A user requirements elicitation tool[1]

Miguel Angel Laguna
Departamento de Informática
Universidad de Valladolid
mlaguna@infor.uva.es

José Manuel Marqués
Departamento de Informática
Universidad de Valladolid
jmmc@infor.uva.es

Francisco José García
Departamento de Informática y Automática
Universidad de Salamanca
fgarcia@gugu.usal.es

Abstract

Use cases are nowadays the technique of choice for the functional requirements definition of computer systems. Their use implies that most of the desired functionality of the new system is well known. The aim of this work is a methodological proposal (and a tool to support it) to define accurately this functionality, starting from the way the end-users currently do their work. This method and tool are independent of the development paradigm and could generate useful results for the more frequently used CASE tools with the addition of the appropriate translators.

Keywords

Requirement, CASE tool, Document-Task diagram, use case, DFD, event list.

1. Introduction

The appearance of UML [1] as standard OO modelling language in the community of computer science have consecrated use cases [2] as the favourite technique in order to identify and define the functional requirements of a computer system, in detriment of DFD and other alternatives. Many criticisms have been made to use cases, see for example [3], but we will focus a problem previous to the their construction, without discussing the goodness of the technique.

The use of this technique implies that in certain way the desired functionality of the new system is well known. In many approaches it seems like the end-user and the analyst know perfectly how will be the future system. Our work is directed to support the previous phase of development, when the end user doesn't still know what he wants or he knows what he wants but not how he wants it. We think it is more realistic to base the requirement elicitation in the experience of the end user own work (he will guide us firmly since he is doing this every day). Then, taking this experience as starting point, the analyst can define the functionality of the new system, studying several possible alternatives. In particular, several automation borders can be simulated seeing the effects in the project span. This previous work should be carry out without delaying of the "true development", i. e. the definition of use cases (or DFD, if we are involved in structured development) for the future system.

Our proposal consists of a method of work, using techniques of proven success, and a tool to support it, in order to define accurately what the functionality of the new system will be, starting from the current way of work.

[1] This work has been partially financed by the CICYT (TIC97-0593-C05-05) as a part of the MENHIR project.

2. Description of Document-Task diagrams

Document-Task diagrams (DTd) [4] have been used as a support for the analysis of the results obtained from the interviews to the users of the future system. It is mainly a diagram representing the movement and treatment of the information since it enters into a system until it goes out conveniently transformed. This information, which can be physically supported on any medium (including not structured oral communication) is represented in an abstract form as a "Document". The use of this technique of analysis is close to the systemic approach to Software Engineering, like is the case of the Merise method [5], for instance.

Although it could be enriched with other icons, specifying several different information supports or means of communication for example, the basic version of the DTd yield enough benefits for our objectives. This is because the abstraction of any type of information in an element, the abstract Document, facilitates the reengineering of business processes that the requirement analysis of the future system could involve. Therefore, only three basic elements will be used, Task, Document and external Agent. Also, we will divide the information system in several subsystems, one for each different kind of job detected (Post). Figure 1 samples a typical fragment of a diagram showing the interaction of clients and suppliers with the system. When the system receives an order from a client, a series of Tasks and associated Documents are produced. One of the Documents is filed, probably for later usage by other Tasks.

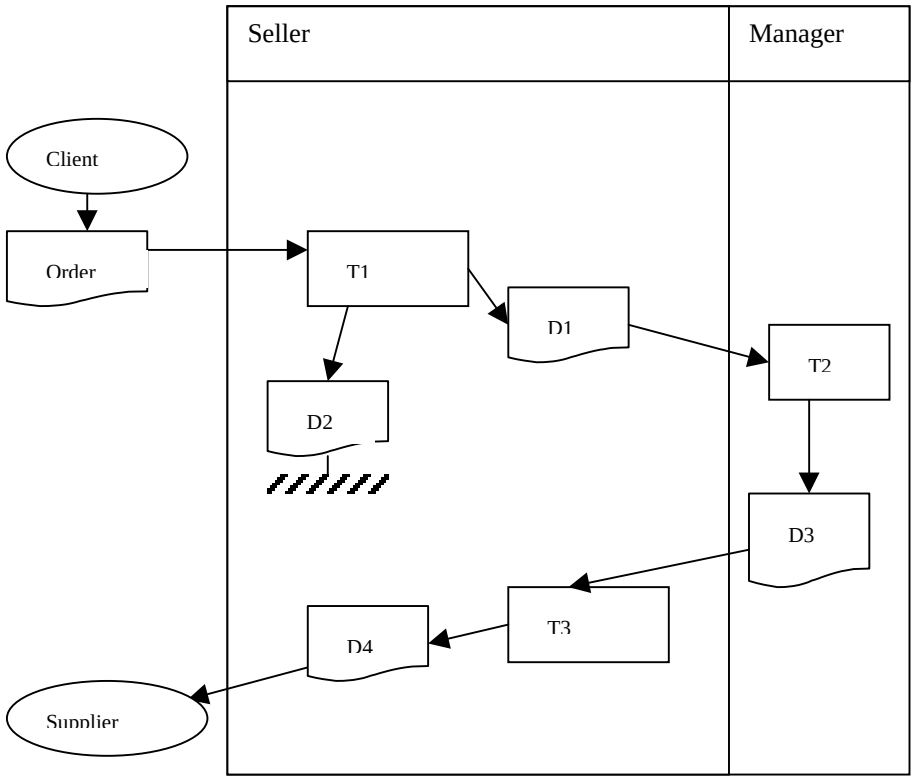

Figure 1: Typical fragment of DTd.

The original usage of DTd's allows obtaining a series of related Tasks, and starting from them, it is possible to remove the current organisation. We propose a simple algorithm to generate automatically profitable results in form of DFD or use cases. Increasing the information of the diagram (adding details about the data of each Document or associating screens or report prototypes to it) would be also extremely useful. The following sections clarify these ideas.

3. Transformation of the DTd's

Once the Tasks performed by the system are established, the Posts can be analysed, particularly it can be determined which of them will carry out exclusive or mainly manual Tasks in an immediate future. As always there are vague situations, in order to deal with them, several combinations can be studied, eliminating one or several Posts in alternative forms. On the other hand, the so marked temporal Tasks (controlled by temporal events) should be examined in detail and the possibility they could be controlled by a Document should be analysed. Also, it is possible that a filed Document (a passive Document) could be transformed to an active Document, able to throw a temporal Task. The aim of this analysis is to remove the constraints of the old system that are associated to the current way of doing things. This analysis allows frequently simplifying the process, unifying Tasks and improving the system.

An example of what we are saying could be seen when changing figure 1, so that the Post "Manager" becomes an external Agent (for example because of the Task he does is manual and should continue to be manual according to the business rules, as is the case of a signature authorising an order of payment). In that situation, two external Documents appear, whose entering or exiting flows cross the limits of the system (D1, D3). At the same time, a series of Tasks that could be unified in a global one, become two independent Tasks, controlled by the arrival to the system of the Order and D3 Document.

For each selected alternative an event list and associate responses of the system can be initially obtained. It is enough to apply a simple algorithm for traversing the diagram. In order to find the external events we must locate each incoming flow toward the system and its associate (external) Document. We could name the events of this type "arrival of the X document." The lists will be completed looking for all those Tasks marked as temporal (the appropriate name for a temporal event will be "every month", for example).

Each event of the list is the starting point of several paths through the graph determined by the related Documents and Tasks. A path finishes when a filed Document, an external Agent (there will be one or several Documents that go out of the system) or a temporal Task (in this case we will finish in the immediately previous Document) is reached. Documents that neither enter the system, nor leave it nor are filed disappear. An event doesn't have a unique path associated to it. Any Task that generates more than one output document will originate a tree or a series of parallel paths that converges.

This first result we have described is not the only possible but the most evident. We are going to propose a transformation of the DTd to specific diagrams for the two main paradigms of development: DFD for structured development and use cases diagrams for OO development.

In structured analysis, all the possible responses of the system (according to the Documents that enter the system or the temporal Tasks) must be used to produce an event list (that we have already obtained) and a first rough version of the essential model [6], expressed like a context DFD and a first level DFD.

The outlines of the transformation to the context DFD will be the following:

- There will be only a process that represents the system in study.
- Each external Agent and each Post outside the system (provided it is a receiver or a direct source of any external document) will become an external entity in the context DFD.
- Each external Document will become an entering or leaving flow, which is associated with the corresponding external entity. A Document that enters or leaves the system will be respectively an entering or leaving flow. The information obtained about these external Documents (for example, the lists of included data) will be the base for documenting the corresponding flows.

In order to generate a first behaviour model, the following additional transformations are needed:

- A process will be assigned to each series of paths found in the building of the previous event list. The entering and leaving Documents will be related with some response and as a consequence with the generated process. The initial definition of each process will be the list of activities of each system response.
- The problem of the data stores could be solved in two ways:
  - Create a data store containing all the data. Any process can read and write this data store (simple but too poor).
  - Create a series of data stores, one for each filed Document in the original DTd. An auxiliary data store, like the mentioned in the previous solution, could be used to synchronise those processes that remain isolated.

The data flows needed to connect these data stores with the processes of the previous point can be obtained from the original DTd (they are equal to the flows of the DTd that link the filed Documents with all Tasks). In this initial version, they will be described by all the data in the data stores.

In the second proposed transformation, the translation of the event list in a group of use cases that show the functionality of the system is needed. Each event in structured analysis had an equivalent process and now each system response is the description of a use case. We have already mentioned in the introduction that a use case includes the actor that will interact with the system and therefore include decisions of the final solution design. Since our idea is to postpone that kind of decisions, the actor of our use cases can be an anonymous one, "the system user", and the developer is responsible of assigning each use case to a concrete actor. Optionally, different actors can be generated using the information we have about the internal Agents of the system. Every Post that receives the external Document controlling the corresponding event or that is responsible for carrying out a temporal Task will become an actor.

For the description of each use case, the list of activities of the related global Task is used. In this proposal, we don't seek to manage situations that take advantage of the whole power of the use cases regarding relationships of "use" or "extend", but it could be possible extensions to our work. The simplest and most intuitive idea is to recognise patterns of serial activities inside the description of a wider Task (this it would lead to identify a use case that "uses" a simpler one) or, on the contrary, "discontinuous" patterns of activities (and in this case we would have an use case "extended" by a more complex one). Both situations are easy to find because of the innate tendency of developers to describe a Task cutting and pasting activities from another (the users also describe frequently a Task Y saying "is like X but varying...").

4. The DTd tool

After explaining the creation and transformation of DTd's, it is quite evident that if we have a tool to support them, drawing and manipulating these diagrams in an easy and effective way, we could obtain profitable results for their usage by other CASE tools, structured or OO. In both cases the starting point is the same. The difference will be the form of representing the final usable result for CASE tools appropriate to the corresponding development paradigm.

The form of representing the event list could be a simple text file and the format of representation of DFD's will be CDIF, as far as possible, and at least a format from a tool of wide presence in the industry and used by us in the Software Engineering courses as Easy CASE is.

In OO we have chosen XML as tool-independent and Rational ROSE as targets of our translation. The reason is evident, due to its wide diffusion this tool has, in particular since the standardisation of UML as "the" modelling language. The possibility of ROSE of exporting and importing diagrams or portions

of diagrams make relatively simple to generate ASCII files with the necessary information in order to build ROSE models.

The current version of the prototype has been developed for Windows [7] and the implemented functionality includes the creation, maintenance and validation of the syntactic correctness of the DTd's. As additional results, only the first proposal (events list of the system) is obtained. After a running period of practical usage in the Software Engineering courses of the University of Valladolid and with the lessons learned, we are working in the complete version of the tool that will be able to generate all the diagrams proposed in this work.

5. Conclusions and future work

We have presented a methodological proposal with the aim of obtaining functional requirements independently of the paradigm of development and with an easy adaptation to both structured and OO analysis. At the same time the technique is easy to understand for the end users and allow validating the partial information received from them. The second advantage of the technique is the simulation of several design scenarios regarding the borders of the new system. For each scenario, a series of results can be generated, mainly initial DFD's or use cases, according to the chosen development paradigm, with evident advantages.

As a final result, now in phase of refinement, an initial prototype of a CASE tool that implements much of the exposed ideas has been built. Many work remain to be done. It is necessary to enrich the functionality of this prototype, therefore we could be able to generate not only text (event list) or very basic versions of DFD's or use cases, but also more complete versions of DFD's, use cases and extension of them with sequence or object interaction diagrams.

6. References

[1] **Rational Software Corporation**. "*Unified Modelling Language*". Rational Software Corporation. Version 1.1. 1997.

[2] **Jacobson, I., Christerson, M., Jonsson, P. and Overgaad G.** "*Object Oriented Software Engineering*", Addison-Wesley, 1992.

[3] **Berard, E.V.** "*Be Careful with Use Cases*". Technical report, 1995.

[4] **Collongues, Huges y Laroche**. "*Merise. Methode de Conception*", Dunod, 1987.

[5] **Tardieu, H. et al**. "*La Methode Merise*". Les Editions d'Organization.1983.

[6] **Yourdon, E**. "*Modern Structured Analysis*", Prentice-Hall, 1991

[7] **Guirles, J.M**. "*Herramienta para el Diseño de Diagramas Documentos-Tarea*", P.F.C., University of Valladolid, 1998.