End User Programming Model for Workflow Web Based Systems

Cecília Kremer Vieira da Cunha[1] *Clarisse Sieckenius de Souza*[1]
Violeta S. T. D. B. Quental[2] *Daniel Schwabe*[1]

[1]*Departamento de Informática*
[2]*Departamento de Letras*
PUC-Rio
R. Marquês de São Vicente 225 – Gávea
Rio de Janeiro – RJ – 22453-900
email: [ceciliak, clarisse, schwabe]@inf.puc-rio.br,
violetaq@let.puc-rio.br

PUC-Rio InfMCC17/99 Agosto'99

Abstract

This paper presents the current stage of development of an End User Programming (EUP) model for Workflow Web Based Systems. The model is described and applied to a real system extension scenario. It is possible to see how the usage of this model allows the task of application maintenance to be shared among professional programmers and empowered end users. Our future step will be to relate this work to other EUP proposals and to describe the semiotic aspects that guide the development of this model.

Keywords

End User Programming, Workflow Systems, WWW Development and Maintenance, Augmented Transition Network (ATN)

Resumo

Este trabalho apresenta o estágio atual de desenvolvimento de um modelo de Programação para Usuários Finais (End User Programming - EUP) para sistemas de Workflow baseados em ambiente WWW (World Wide Web). O modelo é proposto e aplicado a uma situação real de extensão de sistema. É possível verificar como o uso deste modelo permite que a tarefa de manutenção de aplicações seja dividida entre programadores profissionais e usuários finais autorizados. O próximo estágio será o relacionamento deste trabalho com outros da área de EUP e a descrição dos aspectos semióticos que norteiam o desenvolvimento deste modelo.

Palavras-chave

Programação para Usuários Finais (EUP), Sistemas de Workflow, Desenvolimento e Manutenção de sistemas na WWW, ATN

Sponsored by CNPq, TeCGraf and Petrobras

Introduction

One of the main contributions of EUP studies is the acknowledgement that it is hardly possible that one single software manages to solve every specific problem a particular user has in a determined domain [Myers]. Moved by this claim, we have decided to develop an EUP environment for Qualitas, the Workflow Web Based System we are working on. We have seen that a rich information representation is needed in order to support the EUP environment, which turned our focus at this first moment to the development of a model to support EUP environments for this kind of system.

This paper reports the current state of the development of this model. It also exemplifies a real extension situation in which it could be used. For didactic reasons, we will begin with a presentation of this possible extension scenario, applied to Qualitas domain. Then we will move on to the explanation of the EUP model itself.

An Extension Example applied to Qualitas

The main purpose of Qualitas is to assist its users to follow the department's policies that guarantee ISO quality standards. According to these policies, some of the employees' activities are seen as "service provisions". To guarantee the quality of each service, an employee - a technician - basically has to:

1. Draft a service proposal, after a customer requests a service
2. Present the proposal to the customer (a negotiation can be involved in this step, causing several versions of a proposal to be presented, but this has been abstracted away here in order to ease the understanding of our example)
3. Turn the proposal into a Contract, after the customer has accepted the proposal
4. Prepare for the execution of the service documented in the Contract
5. Execute the service
6. Document the Customer evaluation about the service execution

Note that all these activities are performed in Qualitas by the technician. This means that every time a customer has a request, he/she has to get in touch with one of the technicians, who enters the request information into the system. So now let's suppose there is a technician whose phone is broken, or who is having trouble to get his/her job done because the number of phone calls he/she receives is too big. This person would probably want to allow customers to inform their requests directly into system. So, let's see how he/she could try to extend Qualitas to reach this goal.

The general idea would be to allow a customer to enter some of the proposal information directly into the system. This could be done by copying and then adapting Qualitas' interfaces respective to the steps 1 and 2 listed before.

When entering the Extension Environment, the technician would navigate until he/she finds the interfaces wanted. At first, he/she would choose one business process to work with. The business process is the top level representation of the work processes previewed in our workflow system. The interfaces are the bottom level, as will be seen during the explanation of our Workflow Model later on. So, the technician could navigate in a top-down way until reaching the interfaces desired. The following figures demonstrate one possible navigation path. Figure 1 shows two of Qualitas' business processes.

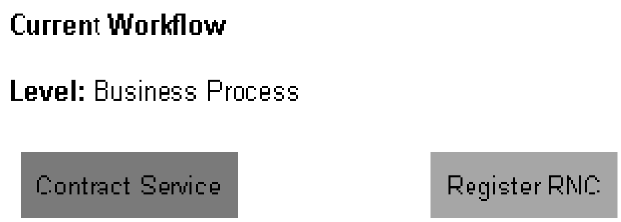

Figure 1. Qualitas' Business Processes

So let's suppose the user chooses the business process "Contract Service". This business process is broken down into a sequence of activities, the Activity Network, as shown in Figure 2.

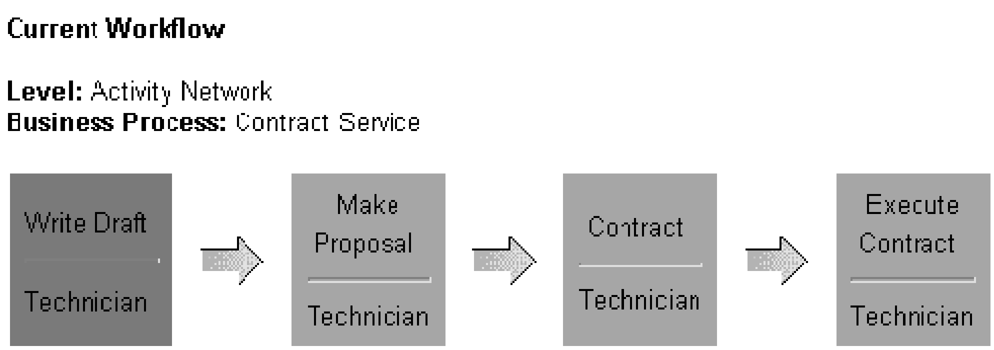

Figure 2. Contract Service Activity Network

Actually, the Activity Network can be much more complex then the sequence represented above, simplified for our example. The representation shows the kind of employee that can perform each activity. For example, the activity "Write Draft" can only be performed by Technicians. It could not be performed by Coordinators, for example.

The user continues the navigation, in order to reach the interface level. He/She chooses the activity "Write Draft", that is sub-divided into the Interfaces presented in Figure 3.

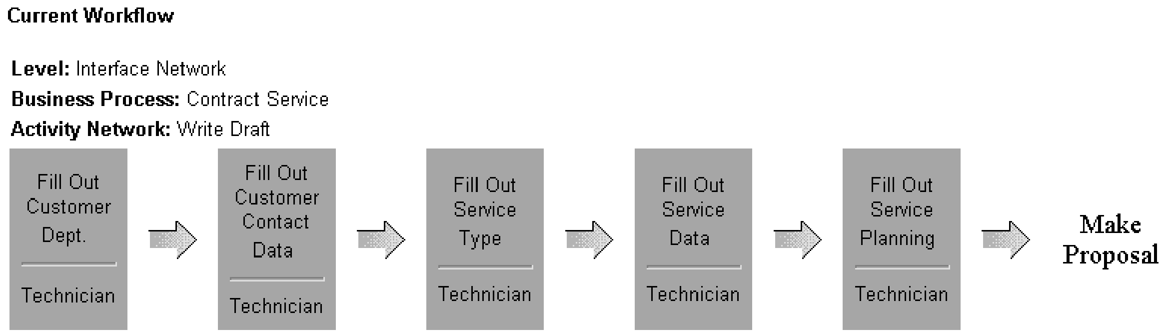

Figure 3. Write Draft Interface Network

Each interface[1] is represented by a box divided into two parts: top and bottom. At the top part, the interface title is shown. This title is the same that appears to the user when he/she is using the application. If this is not sufficient for the user to remember what the interface looks like, he/she can click on the box and the interface look is shown. At the bottom part, inherited from the Activity Network, the user can see the kind of employee that can interact with the interface. The arrows indicate system processing.

Note that the text after the interface "Fill Out Service Planning" is not represented inside a box, because it actually does not represent an interface. It represents a link to the next Interface Network, that actually belongs to the activity: "Make Proposal" (shown in Figure 2).

The user constructs extensions by building a new workflow. This can be done selecting and copying interfaces and arrows (also called actions) in a composition process. The user can also adapt, make changes, to the interfaces or actions during the extension process.

In this example, our user would like to copy the sequence of the three first interfaces and their corresponding actions. Figure 4 shows a schematic view after the user has selected and copied these three heading interfaces: "Fill Out Customer Dept.", "Fill Out Customer Contact Data" and "Fill Out Service Type".

[1] At this point it's important to observe that the interfaces previewed in our proposal are form-based and hypertext interfaces, multimedia and hypermedia interfaces are not included yet.

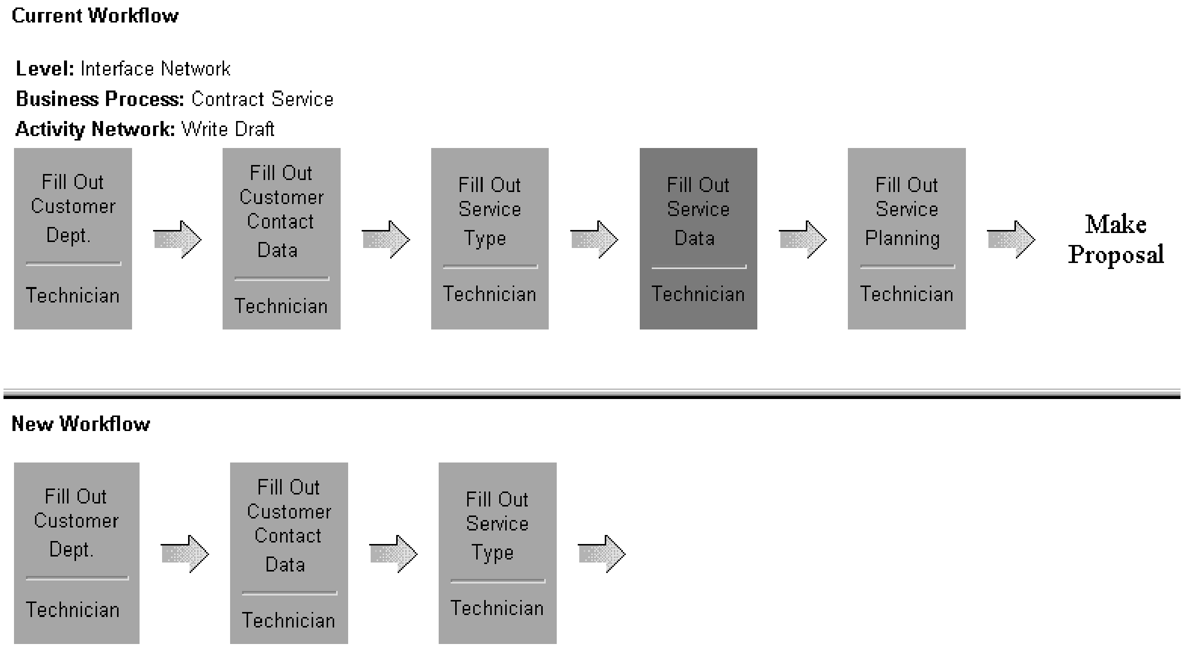

Figure 4. Creating the New Workflow

Our user is satisfied to see that he/she managed to create the new workflow. He/She can already imagine a customer informing, at first, the department, then informing contact data (such as name, phone number and e-mail) and then deciding what type of service is being requested (education, consulting, etc.). But the user also notices that only Technicians can interact with these interfaces (bottom part of each box). So, he/she decides to change this in the new workflow in order to allow customers to access the interfaces. He/She chooses one of the new interfaces and then accesses the interface property **Role**. But the problem is that the allowed values for this property are only: “Technicians”, “Coordinators”, “Chief” and “Administrative Personnel”. “Customer” cannot be indicated as a **Role** that can interact with an interface.

At this point our user has reached a point where the extension task has to be shared with the application developer. From a software engineering perspective, what our user wants is not only to be able to assign a new value to a property, but actually give access to more people to the system, which falls into a security problem.

Our user gets in touch with Qualitas’ developers and they decide that the user can continue extending Qualitas, using the technician as the **Role** for a while, until they build the mechanisms that guarantee customer access to the system.

So, continuing extension, our user’s focus now at the interface “Fill Out Service Data” (see Figure 4). This interface captures details about a service. From our user point of view, the problem is that some of these details cannot be informed by the customer. For example, at this point, in an education service, the technician informs whether slides will be used, what kind of material will be provided to the students, etc.. Thus, our user would like to adapt this interface in order to use it in the new workflow. To do this, the user acts upon the interface and interface fields. Figure 5 demonstrates part of the “Fill Out Service Data” interface. Note that each selected (highlighted) field has, at its right, a set of possible changes that can be done by the extender.

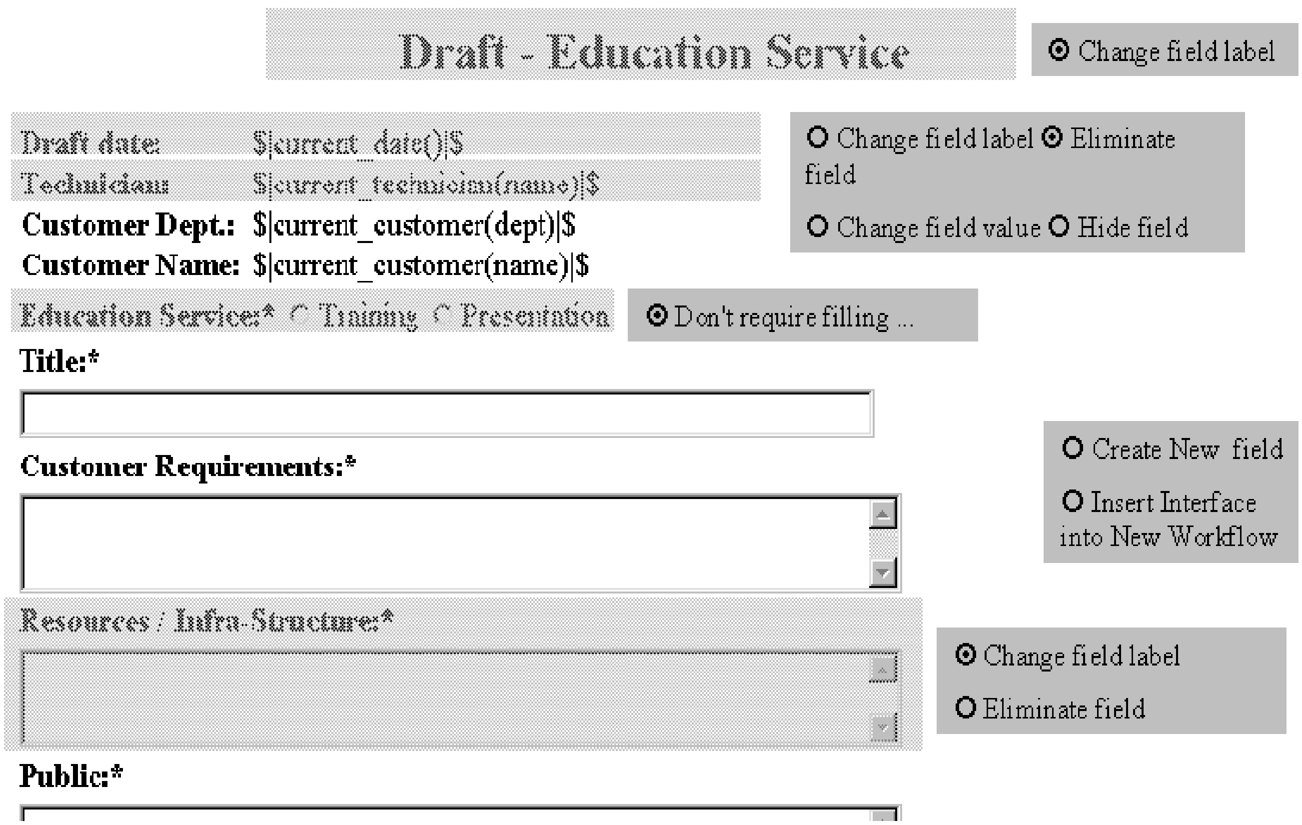

Figure 5: Adapting an interface

After adapting this interface part, it would look like Figure 6. Note that the interface title was changed. The fields “Draft Date” and “Technician” were eliminated. And the input of the field “Education Service” is not required anymore, because the symbol * (that means required field) does not show up in the adapted interface.

Field elimination has an important consequence: it means that the eliminated information will not be available for the actions (in the arrow) that will process this interface later on. The option “Hide Field” gives the possibility not to show the field on the screen, but still have its information available for future processes.

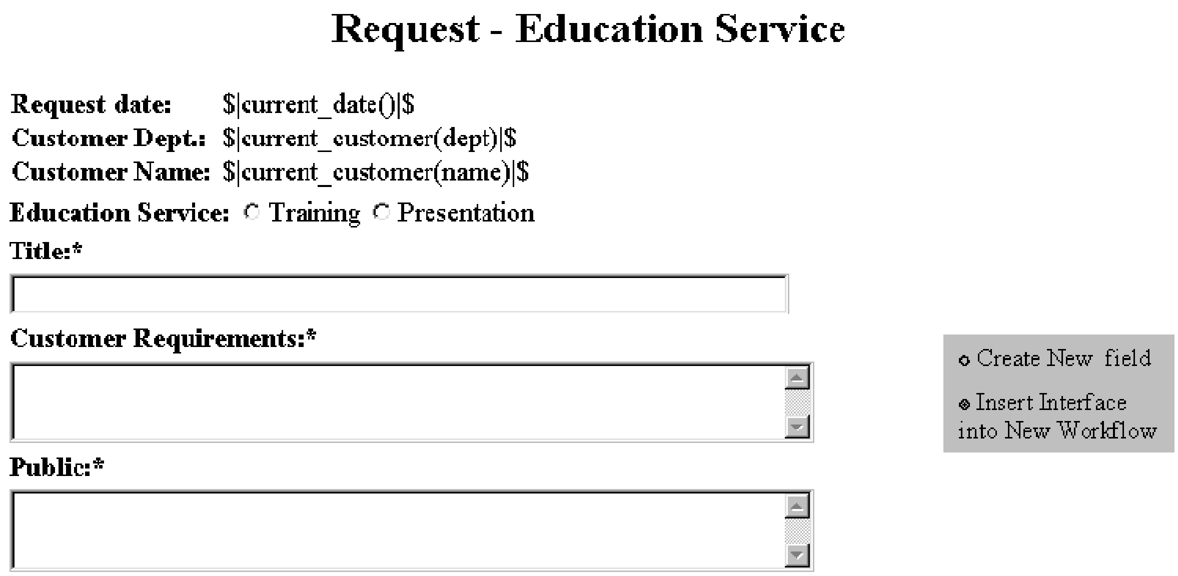

Figure 6: Adapted interface

By changing the interface title, the label that appears in the (last) box of the new workflow is also changed (see Figure 7). Our user will now copy the next action, highlighted in Figure 7.

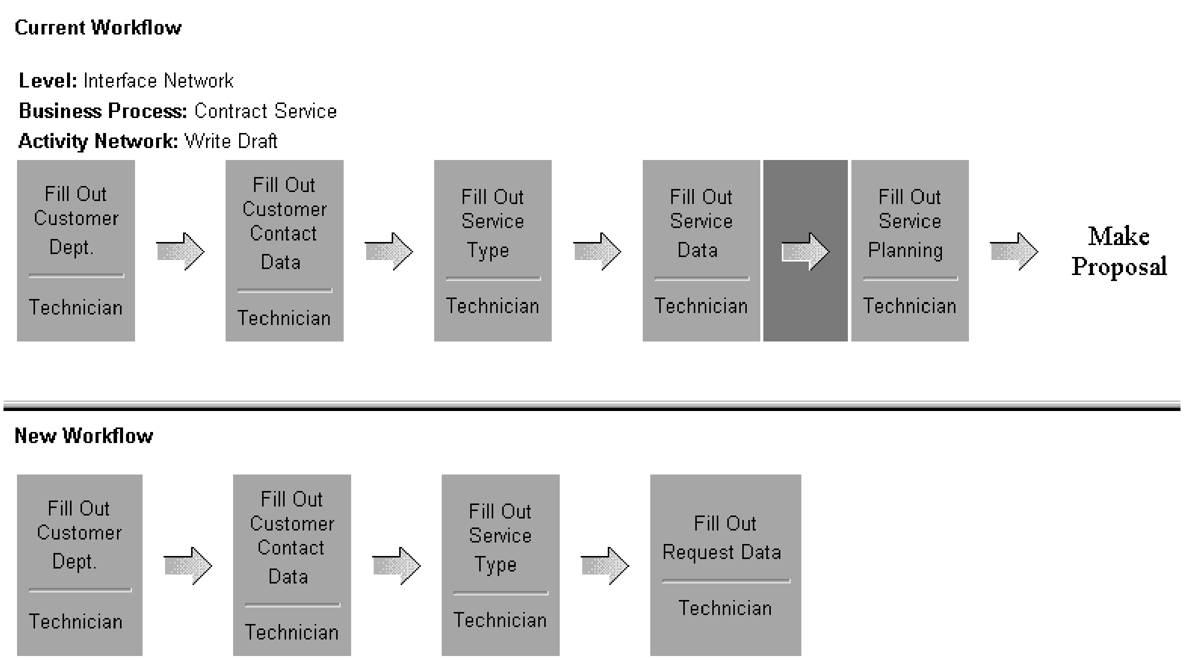

Figure 7: Copying an action

But this action cannot be copied as is into the new workflow, mainly because the information of the eliminated fields is not available anymore. Each step of the action that referred to the eliminated information has to be reviewed. The extension environment guides the user during this revision. It prompts the problems encountered and proposes solutions like this:

This action is executed only if the following rule is satisfied: "There is a technician responsible for the service."

The information "technician" is not available for this action. The possible solutions are:
1. *Make this information available by adding it to the previous interface*
2. *Make this information available by assigning a fixed value to it*
3. *Not to check the rule anymore*

Information that was expected to be saved into the database is not available. Please inform how to proceed in each case.
Resources / Infrastructure:
1. *Don't save this information into the database*
2. *Make this information available by assigning a fixed value to it*
3. *Make this information available by adding it to the previous interface*

...

Instead of trying to copy the action directly, the user could also have chosen to adapt it at first, which could have avoided these messages.

Continuing the extension, the user wants to skip the interface “Fill Out Service Planning”, since it’s not of interest to the customers. Now the user wants to turn the request (a Draft like object) into a Proposal. This would be a way to guarantee that the customer would receive an answer about the request made in three working days (this is the basic difference between a Draft and a Proposal). It would also be a way to connect the new workflow with Qualitas’ original workflow, so that the proposal becomes a contract that will be prepared, executed and evaluated.

The user goes back to the action just adapted and tries to add a step to it that turns the request (called *draft’*) into a proposal. But the system warns that in order to become a proposal, *draft’* needs to have a responsible technician. Thus the user decides to create a new interface where a technician would be allocated to *draft’*.

The basic method for creating a new interface is to compose it by choosing widgets and then associating each widget with system information. Figure 8 shows a possible example. On the left side, we see the interface under construction with its fields. On the right side, in a gray background, the user associates these fields with system information. The user also has to inform the roles that will be allowed to interact with this interface.

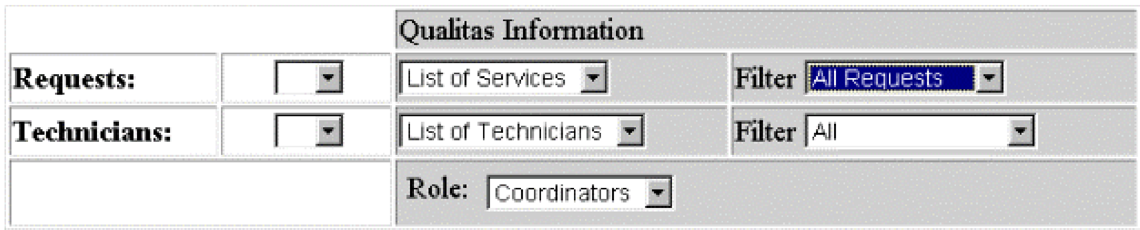

Figure 8. Creating a new interface

At this point our user has decided that only Coordinators would have the right to allocate technicians to customers' requests. It is important to point out that at this extension environment users can make extensions which parts are targeted at users with roles different from his/hers. In order to allow this, the extension environment has CSCW (Computer Supported Cooperative Work) mechanisms that guide the user during the extension, so that users manage to do it in a collaborative way. These mechanisms are outside the scope of this paper. It's a study on its own that will be described in further work.

After composing this interface, the user creates the new action that will process it. The user is provided with a set of steps and constructs the action by choosing some steps. He/She also informs how to process each information available at this point (usually originated at the interface). Figure 9 gives an idea of how this could be possible.

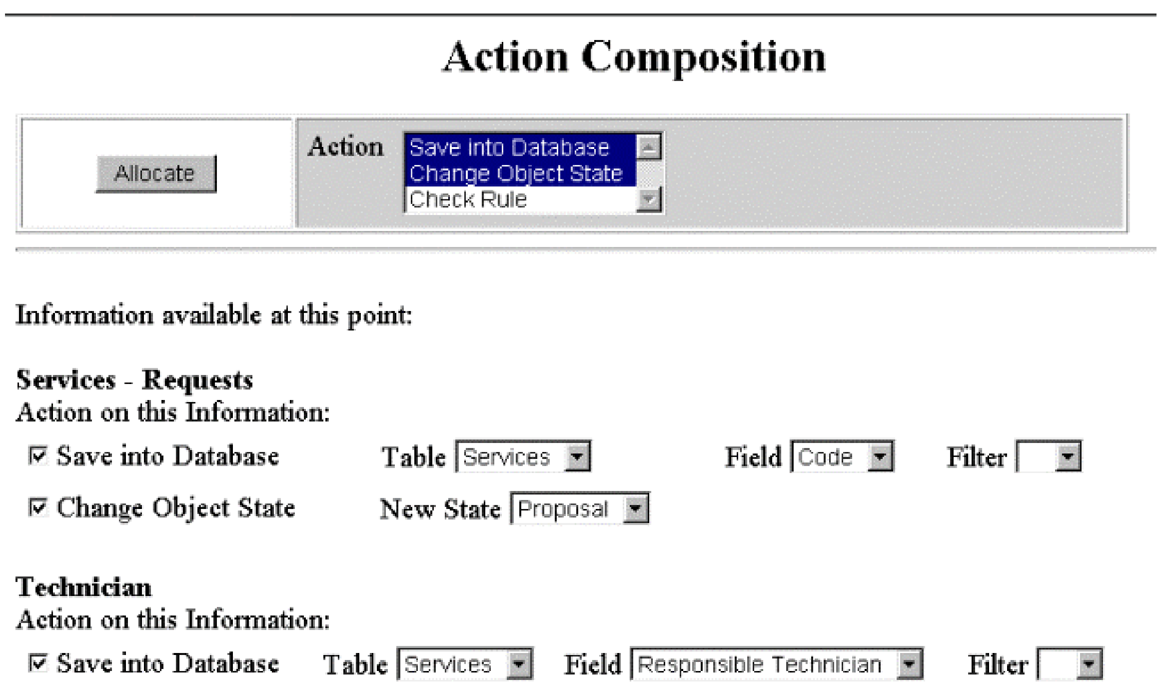

Figure 9. Composing an Action

The user-composed action basically saves the information into the database and transforms the request in a proposal.

After doing this, the user considers the extension to be finished and ready to be tested. Tests are handled in a test environment, separated from Qualitas itself, in order to protect the original system. When the user is done with testing, he/she can make the extension available at the application interface.

Executing Extensions at the Application Environment

During execution, problems not predicted by users who made the extension may happen (such as energy power fault). But the end user should still be able to return to the point where he / she was when the failure happened. Otherwise there would be lost and incomplete information inside the system.

To deal with this, a mechanism that recognizes unfinished, user created, workflows will be made available. The idea is that a log file is saved during execution, containing the current workflow, interface and, if necessary, action step. When loading, the system checks whether there is any log file for that user. If any log file is found, the system reads it and shows the interface from which the workflow is to be continued. In this situation, it also provides additional explanations along with the interface, in order to help the user recover the lost context.

The EUP Model

The model that supports the extension environment here proposed can be divided into three main models:

- The Workflow Model,
- The Interface Model and
- The Information Model.

The Workflow Model

A Workflow is an automation of business procedures "where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal" [WMC'1995].

The representation of Workflow used here is a top-down representation, such as in [WMC'1999], where, at the top level, there is a set of business processes (see Figure 1). At the next lower level, each business process is represented as a network of activities (see Figure 2). An activity represents a logical step within a process and has a defined responsible for performing it. A responsible is actually a role that can be assigned to the participants of the workflow (or users of the workflow system). By using this representation, the actual participants of an activity may change without affecting the workflow representation and processing.

In order to support the extension environment, one more level of detail is added to this representation. Each activity is represented as a network of interfaces. Each interface is a representation that points to the actual system web page (screen) that belongs to the Interface Model. The interface inherits the responsible role from the activity it represents (see Figure 3).

The Workflow Model can be viewed as in Figure 10.

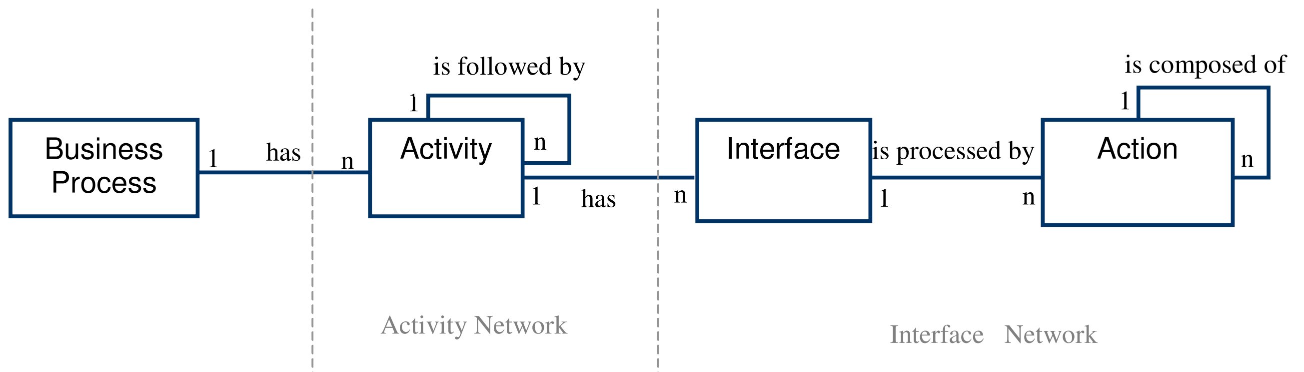

Figure 10. Workflow Model

The Interface Network is processed by an Augmented Transition Network (ATN) [Woods]. The states in this ATN represent each interface of the Interface Network. When moving from one state to the next, actions can be executed. These actions can be Transformational, Non-transformational or a combination of both. Registers work as “clipboards”, allowing information to be passed from one state to the next and making it available to actions executed during transitions.

The transformational actions are:

1. Create a New Object (such as a Draft Proposal)
2. Destroy an Object
3. Substitute an Object by another (which is actually a combination of the previous actions). This action allows, for example, the substitution of a Proposal by a Contract
4. Multiply. This action is useful if there are several Proposals for a single service and one of them is chosen and contracted. Then all the versions that were turned down should be destroyed. This action is actually an action 3, plus a sequence of actions 2.

The non-transformational actions are:

5. Store in database
6. Retrieve from Database
7. Update registers, where
   7.1. these registers can be available at the ATN top level
   7.2. these registers can be available at an ATN sub-level
8. Test. The different tests represented are:
   8.1. Security (access control)
   8.2. Input validation
   8.3. Date limits (that have to be followed by the users in order to attend the organization policy)
   8.4. Destiny State Pre-Conditions (each state can have pre-conditions that are required to be attended so that a transition to it can be completed. For example, the availability of information or input parameters)
   8.5. Synchronization
   8.6. Communication
   8.7. Monitor.

The three last test actions handle parallelism, which happens in Qualitas. But the full specification and the implementation of it are future work at this point.

The ATN is partially specified as the following top level rules:

task	→	(<arc set>$^{+}$)
<arc set>	→	(<state> <arc>*)
<arc>	→	(CAT <cat name> <test> <action>* <terminal act>) |
		(PUSH <state> <test> <action>* < terminal act>) |
		(TST <label> <test> <action>* <terminal act>) |
		(POP <form> <test>) |
		(PARALLEL <task>$^{+}$ <test> <action>* <terminal act>)
<action>	→	(SETR <register> <form>) |
		(SENDR <register> <form>) |
		(LIFTR <register> <form>) |
		(SETBD <table, field> <form>) |
		(GETBD <table, field> <form>) |
<terminal act>	→	(TO <state> <test_pre_cond_state>)
<form>	→	(GETR <register>) |
		➥ |
		(BUILDQ <fragment> <register>*) |
		(LIST <form>*) |
		(QUOTE <arbitrary structure>)

where [Woods]:

|	represents alternative ways of forming a construction
*	indicates zero or more repetitions

+ indicates one or more repetitions

➥ reads terminal symbol from input

( ) indicates list structure, non-terminal symbol

< > represents a non-terminal symbol. Some of them are not expanded, for their names are considered to be self-explanatory

CAT is an arc which can be followed if the current input symbol is a member of <category name> and the <test> is satisfied. At this arc, the actions are structure-building actions and the terminal action specifies the state to which control is passed as a result of the transition.

PUSH causes a pushdown to the state indicated. At this arc, the actions are structure-building actions and the terminal action specifies the state to which control is passed as a result of the transition.

TST permits an arbitrary test to determine whether the arc is to be followed. Also, at this arc, the actions are structure-building actions and the terminal action specifies the state to which control is passed as a result of the transition.

POP indicates under what conditions the state is to be considered a final state and the <form> to be returned as the value of the computation if the POP alternative is chosen

PARALLEL indicates a situation that one arc can lead to more than one task. This arc is to be detailed in future work

action builds structures and specifies the state to which control is passed as a result of the transition. It represents a function call, where the first element inside the parenthesis represents the function name and the next elements are its parameters

SETR sets the value of the register at the current level equal to the value of <form>

register information representation that is built as the ATN proceeds from state to state

SENDR sets the value of the register at the next lower level equal to the value of <form>

LIFTR sets the value of the register at the next higher level equal to the value of <form>

SETBD saves the value of <form> in the specified <table, field> of a database

GETBD reads the value in the specified <table, field> and make it available as <form>

TO indicates that the input pointer is to be advanced, that the next state is to scan the next input data. This will happen if the test in <test_ pre_cond_state> is satisfied

form as the <action> symbol, it represents a function call, where the first element represents the name of a function and the remaining ones, its parameters

test conditions, also represented as function calls

GETR is a function whose value is the contents of the indicated register

BUILDQ is a structure building form. It takes a structure with specially marked fields and returns the same structured with the marked fields filled out with the contents of indicated registers

LIST makes a list of the values of the listed parameters

QUOTE produces as value the argument form

Now, let's see how the Interface Network of Figure 3 would look like when processed by the ATN:

```
(Get Customer Dept.
        (CAT <Customer Dept.> <Validate Input>
                <SETR <Customer>
                        <BUILDQ <Customer> <Customer Dept.>> >)
                (TO <Get Customer Contact Data> <Test Output=Input>) )
(Get Customer Contact Data
        (CAT <Customer Contact Data> <Validate Input>
                <SETR <Customer>
                        <BUILDQ <Customer> <Customer Contact Data,
                                                    GETR<Customer.Dept.>> > >)
                (TO < Get Service Type > <Test Output=Input>) )
 (Get Service Type
        (CAT <Service Type> <Validate Input>
                <SETR <Service>
                        <BUILDQ <Service> <Service Type>> >)
                (TO [Get Service Data] [Test Output=Input]))
(Get Service Data
        (CAT <Service Data> <Validate Input>
                <SETR <Service> <BUILDQ <Service> <Data, GETR<Service.Type>> >
                  SETBD <Service, Customer Dept.><GETR <Customer.Customer Dept.> >
                  SETBD <Service, Contact Data> <GETR <Customer.Contact Data> >
                  SETBD <Service, Service Type> <GETR <Service.Type> >
                  SETBD <Service, Service Data> <GETR <Service.Data> > >
                (TO <Make Proposal> <Test Output=Input>))
```

We will now explain the Interface Model, the Information Model and then we will show how these three models are related to each other.

The Interface Model

The main purpose of the interface model is to allow users to adapt existent interfaces or to create new ones during extension. An interface is defined as a set of fields and a field can be of one of three types: input, output or action. Input fields are data entry and navigational (link) fields. Output fields are basically text that may represent:

- a field label, such as the text "Requests:" on the left side of Figure 8.
- a field value that may not be changed, such as context information originated by previous interactions. An example is the text $|current_customer(name)|$ in Figure 6.
- an automatically generated natural language (NL) text, such as error messages. Our model also previews the generation of workflow summaries and explanations. These summaries can be generated based on the ATN, generally by translating the ATN into NL text. The generation process is template-based. Advances in the NL generation process are previewed as future work, which includes the automatic generation of links embedded in the texts.

An action field is a field that triggers the processing of an interface, such as button. Each field has a corresponding set of widgets that can be selected by the user during extension. Figure 9 depicts the Interface Model.

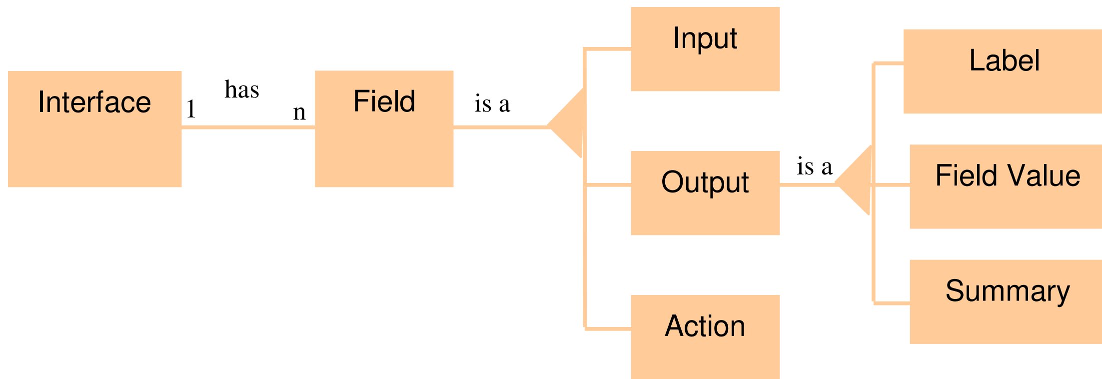

Figure 9. Interface Model

Each field is associated with system information that is represented in the Information Model.

The Information Model

The Information Model represents information about the application domain. Each information unit has many perspectives that are used depending on system processing need in a moment. For example, when generating a NL text, information like gender is needed. When saving information into a database, information like the table and column names are needed. Figure 10 shows partially how the Customer information unit would be represented.

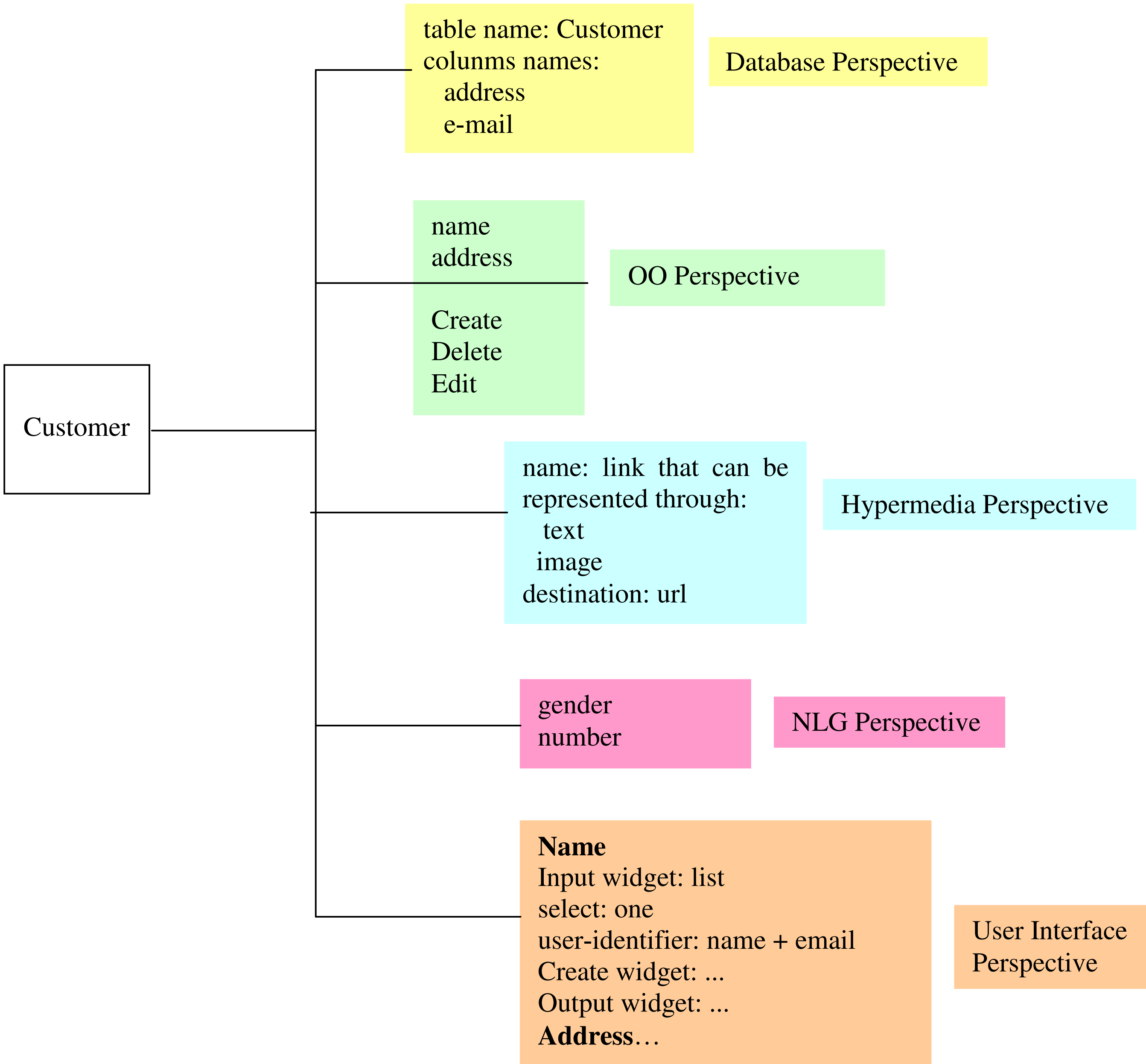

Figure 9. Customer Information Unit

Figure 10 shows a representation of the Information Model. Note that information units are related to each other.

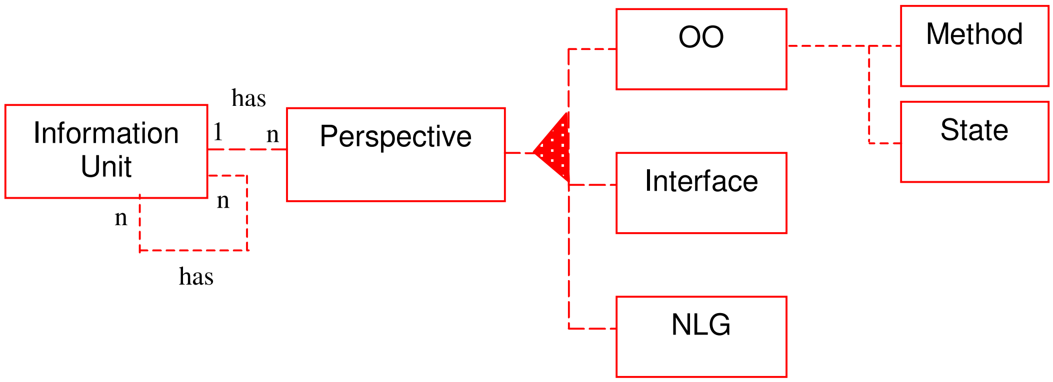

Figure 10. Information Model

Relating the Three Models

The information model serves both the Workflow model and the Interface model, as depicted by the lines of Figure 11.

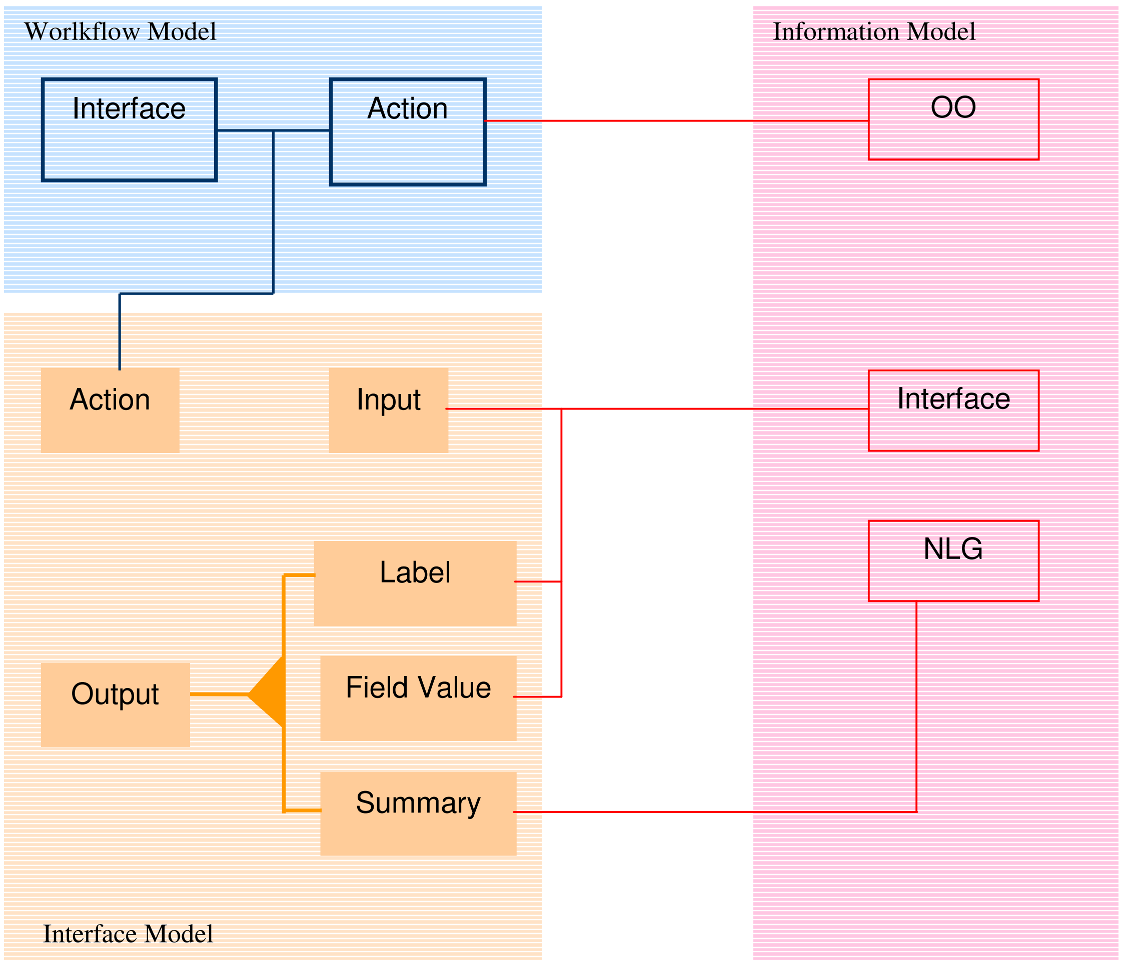

Figure 11. Relation among models

The methods of the OO perspective of a information unit serve to compose actions of the Workflow Model, which in turn are the actions available to user when creating or adapting an interface. The interface perspective helps to guide the user when mapping widgets (input and some output) to information units. The NLG perspective provides information necessary to text generation.

Conclusion

The proposal of an EUP model for Workflow Web Based Systems has been presented. In future work [Cunha et al.] we will be describing how this model relates to other EUP proposals and how semiotic studies guided its development. After this, the next main steps would be to continue the development and tests of this model and then to implement it in real applications.

At this point, as in the exemplified scenario, it was possible to see that this model allows, not only the automation of repetitive single user tasks, but also that the application maintenance task be shared among software engineers and empowered end users.

It will also be interesting to study how our proposal connects the areas of EUP and Groupware, as is the case of extensions that are targeted at users with roles different from that of the end user programmer. Still interesting will be to see how EUP can be possible in an application that has NL and Hypertext automatic generation, so that these parts also work for the extension developed by the end user.

References

[Cunha et al.] Cunha, C. K. V., Quental, V. S. T., Souza, C. S.; End User Programming model for Workflow Web Based Systems; *forthcoming*

[Myers] Myers, B. A.; 1992; Languages for Developing User Interfaces; Jones and Bartlett Publishers, Inc. Boston.

[WMC] Workflow Management Coalition – Terminology and Glossary; Document Number WFMC-TC-1011, Document Status – Issue 3.0; 1999.

[WMC'1995] Workflow Management Coalition – The Workflow Reference Model; Document Number WFMC-TC00-1003, Document Status – Issue 1.1; Hollingsworth, D.; 1995.

[Woods] Woods, W. A.; 1970; Transition Network Grammars for Natural Language Analysis; Communications of the ACM; Vol. 13; No. 10; pages 591-606.