IADIS International Journal on WWW/Internet
Vol. 1, No. 1, pp. 1-14
ISSN: 1645 - 7641

APPROACH FOR COMPONENT-BASED SOFTWARE SPECIFICATION

Using DEMO-UML Based Generic Components to Design a Tele-Work Brokerage System

Boris Shishkov *Delft University of Technology, Faculty ITS, Department ISA. Mekelweg 4, 2628 CD, Delft, The Netherlands*

Shishkov@IS.TWI.TUDelft.nl; http://www.shishkov.tk

Jan L.G. Dietz *Delft University of Technology, Faculty ITS, Department ISA. Mekelweg 4, 2628 CD, Delft, The Netherlands*

j.l.g.dietz@its.tudelft.nl

ABSTRACT

Realization of effective match-making between teleworkers and companies offering Tele-Work positions is considered crucial for the successful development of Tele-Work. This should be realized by ICT (software) applications which operate across distributed computing environments. A frequent cause of software project failure in this regard is the mismatch between the (business) requirements and the actual functionality of the delivered application. We contribute to the solution of this problem by suggesting in this paper an approach for the design of software, basing consistently this design on prior business process modeling. The alignment between these two tasks is realized in a component-based way (by reflecting identified business components in the design of software). This is a definite advantage because of the possibility to identify generic business components and reuse them in the development of different applications. The suggested approach is expected to be useful for Tele-Work (and in particular, for the development of effective Tele-Work brokerage systems) as well as for the design of software in general.

KEYWORDS

Modeling; Tele-Work (TW); Brokerage System; DEMO; UML; Generic Component

1. INTRODUCTION

Tele-Work (TW) is a kind of work, in which all parties (worker-supervisor, customer-supplier, etc.) use the possibilities of the new Information and Communication Technology

(ICT), including modern software environments, to exchange data (e.g. work results) over some distance, and thus – being relatively independent on time and space respects (Barjis and Shishkov, 2001).

ICT creates new opportunities which could dramatically change our ways of living and thus also working. The possibility to establish a full-value distant communication and exchange all kinds of information determines the new reality in which one could work for his company from home, collaborating with his colleagues and granting the work results to his boss virtually. The collaboration technologies which are based entirely on ICT even create some additional "centripetal forces", as defined in (Carmel, 1999), for successful distant collaboration - e.g. global teams.

There is no doubt that modern society should benefit to a maximum degree from TW, because this could not only improve people's standard of living but also save resources, travel time, office equipment expenses, and increase mobility and flexibility of individuals, and business itself. From this point of view, TW would be crucial for the social prosperity of tomorrow. TW application area already covers vitally important spheres such as remote education, remote services (e.g. banking) etc. (Bonyuet and Bagayas, 2000). For more information on TW, interested readers are referred to (Andriessen and Roe, 1994; European Commission, 2000).

Among the actual problems to be solved for the successful development of TW are 1) the insufficient level of facilitating people with advanced ICT infrastructure (essential for conducting TW activities); 2) the insufficient degree of knowledge about the technology used in realizing TW activities; 3) the lack of effective mechanisms for matching between individuals looking for TW positions and companies offering such positions. This particular problem has been addressed in the current paper.

We suggest an approach for developing TW Brokerage Systems (TWBS). TWBS are supposed to match up effectively teleworkers and companies using such workers.

In designing the approach, we have addressed one of the actual problems of modern software development, namely the mismatch (in many cases) between the (business) requirements and the actual functionality of the delivered software application. The suggested approach allows for designing software based on prior business process modeling.

Another essential issue is that we adopt the promising ideas of component-based system development – allowing for reuse of developed components (reusing and replacing components in the design of software is claimed to increase flexibility, ease maintainability, reduce the development time and costs). In our approach, we suggest basing the design of software (supposed to support (business) processes) on components identified from them. Since, according to the approach, these components are to be generic for the considered (business) domain, they would be reusable in designing a broad range of software applications within the particular domain. This would require the generic components to be adjustable in one way or another, depending on the particular use. For example: a generic component with brokerage functionality (such a component will be considered further on in this paper) could be adjusted and extended in one way or another for building a TWBS, e-trade brokerage system, hotel reservation system, etc.

According to the suggested approach, a considered (business) system is presented in terms of a set of generic (business) components. Further on, the business process models of these components are reflected in the design of software. This leads to software models that stem out from corresponding business process models, and is a guaranty that the business requirements are properly reflected in the designed software. This is an essential advantageous

feature of the suggested approach. Another benefit is the possibility to reuse developed components in building different applications.

The outline of the paper is as follows. In section 2, the suggested approach is outlined. Section 3 briefly introduces and discusses DEMO and its relation to UML (these modeling tools and their combined application are considered crucial for the suggested approach). Section 4 illustrates an essential part of the approach, namely the identification of a generic business component and its reflection in a software model. Section 5 contains the conclusions.

2. INTEGRATED APPROACH FOR SYSTEM MODELING BASED ON GENERIC COMPONENTS

Aiming at consistently aligning the design of software systems and prior business process modeling, we suggest in this section an integrated approach that allows software design straightforwardly stemming from a business process model.

Before outlining the approach, we will discuss briefly some fundamental considerations regarding it.

An issue that we claim to be essential in designing software applications which effectively support processes in contemporary (business) domains (e.g. TW, e-Business, Banking, etc.), is the consistent alignment between software design and prior (business) process modeling (as stated in the introduction, one frequent cause of software project failure is the mismatch between the business requirements and the actual functionality of the delivered application). For this reason, the software design activities (within the suggested approach) stem from a model of the (business) processes to be supported by the software under development.

Another fundamental consideration is the adoption of component-based software development (Jacobson et al, 1992), as a promising contemporary way of software development, founded on the principles of object-orientation. As it is well known, object-orientation (characterized by the fundamental concepts of encapsulation, classification, inheritance and polymorphism) is widely considered as a special approach to the construction of models of complex systems, in which a system consists of a large number of objects. This applies not only to software systems but also to business systems (Jacobson et al, 1992). Thus, it seems feasible to expect that software design and business process modeling could be bridged by basing the design on software components which are derived from some (business) components. The components should fill the gap between the two mentioned tasks. If generic components are considered, they could be reused for designing different applications. Next to that, component-based development seems beneficial for the application design itself. By basing application development on encapsulated, individually definable, reusable, replaceable, interoperable and testable components, developers could build applications which possess durable configuration and a high degree of flexibility and maintainability. The process of application development would also be improved because building new applications would include using already developed components. This reduces development time and improves reliability. The performance and maintenance of developed applications would be enhanced because changes could occur in the implementation of any component without affecting the entire application. All this makes the component-based application development much more effective than the traditional way of application development.

A third fundamental consideration is that in contemporary software development (where software systems are built by distributed teams and are supposed to include reusable, replaceable, upgradeable software components, to be operated my a multitude of developers spread across different locations) it is crucial that the software artefacts are being developed using common, unified, standard development tools and environments that are known by as broad circles as possible. For this reason, we suggest basing our approach on the Unified Modeling Language – UML (OMG, 2000); UML is becoming *de facto* the standard language for modeling software systems (Shishkov and Dietz, 2002), widely recognized by both researchers and practitioners. However, because of the limited scope of this paper, we will not discuss UML in more detail. It is considered well-known to the public.

Based on these three fundamental considerations, we can narrow the demands in developing the approach: it should allow for representing a target (business) system in terms of reusable, replaceable, interoperable and extensible generic components, as well as for further reflection of these components in UML software models.

An essential issue in this regard is the necessity to properly model the generic (business) components – to be adopted in the design of software. It is necessary to apply a consistent business process modeling tool that is complete, straightforwardly relatable to UML as well as capable of capturing the essence of the (business) processes to be supported by the software under development (this last feature is considered crucial since only such a full abstraction could offer the right (re)design freedom for the software system designer (Shishkov, 2002)). As such modeling tool we consider DEMO (Dietz, 1999). For this reason, we base out suggested integrated approach on DEMO as a business process modeling tool. The following section considers DEMO as well as its relatability to UML.

Based on all the considerations, stated above, we introduce the suggested approach (Figure 1).

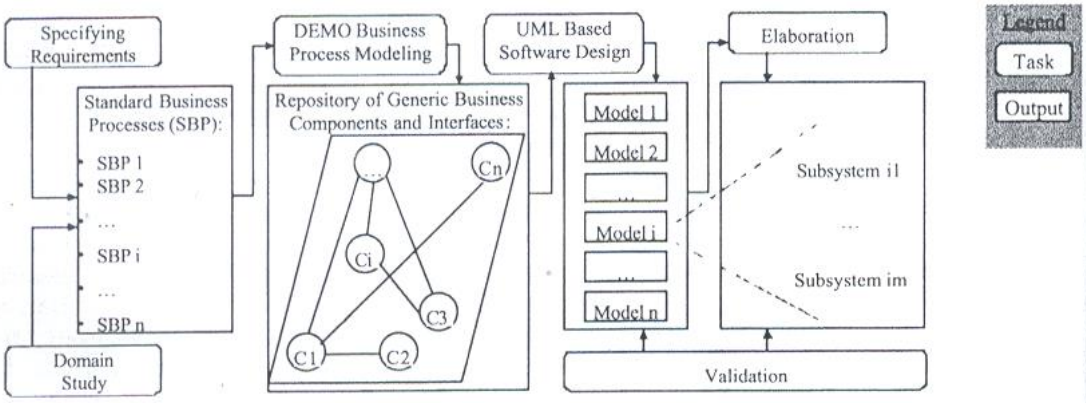

Figure 1. Approach for component-based system development

As seen from the figure, the starting point (according to the suggested approach) should be to study the considered (business) domain(s) and analyze different user requirements related to it. On this basis a set of business processes, standard for the considered domain should be defined (SBP 1 ... SBP n). It should be then possible to represent any particular user requirement(s) in terms of combination(s) of SBP. Thus, by building software capable of consistently supporting SBP, we would respond to the user requirements.

Further on, DEMO should be applied to investigate SBP, aiming at grasping the relationship between the system to be modeled and these business processes. The output of the DEMO investigation should be a repository of generic business components and interfaces.

Based on this, each component should be reflected in the design of software as a self-contained system model. Hence, the overall functionality of the system under development should be modeled, based on the realized business process investigation. Each modeled system needs to be validated.

The next step should be: partial representation of (some) system(s) (with particular importance) as a subsystem, and further granularity of the modeled subsystem(s) with respect to structure and realized activities. Looking inside some of the subsystems is considered useful since it would allow for an extended insight regarding their structure and dynamics. These modeling activities should also be validated.

Actually, validation is considered essential for different parts of the suggested approach since it is useful to know if the source (business) process model(s) is consistently reflected in a (software) system model as well as to validate the constructed subsystems making sure that their structural and dynamic models are relevant to the needed system functionality.

The intermediary results in designing the essential parts of the suggested approach have been reported (Shishkov, 2002; Shishkov and Dietz, 2002) and demonstrated through case studies.

In fact, the fundamental goal behind the suggested approach is related in one way or another to the goals behind Tropos (Mylopolous et al, 2001) and other consistent approaches addressing software system development. However, among the distinctive beneficial features of our approach are the following:

- The software design stems from a complete and consis tent business process study.
- The software design and prior business process modeling are aligned in a component-based way, taking benefits from the advantages of object-orientation.
- The software design is based on the standard language for modeling software systems.
- The software design is based on generic business components that could be reused.

In the following section, we elaborate on DEMO and its relation to UML since these issues are of fundamental importance for the suggested approach.

3. REALIZING BUSINESS PROCESS MODELING WITH DEMO AND ITS RELATION TO UML

Dynamic Essential Modeling of Organizations - DEMO is a methodology for understanding, analyzing, (re)designing and (re)engineering business processes. Its underlying theory about organizations is rooted in the Language/Action Perspective (Flores and Ludlow, 1980), Organizational Semiotics (Liu, 2000) and Philosophical Ontology (Bunge, 1979). DEMO reveals the "construction" and "operation" of an organization, contrary to the current function and behavior-oriented approaches. It is characterized by three major features: 1) a white-box architecture of actors, production and coordination, 2) the extraction of the essence of business processes from their realization, 3) the transaction pattern.

Actors, production, coordination

Like every other system (e.g. an alarm clock or a racing car), the functional behavior of an organization is brought about by the collective working of the constructional components. The construction and the working of a system are most near to what a system really is, to its ontological description (Bunge, 1979). An organization is defined as a (discrete dynamic) system in the category of social systems. This means that the elements are social individuals or actors, each having a particular authority to perform production acts (P-acts) and a corresponding responsibility to do that in an appropriate and accountable way. The structure of an organization consists of coordination acts (C-acts), i.e. the actors enter into and comply with commitments regarding the performance of P-acts. The generic white-box organizational model (Figure 2) consists of: the actors, the P-world, and the C-world (Dietz, 1999).

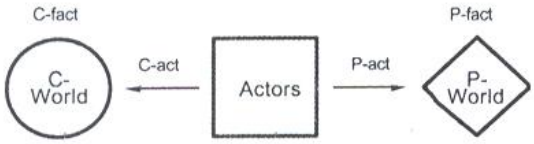

Figure 2. The white-box model of an organization

By performing Pacts, the organization does what it is supposed to do according to its function. C-acts serve to coordinate and control the performance of P-acts.

Essence and realization

In DEMO, three perspectives on an organization are distinguished, called essential, informational and documental (Dietz, 1994), as exhibited in Figure 3:

Figure 3. The three perspectives on organizations

- *essential* - the organization viewed as a system of authorized and responsible actors that create new original facts;
- *informational* - the organization viewed as a system of information processors that remember facts and derive new facts from existing ones;
- *documental* - the organization viewed as a system of formal operators that collect, transport, store, copy and destroy representations of facts.

Take for example the process of withdrawing money from a bank account using an ATM machine. Think of observing this process through essential, informational or documental "glasses" as a metaphor. Looking through documental "glasses" we see someone inserting a

card into a machine, pushing buttons on a keyboard and finally getting out the card and other pieces of paper. Nothing with respect to the information on it or the purpose for which they are used, is seen. Looking at the same process through informational "glasses", we see someone providing information to an ATM system: a PIN code and specification of an amount of money. Also, the machine provides information if withdrawal is possible to the customer. We see that the machine outputs money and receipts. Looking through essential "glasses" shows responsible actors, their actions and interactions. A customer requests a bank to withdraw money from an account. The bank decides to do this and states that the money is withdrawn. Further on, the customer accepts it.

The transaction pattern

Production acts and coordination acts appear to be performed in particular sequences that can be viewed as paths through a generic pattern called the (business) transaction (Dietz, 1999). It is exhibited in Figure 4:

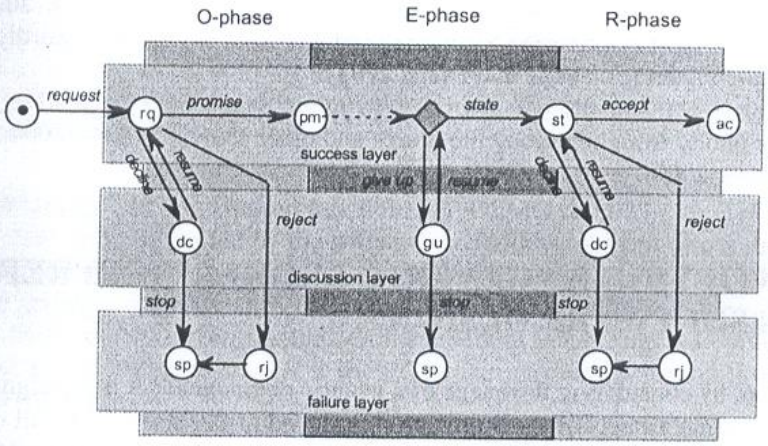

Figure 4. The Transaction Pattern

A transaction is a finite sequence of C-acts between two actor roles, the customer and the producer. It takes place in three phases: the order phase (O-phase), the execution phase (E-phase), and the result phase (R-phase). O-phase is a conversation that starts with a request by the customer and that, if successful, ends with a promise by the producer. E-phase basically consists of the performance of the P-act by the producer. R-phase starts with the statement by the producer that the requested act is performed and ends, if successful, with the accept by the customer. The whole pattern of a transaction is represented by one symbol in the so-called Coordination Structure Diagram (CSD). Figure 5 exhibits CSD for the money withdrawal example. The two boxes represent the two actor roles involved: A0(A1) is the customer(producer). The small black box indicates that A1 is the producer of T1 (and consequently A0 is the customer). The successful result of a transaction T1 is the P-fact "withdrawal W is performed" where W is constituted by the account, the amount and the time.

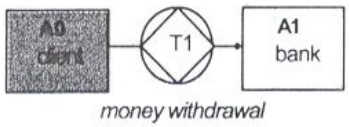

Figure 5. CSD of the money withdrawal example

Relating DEMO and UML

The above paragraphs elaborated on the features of DEMO that are beneficial for using it in the suggested approach. Since, as already stated, the approach is based on UML, another essential issue in this regard is the possibility to reflect a DEMO business process model into a UML software one. This means to consistently derive use cases based on a DEMO business process model (because use cases play a crucial role for linking the application domain (the business world) to the software domain in the UML based software design). This issue has been thoroughly investigated (Shishkov and Dietz, 2002; Shishkov et al, 2002) and it was studied that a DEMO Coordination Structure Model (CSM) is straightforwardly mappable into a UML use case model (Shishkov and Dietz, 2003).

However, we will not discuss use cases in more detail since they are considered well-known to the public from numerous literatures and other sources (OMG, 2000; Jacobson et al, 1992; Cockburn, 2000).

4. IDENTIFYING A "GENERIC BROKER" AND REFLECTING IT IN SOFTWARE DESIGN

In this section, by considering the usage of a generic component for the design of a TWBS, we will illustrate some aspects of the approach introduced in Section 2. We will demonstrate the identification of a generic component and its representation via DEMO CSM, and based on this – derivation of a use case model (extending the component) as well as further elaboration (on structural and dynamic issues) concerning a particular use case from the constructed model, using the use case theory of Cockburn (Cockburn, 2000), (Shishkov and Dietz, 2001) and UML Activity diagram, respectively, as studied in (Shishkov and Dietz, 2002).

The considered component - "Generic Broker" (GB), is supposed to be generic for a particular domain. It is easily seen that similar brokerage functionality is required by a TWBS, e-trade system, a hotel reservation system, etc. Hence, it seems feasible to expect that identifying a GB in the entire domain of e-business would allow us easily use this component for building different brokerage systems, e.g. TWBS.

What should be the functionality of a GB? It should match the data of those looking for TW positions/goods/accommodation/etc. (we will call them "Buyers") and those offering TW positions/goods/accommodation/etc. (we will call them "Sellers").

The general view of the required functionality of a GB is depicted on Figure 6:

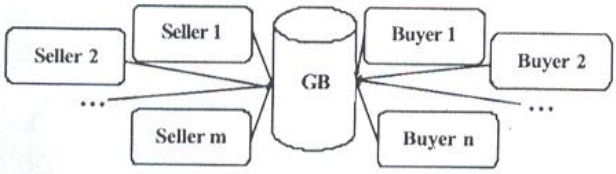

Figure 6. General view of the functionality of a GB

There are different Sellers (**S**) aiming at succeeding to sell their goods as quickly as possible; different Buyers (**B**) aiming at purchasing specific goods they are interested in, as seen on the Figure. GB is supposed : 1) to let seller *i* find the buyer being interested in the goods offered by him; 2) to let buyer *j* find the seller offering the goods he is interested in. **S** and **B** could, for example, pay on a subscriptional basis for the realized service.

Anyway, behind this not so complex general functionality, there are many issues which should be taken into account when developing such a distributed application: how to store, operate and maintain the data; how the application should provide its services to users, how some non-standard situations should be approached, etc. These issues are to be addressed in the modeling process that follows below.

First

Based on the description of the required functionality, DEMO should be applied to explore the business processes to be supported by the software under development. From the description, two essential business transactions (transaction types) are identified. They are listed in Table 1 for illustrative purpose, together with their corresponding resulting fact types. It should be noted that the focus is only on transactions on the essential level. That is in order to keep the business model abstract enough so that it should remain unchanged during (eventual future) re-design of its realization.

Table 1. Business Transactions List

transaction type	result fact type
T1 match-making	F1 *match <M> is made*
T2 payment	F2 *the fee for period <P> by <S/B> is paid*

On the basis of the transactions and result facts, the system(s) to be investigated should be selected, relevant DEMO actor(s) should be identified, and their roles – determined (as customer and producer). Once this is done, all interaction relationships are determined. All this is depicted in Figure 7, representing the Coordination Structure Model or CSM (the model is not complete, because the purpose is only to illustrate the application of DEMO within the suggested approach).

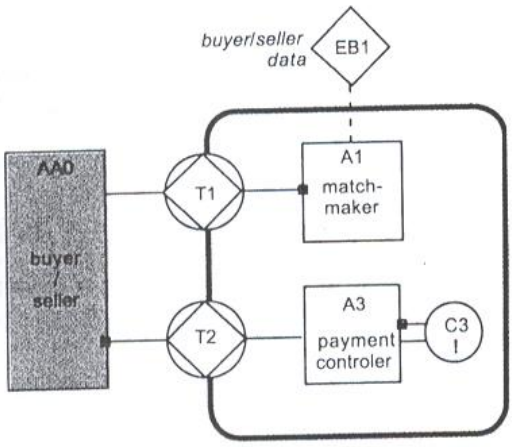

Figure 7. Coordination Structure Diagram of GB

The system under study (GB) is considered as well as the Seller and Buyer (as actors). Regarding the system under study, it is represented on the figure in more detail: actors A1 and A3 (white boxes) whereas the Seller and the Buyer are taken together in the aggregate actor AA0 (grey box) since they basically play the same role towards the actors A1 and A3. The transaction types are represented by a symbol combining a disk and a diamond symbol. The small disk C3 represents a so-called conversation for initiation. It models the periodic activation of A3 to issue payment requests. The system boundary is represented by a grey round angle. There is a so-called external bank (EB1) which contains the company data provided by the sellers and buyers. The dotted line between EB1 and A1 means that actor A1 is allowed to inspect the contents of EB1. In other words, actor A1 is allowed to know the information provided by the sellers and buyers. The reason for this allowance is of course that A1 needs to know the provided information. How A1 gets access and also how sellers and buyers add and remove data is not shown. These matters are considered to belong to the informational and documental perspective and therefore are not represented in the (essential) CSM.

Second

The second step is development of a use case (UC) diagram, based on the DEMO CSM. The diagram (Fig. 8) shows UC and actors in the context of the considered system – TWBS (GB is being extended for this particular purpose). Since the goal is just illustrative, only some of the UC and actors typical for such a system are considered.

It should be noted that, in addition to the essential business transactions (focused by DEMO), the UC diagram considers also the actions which represent information providing (but are not essential business transactions), e.g. adding data to the database used by TWBS. These actions are additionally identified in building the UC diagram and are of great importance for the particular design of an ICT application.

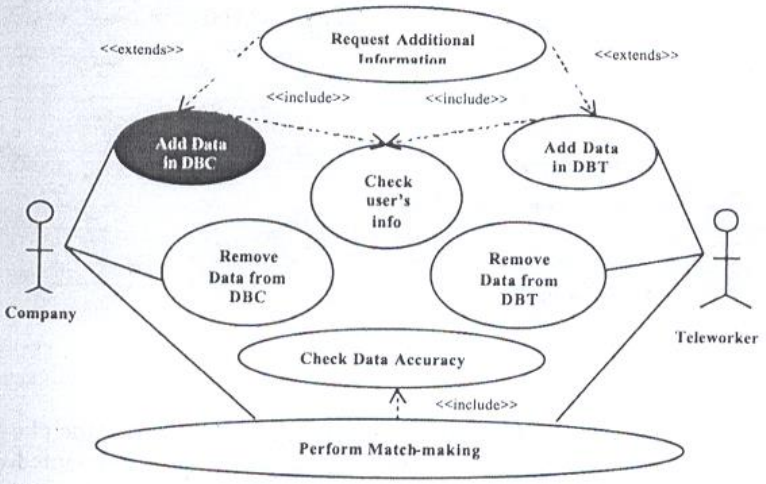

Figure 8. Use case diagram of TWBS

Regarding the diagram, the abbreviation DB stands for the database, used by TWBS. For convenience, DB is virtually divided into DBC/DBT (containing data of offered/searched TW positions respectively). There are two actors: Company and Teleworker. Concerning Company (Teleworker) – he takes the decision, has the responsibility, has the goal to add data in DBC (DBT), and/or remove data from DBC (DBT), and have his information matched up with relevant data from DBT (DBC). The diagram contains eight UC: "Add Data in DBC", "Request Additional Information", etc. The UC "Add Data in DBC" is highlighted since it will undergo the further steps of the approach. There are three <<include>> relationships ("Perform Match-making" requires "Check Data Accuracy"; "Add Data in DBC" and "Add Data in DBT" require "Check user's inf.") and two <<extends>> relationships (in some cases, before adding their data to DBC/DBT, the system might request from Company/Teleworker additional data, so the basic UC are "Add Data in DBC" and "Add Data in DBT", and they are extended with "Request Additional Inf.").

Third

The third step is further investigation of any particular UC of interest, based on the concept of Cockburn (Cockburn, 2000). We have selected, for illustrative purpose, the UC "Add Data in DBC", and the mentioned investigation is applied to it – Figure 9 (only those extensions related to activity six are depicted).

The UC is written at 'system' scope (as opposed to 'enterprise' scope) since it describes an interaction with a computer system. The indicated 'summary' level means that the UC is long running (executed over months or years), showing the context in which the user goals operate.

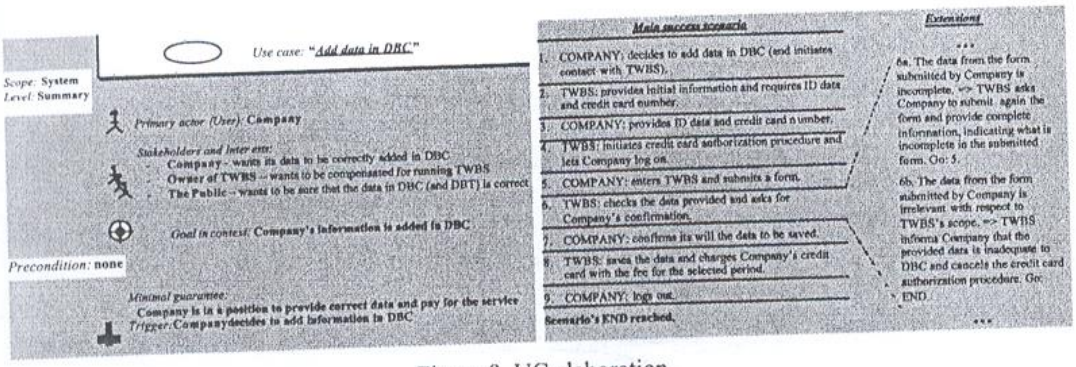

Figure 9. UC elaboration

Fourth

The fourth step is construction of an activity diagram (AD) model for the chosen UC. As seen from the main success scenario, there are nine core activities (complemented with extensions) in the UC "Add Data in DBC". Some of them are shown on Figure 10, as an overall AD. And finally, from the AD model it is straightforward to proceed with computer simulation, in order to validate the model. This was studied in (Barjis and Shishkov, 2000).

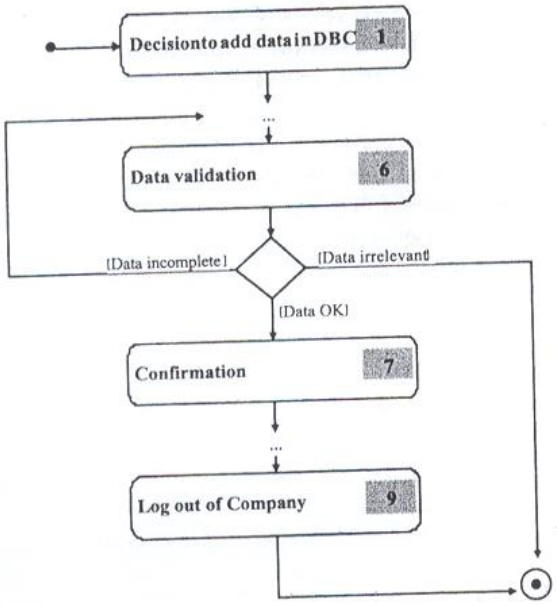

Figure 10. AD model for the UC: "Add data in DBC"

5. CONCLUSION

The purpose of this paper, as it was stated in the introduction is to suggest an approach for developing TWBS (an approach, usable for developing ICT applications also in other domains) contributing in this way to the TW domain on one hand and to contemporary software development, on the other hand. The suggested approach aligns in an original way software design and business process modeling, using the principles of component-based system development. It is suggested to identify generic business components and further reflect them in the design of software, taking advantage of the reuse of identified components in the development of different applications.

It was shown how a UML software model could be consistently derived from a business process model. It was studied that DEMO is a proper tool for that purpose. It was studied also how a business system could be represented in terms of reusable generic components. The reflection of such a generic component in the design of software was illustrated.

The interaction among components and its reflection in software design is planned for further research.

REFERENCES

Andriessen, J.H.E. and R.A. Roe, 1994. *Telematics and Work*. Hove, Sussex: Lawrence Erlbaum.

Barjis, J. and B. Shishkov, 2000, UML based Business Systems Modeling and Simulation. *Proceedings of 4th International EUROSIM Congress*. Delft, The Netherlands.

Barjis, J. and B. Shishkov, 2001, Telematic Applications for Supporting Telework Related Activities. *Proceedings of 6th International Conference on Computer Supported Cooperative Work in Design*. London, Ontario, Canada.

Bonyuet, D. and L.H. Bagayas, 2000, The Internet in the New Millenium. *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*. Orlando, Florida, USA.

Bunge, M.A., 1979. Treatise on Basic Philosophy. *In D. Reidel Publishing Company*, Vol. 4, Dordrecht, NL.

Carmel, E., 1999. *Global Software Teams*. Prentice-Hall, Inc., USA.

Cockburn, A., 2000. *Writing Effective Use Cases*. Addison-Wesley, USA.

Dietz, J.L.G., 1994, Business Modelling for Business Redesign. *Proceedings of 27th IEEE Hawaii International Conference on System Sciences*. Los Alamitos, USA.

Dietz, J.L.G., 1999, Understanding and Modelling Business Processes with DEMO. *Proceedings of ER*. Paris, France.

European Commission, 2000. *eWork 2000 – Status Report on New Ways to Work in the Information Society.*

Flores, F. and J.J. Ludlow, 1980. Doing and Speaking at the Office. *In G. Fick and H. Sprague, Decision Support Systems: Issues & Challenges*, Perg. Press, New York, USA.

Jacobson, I., M. Christenson, P. Jonsson, G. Overgaard, 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, USA.

Liu, K., 2000. *Semiotics in Information Systems Engineering*. Cambridge University Press, London, UK.

Mylopoulos, J., M. Kolp and J. Castro, 2001, UML for Agent-Oriented Software Development: the Tropos Proposal. *Proceedings of 4th International Conference on the Unified Modeling Language*. Toronto, Ontario, Canada.

OMG, 2000. *UML, version 1.3*. Object Management Group – www.omg.org.

Shishkov, B., 2002, Business Engineering Building Blocks. *Proceedings of 9th Doctoral Consortium on Advanced Information Systems Engineering*. Totonto, Ontario, Canada, pp. 85-96.

Shishkov, B. and J.L.G. Dietz, 2001, Analysis of Suitability, Appropriateness And Adequacy of Use Cases Combined With Activity Diagram For Business Systems Modeling. *Proceedings of 3rd International Conference on Enterprise Information Systems*. Setubal, Portugal, pp. 854-858.

Shishkov, B. and J.L.G. Dietz, 2002, Integrated Methodology Allowing Design of ICT Applications Based on Business Process Investigation. *Proceedings of IASTED International Conference on Applied Simulation And Modeling*. Crete, Greece, pp. 1-6.

Shishkov, B. and J.L.G. Dietz, 2003, Deriving Use Cases From Business Processes, The Advantages of DEMO. *Proceedings of 5th International Conference on Enterprise Information Systems*. Angers, France.

Shishkov, B., Z. Xie, K. Liu, J.L.G. Dietz, 2002, Using Norm Analysis to Derive Use Cases From Business Processes. *Proceedings of 5th Workshop On Organizational Semiotics*. Delft, The Netherlands, pp. 187-195.