitting it into sub processes in the next level diagram.

There are several data flow design issues that should be avoided:

- "black holes" – data flows are all pointing to the process/data source and no data flow is coming out
- "miracles" – there's no input to the process/data source, only output
- data flows unassociated with a process
- too complex diagrams (more than 9 processes)

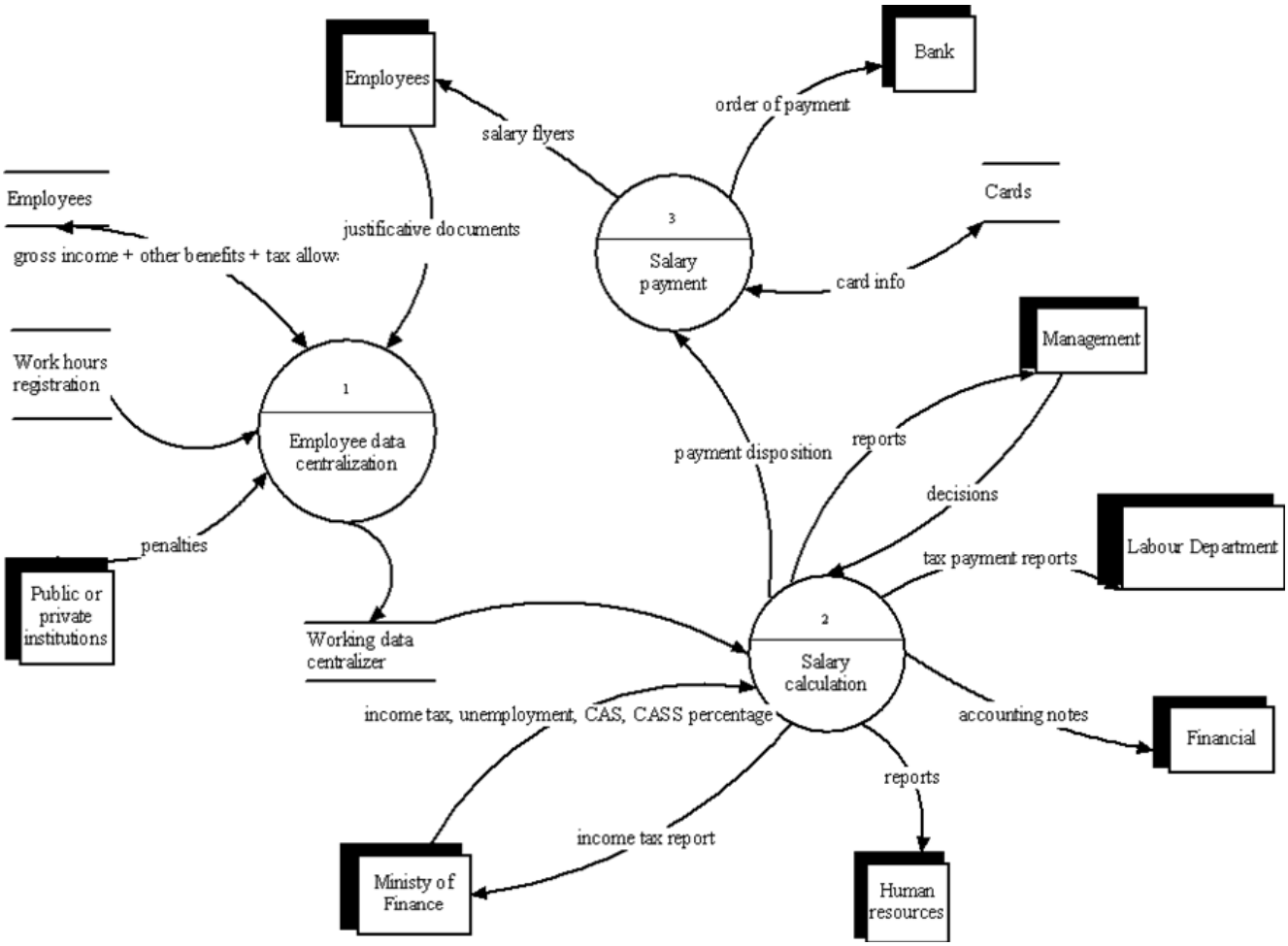

Fig.2 A level 0 diagram for the salary calculation application

Data flow diagrams consist of processes, data flows, data stores, terminators (external entities) and offer an overview of the functionalities of the system. For a more detailed description of the processes and transformed data that passes through them process specifications and a data dictionary are needed. Process specifications have the purpose of defining the actions that need to be done in order to transform inputs into outputs in a form that is easy to understand for analysts and users. The data dictionary completes the overall understanding of the system by describing the meaning of the flows and stores in a data flow diagram, the data composition and the relationships between data stores by using a formal notation. [5]

For example, the *Cards* data store can be defined as:

Cards = id + name + surname + CNP + account-number
id = {number}4
CNP = 13{number}13
account-number = {number}
number = [0-9]
name = {alphabetical-character}
surname = {alphabetical-character}

alphabetical-character = [A-Z | a-z]

The meanings of the notations used are:

"=" – composed of

"+" – and

"{}" – iteration (the number to the left symbolizes the lower iteration limit and the one to the right the higher limit)

"[]" – selection of one of various choices

"|" – alternative choices separator

For complex systems where real-time response is crucial, time-dependent behavior is modeled by state-transition diagrams. In [5] it is pointed out that these diagrams are not essential for some business systems where the sequence of steps is trivial and the outputs of a process are inputs to the other. In the case of salary calculation the state-transition is not compulsory, but we are showing it as an example (Fig. 3).

In state-transition diagrams rectangular boxes represent possible states of the system, maintained over a period of time. Transitions from one state to another are pointed out by arrows and generated by one or more *conditions*. Each state can have zero or more *actions* following the change of state.

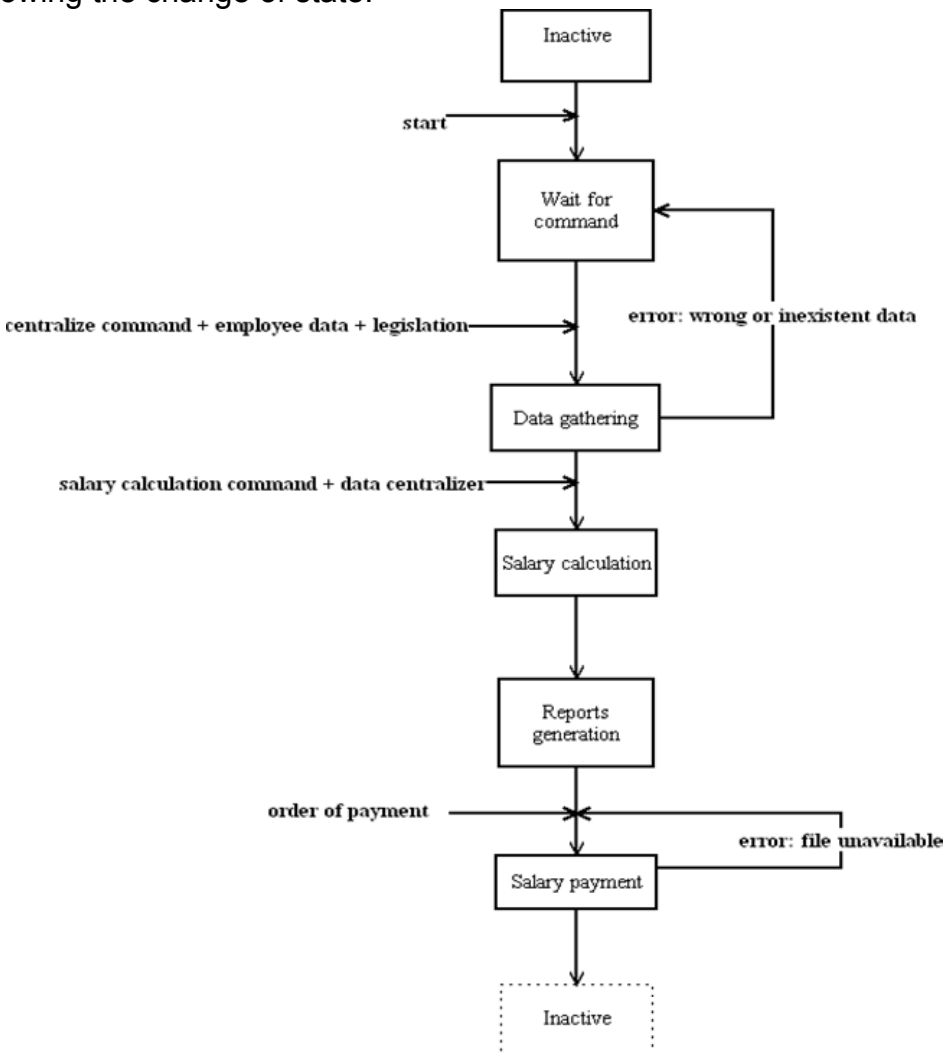

Fig. 3 State-transition diagram for salary calculation

As stated above, there are other diagrams and graphical instruments to be used for system analysis. The ones presented in this paper are, besides UML diagrams, the most popular because of their effectiveness and ease of use.

Companies employ them, together with UML diagrams, to reduce the risk of software projects, to catch the primary use-cases and functionalities and to communicate with users.

References

[1] Jacobson, I., Booch, G., Rumbaugh, J., "The Unified Software Development Process", Addison-Wesley, 1999
[2] Pop, O., "Informatică Administrativă şi Economică", UTCN, 1997
[3] "Rational Unified Process. Best Practices for Software Development Teams", Rational Software Corporation White Paper
[4] Dennis, A., Wixom, B.H., RothJohn, R., "System Analysis and Design", Wiley & Sons, Inc., 2006
[5] Yourdon, E., "Just Enough Structured Analysis", 2006, www.yourdon.com