Laguna, M. A., Marqués, J. M., & García-Peñalvo, F. J. (2003). DocFlow: workflow based requirements elicitation. *Information and Software Technology, 45*(6), 357-369. doi:10.1016/s0950-5849(03)00031-4

DocFlow: Workflow Based Requirements Elicitation

Miguel Ángel Laguna, José Manuel Marqués,
Departamento de Informática
Universidad de Valladolid
{mlaguna, jmmc}@infor.uva.es

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

Abstract

Use cases are the favoured technique for defining the functional requirements of a software system, but their use implies that the desired functionality of the new system is well known. The aim of this work is to present an alternate procedure -and a supporting tool- to accurately define this functionality, expressed as use cases, starting from the workflows that describe the end user work. The use of hypergraphs in the proposed algorithm of transformation reinforces the generation process. In addition, the technique is independent of the development paradigm and a variation in the algorithm allows obtaining Data Flow Diagrams.

Keywords: Requirements Engineering, CASE tool, Workflow, Use case, DFD

1. Introduction

Use cases, first introduced by Jacobson [12], are used to define the behaviour of a system: Interaction between the user and the system is described through a main course of actions along with a possible set of alternative courses. The appearance of the Unified Modelling Language (UML) [3] as standard object oriented (OO) modelling language in the computer science community has promoted use cases as the favourite technique for identifying and defining the functional requirements of a computer system, to the detriment of Data Flow Diagrams (DFD) and other alternatives. UML defines a use case as "The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system". Many criticisms have been made of use cases, see for example Berard [1], but we shall focus on a problem prior to their construction, without discussing the worth of the technique. The use of this technique implies that, in a certain way, the desired functionality of the new system is well known. In many approaches the end user and the analyst seem to know perfectly what the future

system will be like. Our work aims to support the previous phase of development, when the end user still doesn't 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 requirements elicitation on the experience of the end-user's own work (he will give us clear guidance since he is doing this every day). Then, taking this experience as the starting point, the analyst can define the functionality of the new system. This preliminary work should be carried out without the delay of the "true development", i.e. the definition of use cases (or DFDs, if we are involved in structured development) for the future system.

There are many works whose aim is to obtain models of the future user interface, such as [22], and tools for extending the possibilities of the use cases, or to facilitate its definition as in Cockburn [5] or Díaz [7]. Similarly, in the business world, there are specific techniques for business process reengineering (BPR) [11]. However, there are not many techniques that relate the business world with that of Software Engineering, although recently, in [13] a two phase method for obtaining requirements is proposed: a use case model of the enterprise as seen from the outside (the clients are the main actors) and a use case model of the future system.

In a parallel way, in recent years, the workflow concept has been incorporated into the information systems of enterprises as a modelling tool for processes and, at the same time, as infrastructure (*workflow enactment*) to give support to and to enable intercommunication between the specific applications of invoicing, accounting, etc. The *Workflow Management Coalition* (WfMC) [21] has established standards for both possibilities. From the point of view of requirements elicitation, the part of the standard concerning the modelling and definition of workflows is of special interest.

Our proposal consists of a work method, using techniques of proven success and a tool to support it, to accurately define the functionality of the new system, starting from the current work method. It is based on a kind of documental workflow or DocFlow. This workflow allows to express in a natural way the work of a group of end users and can be easily validated by them. After this validation, the analyst must transform this workflow into the new system requirements, studying several possible alternatives. In particular, several automation borders can be simulated and the effects on the project span can be foreseen. We propose algorithms to generate profitable results in the form of DFDs or use cases. A CASE tool designed by us can perform automatically this transformation, reducing considerably the time dedicated by the analyst to the specification of functional requirements. As a side effect, the use this tool permit to study several alternatives set of requirements, without significant cost increase.

The proposal is organized as follows. Section 2 gives a more detailed description of the workflow diagram used. Section 3 describes the transformation of DocFlows into use cases. Section 4 presents an alternative DFD generation. Section 5 is dedicated to the tool we have designed and describes the characteristics implemented in a prototype used in the University of Valladolid. Finally, the conclusions are presented.

2. Documental workflows: The DocFlow diagrams

The standard model proposed by the WfMC [21] establishes a series of minimum requirements for a tool that models and/or controls the Tasks carried out in an information system can be considered to be a workflow management system. To be precise, it defines a Workflow Management System as "a system that defines, manages and executes workflows" and then describes each of these three characteristics. The definition of workflow models is the most interesting point for our purposes. Each workflow management system usually has its own tool for defining models and even admits the use of pencil and paper to define them. However, the current trend recommends the use of standards for exchanging models between different tools, and all the proposals coincide in the use of XML to define the exchange formats [19].

The metamodel proposed by the WfMC makes up a minimum basis that incorporates its own extension mechanisms. Basically, it defines activities (with associated applications, resources and participants) and transitions that connect these activities. The constructions it supports include sequential, alternative, iterative and parallel activity Flows. On the other hand, the workflows can be seen with differing degrees of detail: activities alone, activities plus the necessary physical resources, activities plus those responsible for them, etc. This view leads to a representation of multiple planes, very useful for analyzing each aspect separately. In this way, for example, the paths the activities form can be studied independently of who carries them out, or the use of necessary resources can be optimized.

Among the different model extensions, the documental workflows are the most attractive variation for information systems analysis. In this variation, the documents handled by the system are explicitly added to the model. Document-Task diagrams [4] have been used as a support for the analysis of the results obtained from the interviews of the users of the future system. It is mainly a diagram representing the movement and treatment of the information since it enters a system and remains there until it comes out conveniently transformed. This information, which can be physically supported on any medium (including unstructured oral communication), is represented in an abstract form as a *Document*. The use of this analysis technique is close to the systemic approach to Software Engineering as proposed in the Merise method [20].

Although it could be enriched with other icon types (to specify different information supports or media types), a simplified version of this technique, the DocFlow diagram, yields 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 in the diagram: Task, Document and External Agent. We will also divide the information system into several subsystems, one for each different Internal Agent detected. Figures 2 and 3 sample typical fragments of diagrams from a case study, showing the interaction of Zone Agents, Customers, and Industry Delegation with a system that records the request for Discharge of electrical installations. When the system receives a requests from a Zone Agent, a series of Tasks and associated Documents are produced. The matching of the Tasks

with the activities of the standard workflow model and of the Documents with the transitions (if they are shown explicitly instead of as simply arcs on a graph) is obvious.

Consequently, the basic elements of a DocFlow diagram are as follows:

- **External Agent:** This is the origin or final destination of the Documents handled by our system. It may be an individual person, organizations or other system. They are the equivalent of the concept of external entity of the DFDs.

- **Internal Agent:** Information processor in the current system, equivalent to a type of worker or set of workers (even a department) that carries out the same Tasks. Automatic subsystems (e. g., the accounting system can be considered as a component to be integrated in the new global system) can be considered Internal Agents. We must stress the fact that we are talking about Agents belonging to the enterprise's Information System and it is usually necessary to leave out management and production personnel. On the other hand, especially in small organizations, the same person may carry out the functions pertaining to any of the enterprise's three systems. In this case it would be convenient to distinguish between the different roles that person assumes and to differentiate between them. Finally, some of these Internal Agents can later be redefined as External Agents depending on the computerization frontiers of the proposed system. The concept of Internal Agent is similar to that of the actor in use cases.

- **Document:** Defined as any transmission of information, independently of the format or physical support used. The Documents can be generated in the exterior by an External Agent or in the interior in which case they will be associated to a Task and, indirectly, to an Internal Agent. For the sake of convenience, they can be duplicated in different parts of a DocFlow and even in different DocFlow that represent various workflows within the organization. This last option allows several DocFlow to be connected for joint study. An external Document is defined as a Document created by an External Agent, or rather, a Document sent to an External Agent (in both cases the system's frontier is said to have been crossed).

- **Task:** set of activities carried out in the system by any Internal Agent. A Task can be triggered by the presence of one or several input documents or by a temporal event (alternatively, a temporal event can be considered as a Document coming from a special External Agent that provides information concerning the time that has passed). Each Task must have a visible result, that is, it must generate a Document. The Tasks can be controlled by their preconditions (logical expressions built up from data from the input documents and from the temporal events involved). Although it is not necessary from the conceptual point of view, it is interesting to consider the possibility of exploding a Task to a new DocFlow with greater detail using the well-known mechanism that is applied to the DFDs.

- **Flow:** is an arc that joins the nodes of the three previous types. Flows in both directions are legal between a) an External Agent and a Document, b) a Document and a Task. A Flow that crosses the system's frontier determines the existence of an associated external Document. If it begins in a Document generated in the exterior and ends in an internal Task, it is an external input Document. If it begins in a Document generated in the interior and ends in an External Agent, it is an external output Document.

The route types proposed by the abovementioned workflow standard can be built with legal Flows, providing a correct DocFlow has the following characteristics:

- Two or more Documents can appear as inputs to the same Task. It should be specified whether they are two necessary Documents (AND-join Task) or whether the presence of only one of them is sufficient to trigger the said Task (OR-join Task). The existence of a Task triggered by a temporal event implies that the temporal condition forms an AND expression with the set of the input Documents.
- A Task can generate two or more Documents alternately (OR-split Task) or simultaneously (AND-split Task)

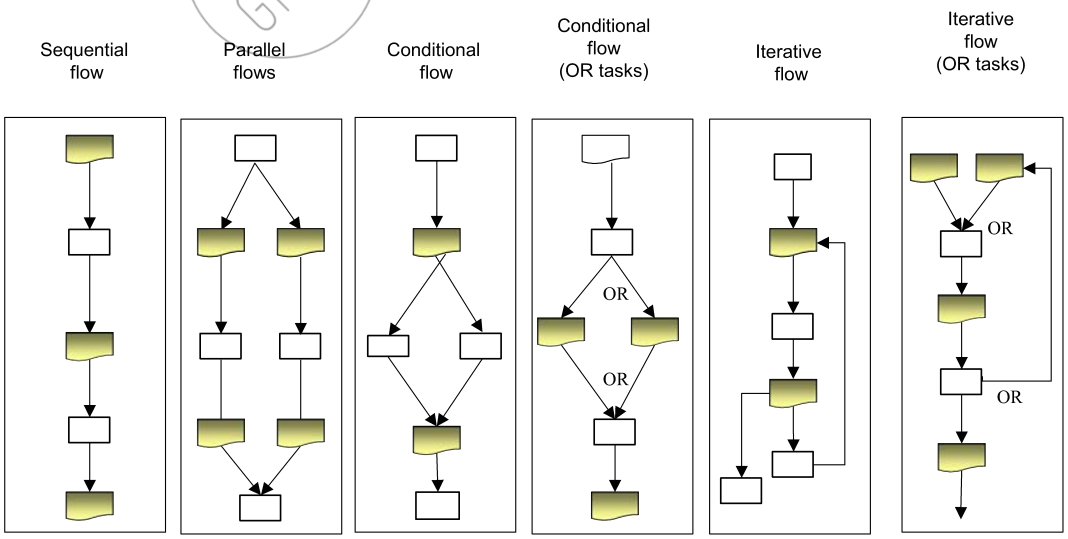

Figure 1: Different types of workflow as DocFlow diagrams. OR-split Tasks allow conditional flows, OR-

- A single Document can only be generated by two or more Tasks alternately.
- A single Document can only be used by two or more Tasks alternately.

Thus, Documents behave by definition as OR-join and OR-split. With these possibilities we have the necessary elements for building loops or alternative routes (using, to do so, Documents generated alternately by a Task or Documents that alternately trigger two Tasks) and parallel routes (with Documents generated simultaneously by a Task).

Figure 1 shows the possibilities of the DocFlow for modelling the different types of workflow. The sequential routes are obvious. The parallel Flows should be built with AND-split and AND-join Tasks (if the branches converge). The conditional and iterative Flows can be built using Documents or OR-join and OR-split Tasks.

The possibility of duplicating Documents in different, related DocFlows gives us the possibility of connecting several parts of the organization in a linear manner, in so far as the possibility of exploding Tasks allows us to connect the workflows between each other in a hierarchical manner. The joint use of both techniques creates a powerful tool for analyzing the business Flows in the organization.

The abovementioned elements and their graphic representation, which can be intuitively appreciated in the examples

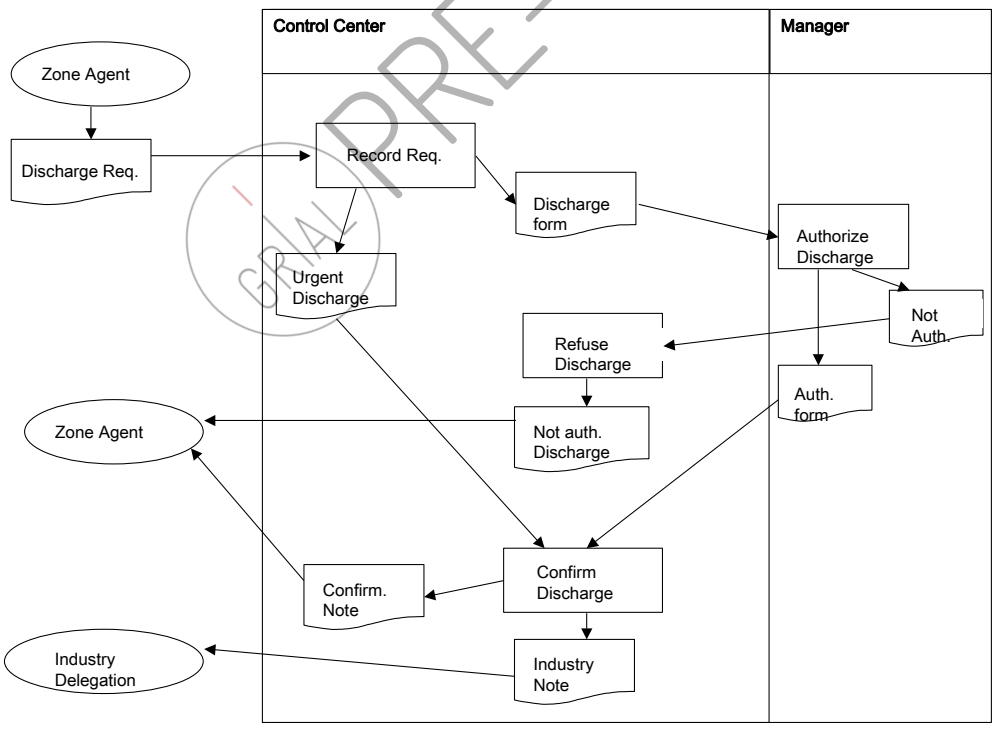

Figure 2. A partial DocFlow of a system that records the request for Discharge of electrical installations

of figures 2 and 3, allow this kind of diagram to be built in a mechanical way just by following the description users

make of their daily work (each user usually begins by explaining that "s/he receives a document from the department W1 and deals with it, sending a copy to the department W2 and filing a copy..."). The Internal Agents of the system under study can optionally be shown in the DocFlow as lanes, following the style of the UML activity diagrams [3]. One advantage of this diagram is that it shows inconsistencies and inefficiency in the present way of working (documents that are filed away and never recovered, documents that go to the department W2 but which are never used by that department, etc.). We thus have at our disposal the necessary elements to carry out a reengineering of the business processes.

During the rest of the article, we use a case study, showing two partial DocFlows from a real system that records the request for discharge of electrical installations. This fragments represent only a simplified version of a relatively complex system designed for an electric company [18]. When the system receives a requests for discharge (i.e. a disconnection) from a Zone Agent, a series of Tasks and associated Documents are produced. The internal agents are the Control Center, Local Operator and Manager.

Once all the Tasks that are carried out within the system have been established, and admitting that the Flow of Tasks itself is correct and has been validated by the users, improvements in the organization's global workflows can be proposed.

On the one hand, Internal Agents can be analyzed and if some of them carry out basically manual Tasks (delivery of documents to the post office, signatures and authorization for outgoing documents, etc.), or Tasks it would be difficult to do automatically, at least in the near future, then these Internal Agents can be labelled "not automated" and will be dealt with in exactly the same way as External Agents. As always, there are vague situations which can be dealt with using one of several combinations, or one or several Internal Agents can be removed. The complexity of the resulting system can be very different in each case and can be estimated as a function of the number and scope of the events and the system's responses, obtained as shown in the following section. An example of this can be seen by changing figure 2 so that the Internal Agent "Manager" becomes an External Agent (for example, because the Task s/he does is manual, and should continue to be so according to the business rules, as is the case with a signature authorizing a Discharge). In such a situation, three external Documents appear, whose entering or exiting Flows cross the limits of the system: Discharge Request and (Not) Authorization Forms. At the same time, a series of Tasks that could be unified in a global one become two independent Tasks, controlled by the arrival in the system of the Discharge Request and (Not) Authorization Forms.

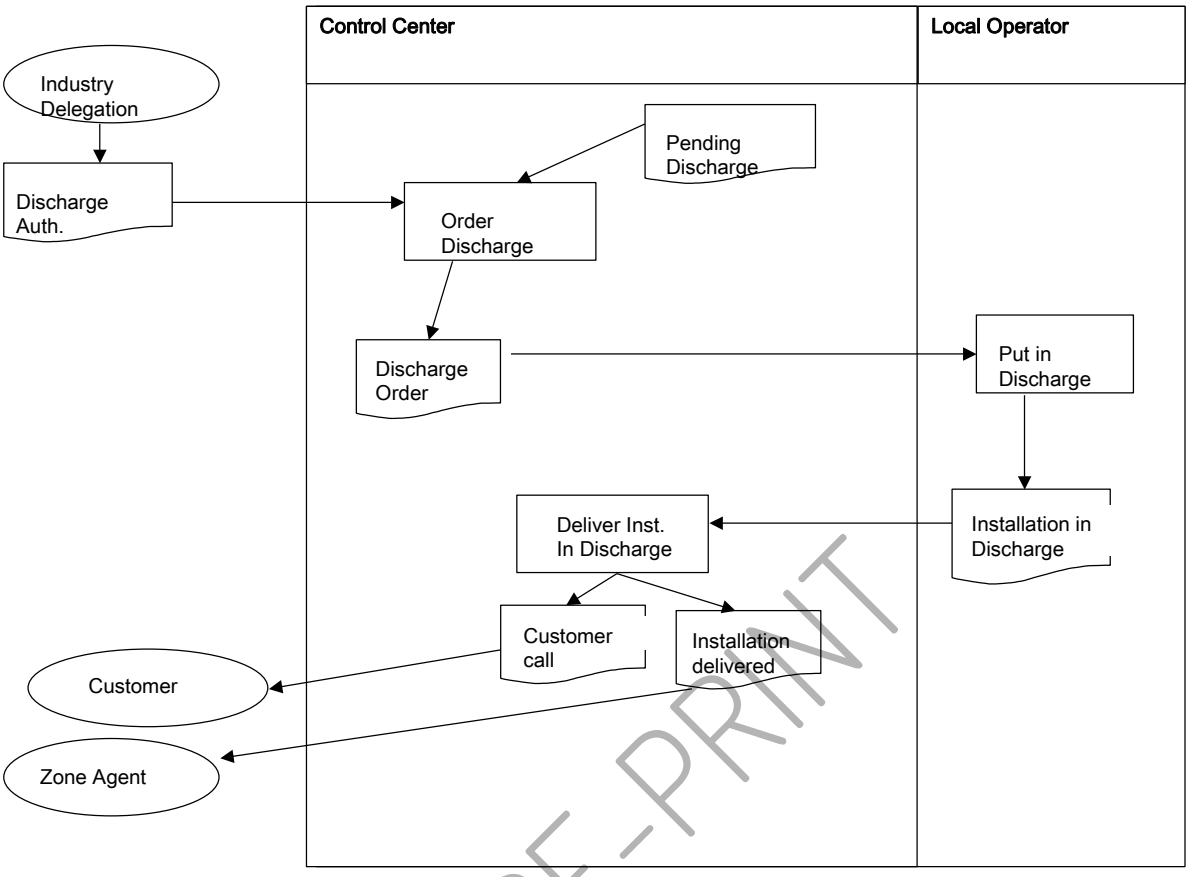

Figure 3. A second fragment of the DocFlow of figure 2. The Document "Pending Discharge" connects the two DocFlows

On the other hand, the so-called temporal Tasks (controlled by temporal events) should be examined in detail and the possibility that they could be controlled by a Document should be analyzed. Finally, the Tasks-Internal Agents association can be changed by looking for greater agility in the circulation of documents (in the case of our tool, by simply dragging a Task from the column corresponding to one Internal Agent to another). We can even ignore the Internal Agents as independent subsystems. Thus the analysis of basic aspects of the workflows can be limited, reintroducing, at a later stage, the enterprise's organization that can be modified in parallel. The aim of this analysis is to remove the organization constraints of the old system and it allows the process to be frequently simplified, unifying Tasks and improving the system.

In short, following the study, we obtain a set of DocFlows with different alternatives marked, above all, by the automation frontiers that have been defined in each case and with the study plans, according to our consideration of the presence or not of the Internal Agents associated with the Tasks.

3. Transformation of DocFlows into Use Cases

All the possibilities mentioned in the previous section are included in what can be considered as workflow analysis and business process reengineering. Although this analysis is of great interest in itself, the fundamental proposal of this work states that very valuable information for the modelling of systems can be obtained from the DocFlows, independently of the development paradigm used. We propose a transformation of the DocFlow to specific diagrams for the two main development paradigms: use case diagrams for OO development and DFDs for structured development. This section analyzes the transformation to use cases and the next one, to DFDs.

For each selected alternative a set of events and associated responses of the system can initially be obtained. It is sufficient to apply an algorithm for traversing the diagram. In order to find the external events we must locate each incoming Flow toward the system and its associated (external) Document.

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 an External Agent (there will be one or several Documents that leave the system) is reached. An event doesn't have a unique linear path associated to it. Any Task that generates more than one output Document will originate a tree or a series of alternate paths.

The internal structure we use to represent a DocFlow is a hypergraph [2], where the following equivalences are established:

- The External Agents, Documents and Tasks are three types of nodes.
- The Flows are arcs or, in general, hyperarcs. If there is an AND-join or AND-split Task, the set of AND Flows are represented as a hyperarc. In all other situations, each Flow is a simple arc.

With this equivalence, the path analysis of a DocFlow diagram becomes a directed hypergraph problem, already described in the literature [10]. We shall now explain, informally, the process to be followed once the system frontiers have been defined.

Algorithm 3.1:

1. The external events corresponding to each Flow that enters the system (crossing the frontier) and its associated external Document are located.
2. For each external event located:

   Using the visit procedure described in [10], obtain the set of nodes connected through a hyperpath with the node corresponding to the external Document. Each hyperpath defined by the connected

Tasks and Documents is followed in the direction of the Flows, taking into account the existence of bifurcations, until an External Agent is reached.

Step 2 requires more detail. The existence of bifurcations will provoke special situations and possibly the appearance of new hyperpath. Given a hyperpath begun in an external Document, each Document or Task found can be from any of the following types, from the point of view of the Flows entering (2.1) and exiting (2.2 to 2.4):

2.1. If an AND-join Task is reached through any of the Documents that enable it, incorporate all the Documents belonging to the tail of the hyperarc.

2.2. If it is a Document or Task with a single outgoing Flow, add that Document or Task to the hyperpath.

2.3. If a bifurcation due to an OR-split type Task or a Document used by two Tasks is reached, duplicate the hyperpath followed up to that point and complete the two hyperpaths independently. If the possible hyperpaths are more than two, the generalization is immediate. The existence of cycles must be considered, and to do so the Flow selected for the division of the hyperpath is marked and the cycle is not covered again.

2.4. If a bifurcation due to an AND-split Task is reached, they are simultaneous paths, and both branches contain the only response of the system, unlike the situation described in the step 2.3.

All the possible paths are obtained using this algorithm (in reality, a set of sub-hypergraphs). Starting from each event all the external Documents that come out of the system should be reached as, otherwise, the diagram is not well constructed. The result is a list of system responses (or, in other words, a list of scenarios) that include:

- The event that triggers the response (arrival of an external Document)
- Input Documents used by the Tasks included on the hyperpath
- Actions that must be carried out (the list of the activities of all the Tasks included on the hyperpath)
- Output Documents produced by the Tasks included on the hyperpath

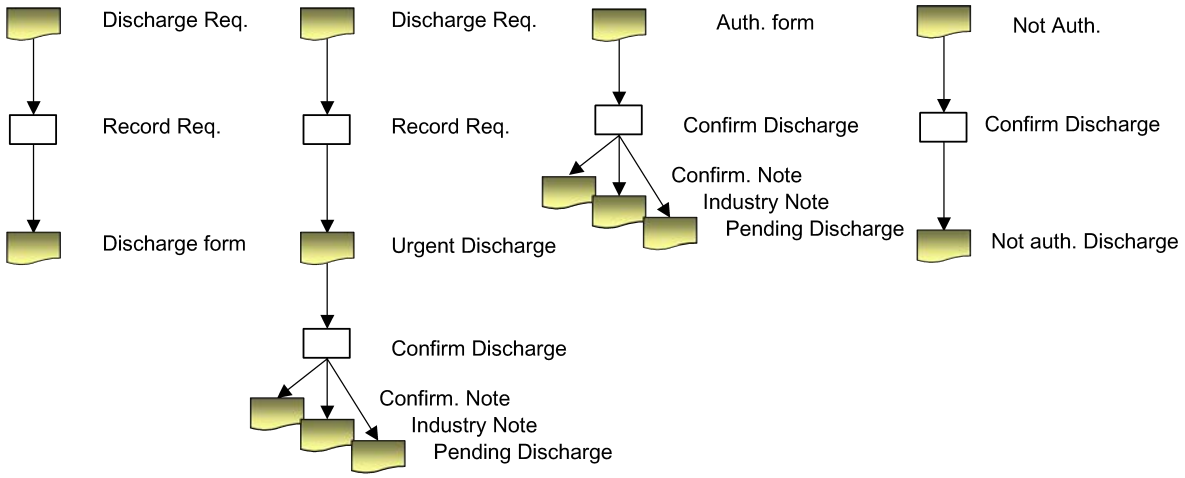

Figure 4. Set of resulting hyperpaths from the DocFlow of the case study (figure 2), after application of algorithm 3.1

It should be pointed out that an external input Document may occur several times in the list, one per possible response. On the other hand, several hyperpaths may generate the same output of the system. This list of scenarios of the system should serve as a validation of the system's behaviour, that is, each of the possibilities obtained should represent a real scenario of the desired system.

Figure 4 shows the set of resulting paths from the DocFlow o figure 2. In this version the Manager is considered as an external agent and this choice conditions the number of hyperpaths.

In object-oriented development, the transformation of the scenario list into a group of use cases that show the functionality of the system is needed. Different actors can be generated using the information we have about the Internal Agents of the system. Every Internal Agent that receives the external Document controlling the corresponding event will become the primary actor. For the description of each use case, the list of activities of the related global Task is used. On the other hand, the external actors become the actors of the business use cases (BUC), according to the proposal of Jacobson [13]. Consequently, two complementary transformations are proposed to obtain the business use cases and the use cases themselves.

Transformation into business use cases:

The transformation of a DocFlow in the corresponding business use cases diagram takes advantage of the results of the algorithm 3.1, which permits all the possible responses to an external event to be identified:

- Each External Agent or Internal Agent that remains outside the system (if and only if it is the direct receiver or transmitter of an external Document) becomes an external actor of the BUC.
- Each external input Document implies the existence of a business use case whose initial description consists of the Tasks connected to that Document, following any of the possible hyperpaths detected in the algorithm 3.1. The External Agent, origin of the initial Document, will be the primary actor of the BUC. The receiving Agents of external output Documents become secondary actors of the BUC.

Due to the level of granularity of the BUC, the diagram obtained only gives us a bird's eye view of the system's functionality. On the contrary, the detailed diagram of use cases provides us with much more information.

Transformation into use cases:

Applying the results of algorithm 3.1 and factorizing the initial result achieve the transformation of a DocFlow into a detailed use case diagram:

- Each Internal Agent considered apt for automation in the outgoing DocFlow becomes a possible actor with the same name (Administrative, Manager, etc.).
- Each possible answer by the system to a particular event (that is, precisely the result obtained by applying algorithm 3.1) becomes a single use case, whose initial description will be the hyperpath (Documents and Tasks) this answer follows. The list of activities, linked to each original Task included in the hyperpath, can finally be used to describe the use case.
- The actor assigned to each use case is initially the Internal Agent of the DocFlow that receives the Document associated with the external event. We are obviously dealing here with a provisional assignation subject to revision, due to the focus that ignores the current organization.

Following this if, when comparing the descriptions of the hyperpaths, final fragments can be identified that are common to more than one use case, then use cases that include others can be factorized and defined. If the common fragment existed as the description of an isolated use case, it would then have its own associated actor; otherwise, it will be an abstract use case. On the other hand, if we are dealing with initial common fragments of those hyperpaths, then it is a use case with an extension point and several alternatives. The first case comes from OR-join or AND-join Tasks and the second from OR-split Tasks. Parallel hyperpaths resulting from AND-split Tasks (that did not generate hyperpath duplication) do not give rise to this kind of situation. In any case, the possible "extend" or "include" relationships with other use cases will remain in the old use cases or in the new ones depending on the point of connection (Document or Task).

The factorization algorithm, with the alternatives applied in strict order, will be as follows:

Algorithm 3.2:

Given the set of use cases generated while some duplicated fragment remains:

1. If two complete use cases can be found that are defined by the same hyperpath, they become a single use case, respecting all the possible relationships with other use cases and actors involved.
2. Otherwise, if two common initial fragments (that include at least one Task) are found, the common part becomes a single use case that is extracted from the old ones. The two original use cases (without the common part) will now be related to the new use case by the "extend" relation (see figure 5.b).
3. Otherwise, if two common final fragments are found, the common part becomes a single use case. Each part obtained should have at least one Task. The two original use cases (without the common part) will now be related to the new use case by the "include" relation (see figure 5.a).

The algorithm ends when no more situations like those described in 1, 2 and 3 can be found.

Should there be loops in the original DocFlow (built with OR bifurcations), then a situation such as that shown in figure 5.c is reached, in which a use case includes (or is extended by) itself. In such a situation, the auto inclusions

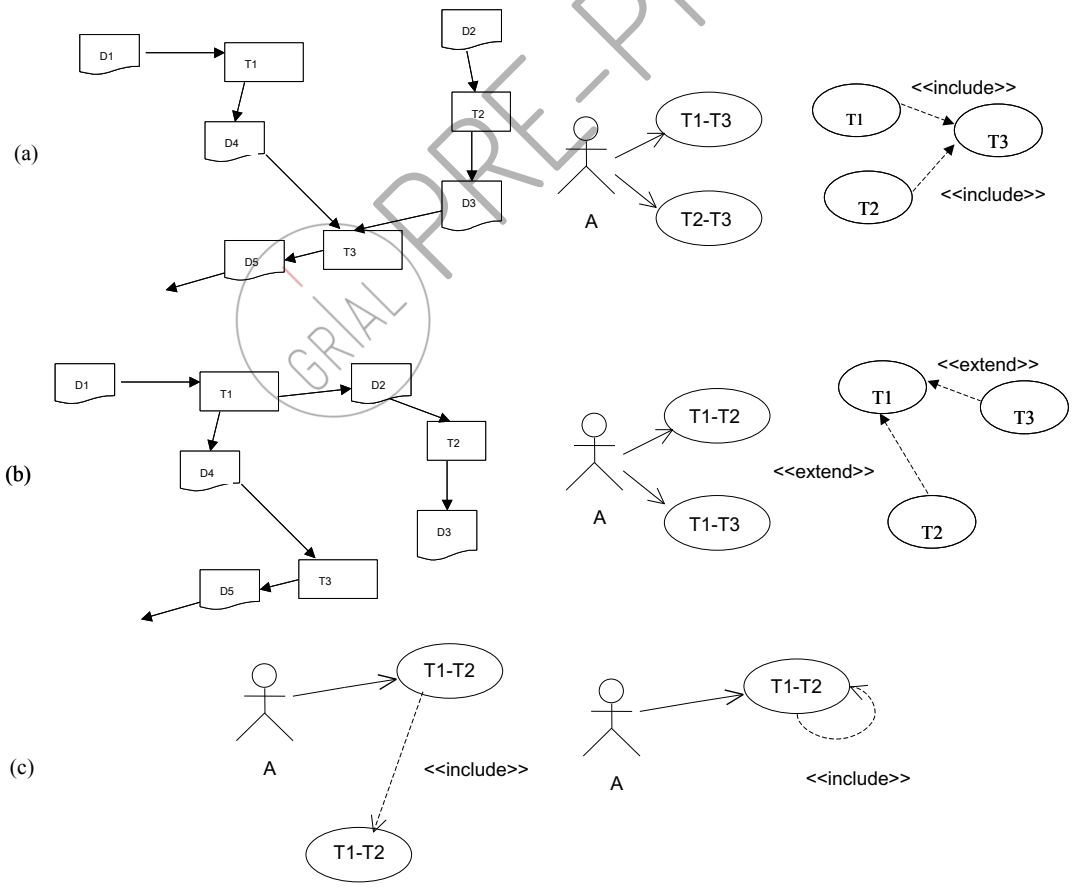

Figure 5: DocFlow Transformations in Use cases (a) include; (b) extend; (c) auto-inclusion

can be eliminated, as they are not linked to the functionality but to the problem solving process.

The application of the algorithm to the case study originates the case use diagram shown in figure 6. There are a common initial fragment (Task Discharge Req.- Doc. Record Req.) that is transformed in the “extend” part of the diagram. A final common fragment (Task Confirm Discharge - Docs. Confirm. Note, Industry Note, Pending Discharge) causes the “include” part of the diagram.

If, alternatively, the organization shown in the original DocFlow is recognized as stable, then the use cases can only accumulate Tasks corresponding to a single Internal Agent. This means that the Tasks included in a use case obtained from the above transformation breaks down into at least as many use cases as actors involved (the hyperpath described by the use case is followed and each time the Internal Agent changes a new use case is generated). The relation between these use cases that appear on travelling along an answer-path is solely of a temporal nature. Thus, a "precede" relation, such as that proposed by Rosenberg [17], which has not found too much echo, can be used. The same effect can be reached by simply adding a precondition to the following use case that makes a reference to the fact that the previous use case should have been completed beforehand.

This first outline allows later design changes to be made but which should be left in the hands of the requirement engineers. In any case, it should be pointed out that if the External Agents have an immutable equivalence with the external actors of the BUC (they are beyond our reach), in the use cases of the system the equivalence is between Internal Agents and actors who are to handle that system, and in this case, they can change their way of working as a consequence of the implantation of the new system. On the other hand, there is a clear difference of style between what is a list of activities ("issue an invoice", "calculate the salary", etc.) as defined in a DocFlow Task and the textual description of the steps of a use case ("the system should issue an invoice", "the system should calculate the salary", etc.). This problem should be dealt with by using appropriate patterns to force the use of certain linguistic constructions and to facilitate the automatic passage from some patterns to others. In this sense the technique for describing requirements proposed by Durán [8], supposes a valid starting point for the conversion. The maintenance tool for use cases proposed has a format for use case description that can be used to automatically generate the textual descriptions of use cases (based on patterns) associated to the corresponding diagrams.

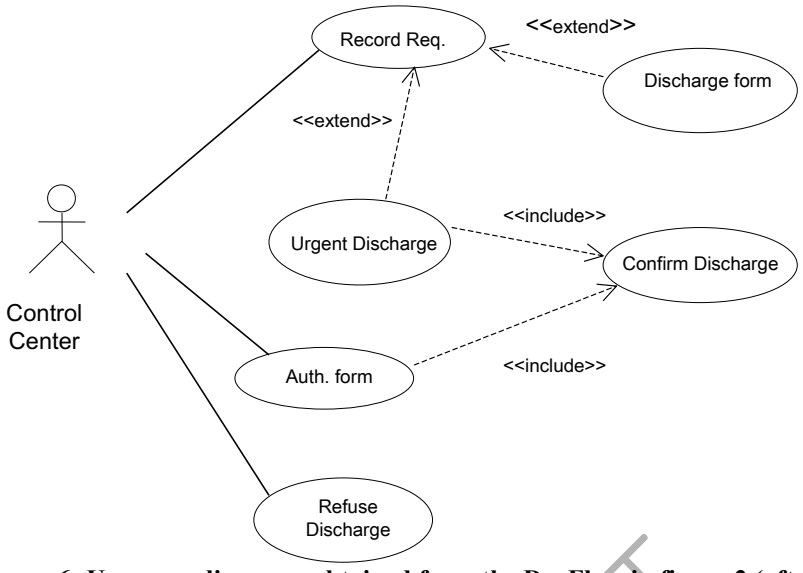

Figure 6: Use cases diagrams obtained from the DocFlow in figure 2 (after application of algorithm 3.2)

4. Transformation of DocFlows into DFDs

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 (similar to the one we have already obtained) and a first rough version of the essential model [23], expressed as a context DFD and a first level DFD. If several automation alternatives are considered, a different essential model will be obtained for each DocFlow.

The outline of the transformation to the context DFD will be as follows:

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

Figure 7 shows, graphically, the equivalence between External Agents (or Internal Agents not included in the system) and external entities, as well as the relation of the external Documents with the flows of the context diagram. Carrying out this transformation automatically leads to the context DFD of the figure (in this case the second version of the two possible variations mentioned above has been chosen: The Manager has been declared an Internal Agent that cannot be automated and is thus outside the frontiers of the system).

The list of events and responses (obtained in section 3), together with the context DFD make up the first model proposed by the structured analysis method. In order to take advantage of all the information gathered in the DocFlows, we propose two possibilities for obtaining the behaviour model, depending on whether or not the Internal Agents are taken into account.

To generate the initial behaviour model an algorithm similar to that mentioned for obtaining the list of responses in the previous section is used, with some variations such as the process-Task association, the use of graphs (as a particular kind of hypergraphs), and the consideration of the temporal events:

Algorithm 4.1:

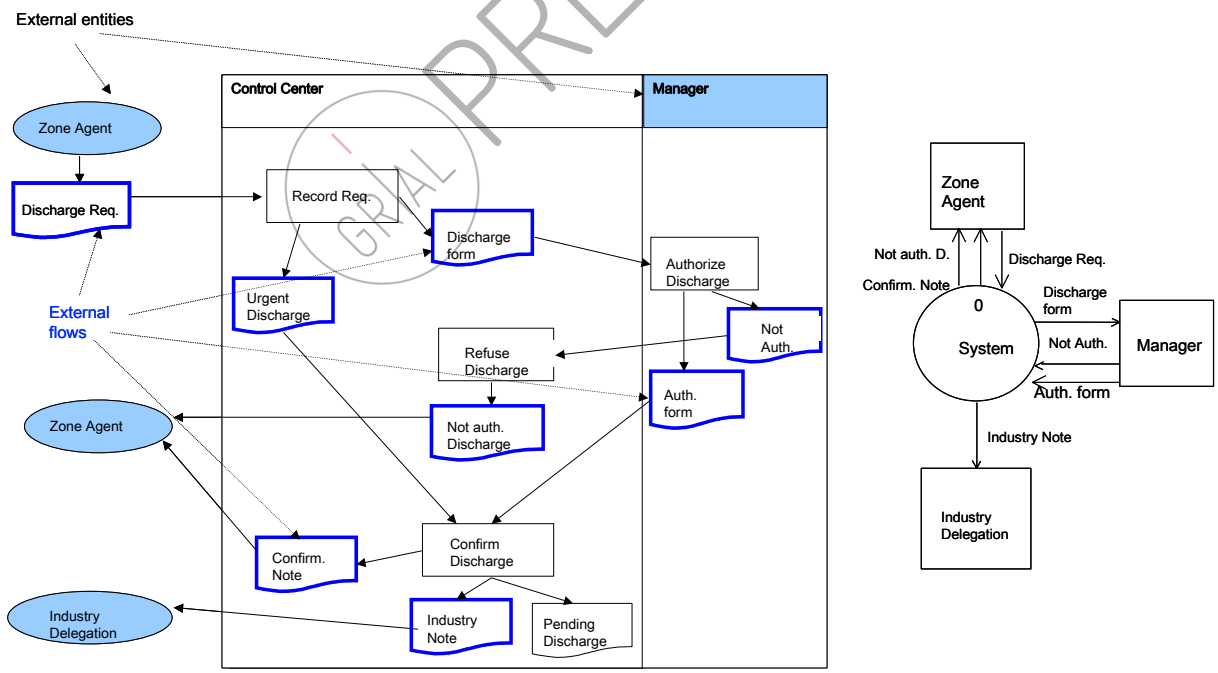

Figure 7: Automatic transformation of a DocFlow into a context DFD . The external Documents are transformed in flows and the External Agents in external entities of the context DFD

1. Every External or Internal Agent that remains outside the system (if and only if it is a direct receiver or emitter of an external Document) becomes an external entity of the DFD.
2. Find the external events by locating each Flow entering the system. Each external Document becomes an entering flow, associated to the corresponding external entity. The information included in the external Documents is used to document the corresponding flows. Each external event begins a process that initially includes the receiving Task (which is associated to the new process).
3. Locate the Tasks caused by a temporal event. Each of them originates a process in the DFD that initially includes the temporal Task, which is associated to the new process.
4. For each process discovered in the steps 2 and 3, follow the DocFlow, starting from the Task initially identified with the process and following the path marked out by the connected Tasks and Documents. Each Task or Document considered is marked to avoid duplications, associating it with the process in which it is included. The information on the process includes the list of activities of each of the Tasks incorporated in it and the original path that originated the process.
5. The course ends when, on each of the possible paths, a Document (or Task) already associated to a process or External Document is reached (it becomes a flow that ends in the external entity corresponding to the External Agent that receives the Document).

Step 4 should be examined in more detail: According to the type of element found, the following situations can be produced:

4.1. If it is a Document or Task with a single output Flow, the Document or Task found is incorporated into the process.

4.2. If a bifurcation is reached due to an OR-split Task or a Document susceptible to being used by two or more Tasks, complete the two paths independently, applying step 4 of the algorithm to each of them recursively. This way of acting implies adding the Tasks found in both paths to the initial process while possible.

4.3. If an OR-join Task or a Document generated by more than one Task is reached, it is possible that the Task or Document has previously been incorporated into a process (it will be marked with the associated process). In the first case the two processes become a single process and the path is completed. Otherwise, it continues as in step 4.1.

4.4. If a bifurcation due to an AND-split Task is reached, the Task is included in the process being built but, connected to it, two or more data stores are generated (one for each outgoing Document). The new data stores and their output flows represent the starting point of two paths that run in parallel. It

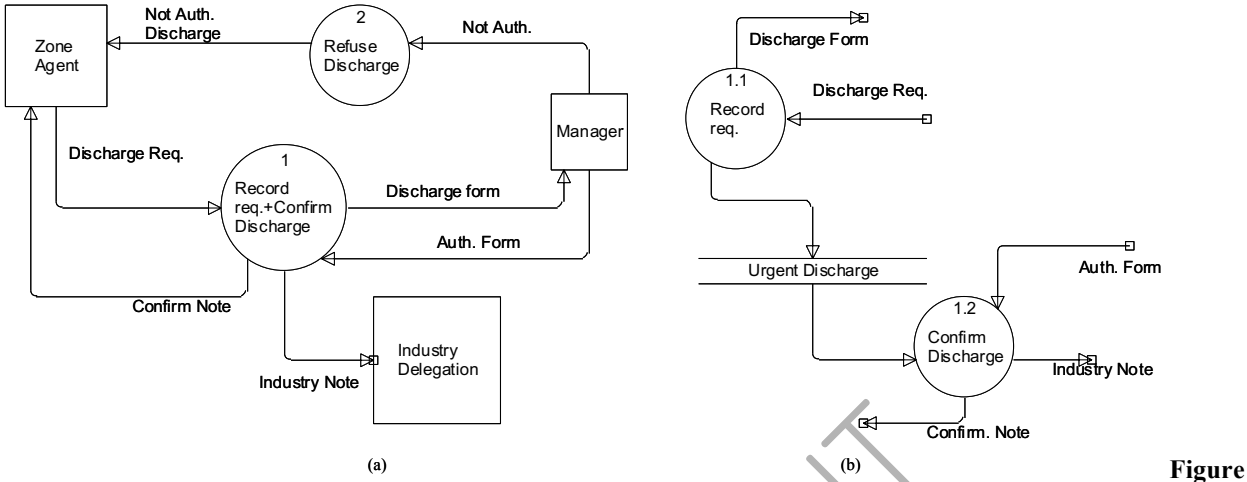

Figure 8: DFDs obtained from the DocFlow of the case study: (a) First level DFD; (b) explosion of procces 1(Record Req.+Confirm Discharge)

related DocFlows. In order to study them separately it is necessary to use External Agents that pick up the Documents (in which case they represent independent subsystems).

- Each process keeps the information concerning the original Tasks and subtasks as well as that concerning the Documents eliminated in the transformation.

Figure 8 shows the DFDs obtained from the DocFlow of figure 2 and related to context DFD of figure 7. Initially, the DFD of Figure 8.a is obtained. The explosion of the process "Record Req.+Confirm Discharge" can be seen in Figure 8.b

Alternatively, the algorithm can be forced in such a way that it respects the current organization, making no changes. In this case it is necessary to consider the Internal Agent associated to each Task and only those Tasks that are carried out consecutively by the same Internal Agent can be remodelled. To do this, algorithm 4.1 is modified in step 4: If a Task with an associated Internal Agent different from the Agent associated to the process under consideration is reached, action taken is as in step 4.5, generating an intermediate data store. In reality, applying this variation of the algorithm is equivalent to exploding the processes obtained in algorithm 4.1, following, for orientation, the current organization (included in the information of each process). Figure 8.b shows the explosion of process 1 of 8.a.

This last possibility only makes sense if a) reengineering work has previously taken place on the business processes and the organization has reached an optimum point or b) when the organization is reasonably stable (the normal case

with governmental organizations, the armed forces, etc.). In cases other than those mentioned, it is more practical to begin the analysis from a more open situation (provided by algorithm 4.1).

The result of this transformation does not represent the final version of the analysis of the new system, and neither is it what this proposal is trying to do, but it does contain sufficient elements to begin the work of specification, in the strict sense, with fairly good guarantees.

5. The DocFlow Tool

After explaining the creation and transformation of DocFlows, 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 structured or OO CASE tools. 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 scenario list could be a simple text or XML file and the DFD representation format will be CDIF [9] and a format from a tool widely used in the industry, such as Easy CASE is, also used by us in the Software Engineering courses. In OO we have chosen XMI [15] (as independent tool) and Rational ROSE [16] as target formats of our translation. The reason is evident, due to the wide diffusion this tool has, especially since the standardization of UML. The possibility of using ROSE for exporting and importing diagrams or portions of diagrams makes it relatively simple to generate XMI files with the necessary information to build ROSE models. Although internally it is possible to use a representation format to include all the information that is of interest, it is also necessary to include in the tool the utilities that allow the workflow diagrams to be imported and exported from and to some of the standard definitions of more widely known workflows tools [19].

On the other hand, the tool that is being developed should use already existing techniques as well as the power provided by the hypertext links with other tools such as screen formatters, text processors, Web page generators, etc. All the information that can be structured in the exchange of ideas with the clients can be incorporated into auxiliary files. Firstly, each Document (in particular those that are filed and the external ones) should be defined with the greatest possible accuracy, using some kind of pattern, so that its description can be automatically converted into an information requirement. The Tasks should be described in a similar way using a pattern that describes the activities included in each Task in a tabular format. The abovementioned proposal for describing requirement from Durán [8] can be used as a guide. The automatic storage of the obtained models in a Reuse Repository is also desirable.

Another interesting use is the description, in sketch form, of the user interface for each external entering Document (corresponding in general terms to a future information capture) and of the queries and reports associated with the external outgoing Documents. Once again, rather than overload the tool, what is needed is a connection with other already existing tools.

The scope of the project, whose final objective is to develop an attractive tool that can be applied to industrial purposes, has led us to consider a development in stages, priority being given to essential aspects. In this sense, the tool for building, handling and validating workflow diagrams is the starting point. The second stage concentrates on obtaining DFDs and diagrams of use cases by applying the algorithms of sections 3 and 4. The rest of the proposed aims are medium term goals. Finally, we must build a multi-platform tool that can be used in Windows and Unix systems.

A previous version of the prototype was developed for Windows [14] and the implemented functionality includes the creation, maintenance and validation of the syntactic correctness of the DocFlows. As additional results, only the first proposal (events list of the system) was 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 on a new version of the tool that will be able to generate the diagrams proposed in this work. The available current version [6] was developed in Java so it can be used on any platform. It can apply algorithms 3.1 and 3.2 and generate use cases that the tool itself can handle and keep in XMI format (Rational ROSE or other commercial tools can import them without difficulty). Figure 9 shows a working image of the tool. As it is possible to make fast changes to the Task assignation of the DocFlow among different users of the system (and the corresponding implications of these changes can be reflected immediately in the corresponding use cases), the time saving is undeniable. The tool is available from the Web page of the group GIRO (Research Group in Reuse and Object-orientation), accessible at http://giro.infor.uva.es.

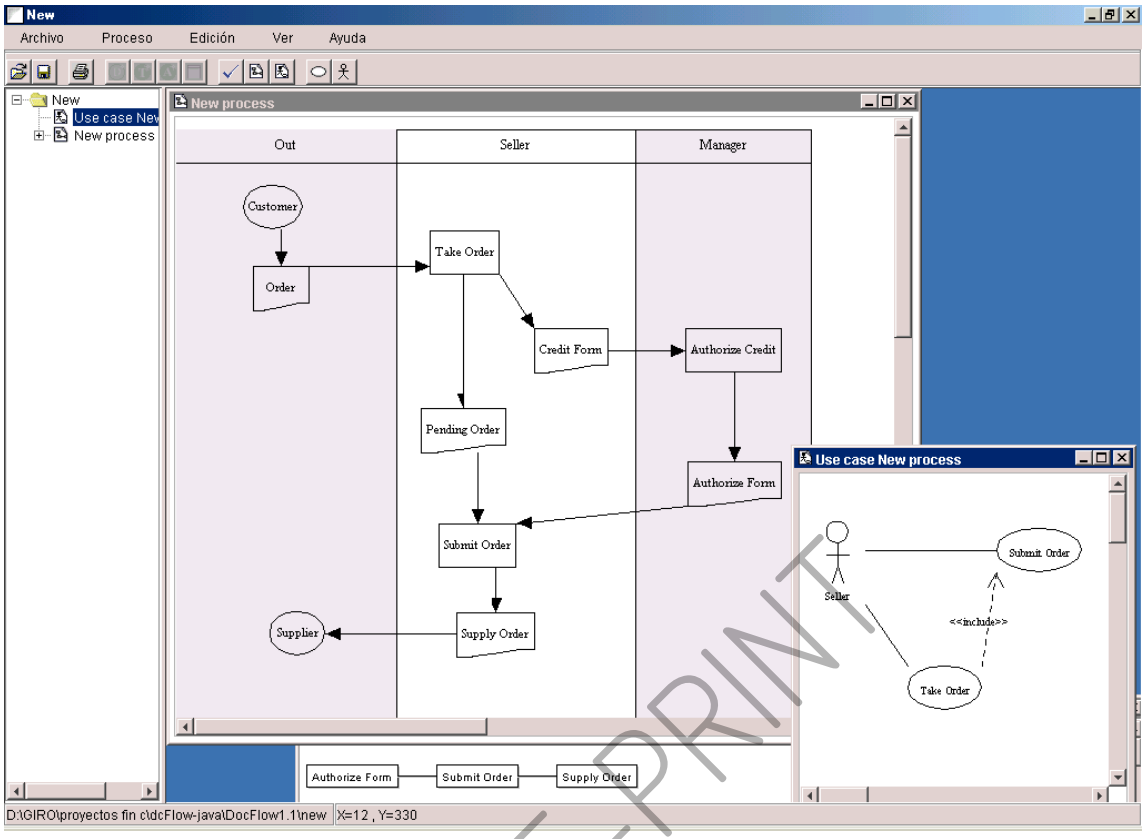

Figure 9. An image of the DocFlow tool with work in progress

6. Concluding remarks

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. The proposal consists of a procedure to accurately define the functionality of the new system, starting from the current work method, expressed as a kind of administrative workflow. We had presented two algorithms to automatically generate profitable results in the form of DFDs or use cases. The use of hypergraphs well known algorithms reinforces the transformation process and facilitates its automation.

The workflow technique is easy to understand for the end users and, at the same time, allows the partial information received from them to be validated. The automatic generation of uses cases eliminates one of the main problems in requirements elicitation. Another advantage of the approach is the simulation of several design scenarios regarding

the borders of the new system. For each possible scenario, a series of results can be generated, mainly initial DFDs or use cases, according to the chosen development paradigm, with evident advantages.

As a final result, now in the refinement phase, a prototype of a CASE tool has been built that implements many of the ideas developed. As future work, it is necessary to enrich the functionality of this prototype, so that we can generate not only event lists or basic versions of use case diagrams, but also complete and validate versions of DFDs and use case descriptions. The automatic storage of the obtained models in a Reuse Repository is also pending. In a more general sense, increasing the information about Documents in the initial workflow diagram will allow to add details to the generated diagrams.

Acknowledgements

This work has been partially financed by the CICYT (TIC2000-1673-C06-05) as a part of the DOLMEN project.

Cited references

1. E.V. Berard, Be Careful with Use Cases. Technical report. http://www.toa.com/pub/use_cases.htm (1995).
2. C. Berge, Hypergraphs, North-Holland, Amsterdam, 1989.
3. G. Booch, I. Jacobson, and J. Rumbaugh, The Unified Modelling Language User Guide, Addison-Wesley, 1999.
4. A. Collongues, J. Huges, and B. Laroche, Merise. Methode de conception, Dunod, 1987.
5. A. Cockburn, Goals and Use Cases, Journal of Object Oriented Programming (sep. 1997), pp. 35-40.
6. L. Domínguez, Herramienta para el diseño de diagramas Documentos-Tarea, P.F.C. University of Valladolid, 2001, available in http://giro.infor.uva.es.
7. O. Díaz, and J.J. Rodriguez, On use case elicitation, in Actas de las III Jornadas de Ingeniería del Software, Murcia, Spain, 1998.
8. A. Durán, B. Bernárdez, A. Ruiz, and M. Toro, A Requirements Elicitation Approach Based in Templates and Patterns, in WER'99 Proceedings, Buenos Aires, Argentina,1999.
9. Electronic Industries Association, CASE Data Interchange Format, EIA/IS-106, http://www.cdif.org, 1996.
10. G. Gallo, G. Dongo, S. Nguyen and S. Pallotino, Directed Hypergraphs and applications, Discrete Applied Mathematics, 42(1993) 177-201.
11. M. Hammer, and Champy, J., Reengineering the Corporation: A Manifesto for Business Revolution, Allen and

Unwin, 1993.

12. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, Object Oriented Software Engineering, Addison-Wesley, 1993.
13. I. Jacobson, M. Ericsson, and A. Jacobson, The Object Advantage - Business Process Reengineering with Object Technology, Addison-Wesley, 1994.
14. M. A. Laguna, J. M. Marqués, and F. J. García, A user requirements elicitation tool, ACM SIGSOFT Software Engineering Notes, 26:2 (2001) 35-37.
15. OMG, XML Metadata Interchange (XMI), OMG Document ad/98-10-05. Available in http://www.omg.org, 1998.
16. Rational Software Corporation, Rational Rose, http://www.rational.com, 2001.
17. D. Rosenberg, Use Case Driven Modelling with UML: A Practical Approach, Addiso-Wesley, 1999.
18. M. B.Sánchez Alonso, Gestión de intervenciones en red de Iberdrola. P.F.C., University of Valladolid, 1999.
19. M. T. Schmidt, The evolution of workflow standards, IEEE Concurr., 7 (1999) 44-52.
20. H. Tardieu, La Methode Merise, Les Editions d'Organization, 1983.
21. WfMC, Workflow Management Coalition Terminology and Glosary (WFMC-TC-1011), Technical report, Workflow Management Coalition, available in http://www.wfmc.org, 1996.
22. A. Wagner, Prototyping: A day in the life of an interface designer, in The Art of Human-Computer Interface Design, Addison-Wesley, 1990.
23. E. Yourdon, Modern Structured Analysis, Prentice-Hall, 1991.