

Role-Based Transaction Management In Collaborative Systems

K. Harrison-Broninski

Role Modellers Ltd
Brazenhall, Horn Street, Nunney
Somerset BA11 4NP, UK
khb@rolemodellers.com

F. Hayden

Francis Hayden Ltd
Well House, Nunney
Somerset BA11 4LZ, UK
francis@francishayden.com

Abstract - *First generation process support for Collaborative Systems, Workflow, does not support open-ended human activities like problem solving and design. The second generation, Business Process Management (BPM), does permit flexibility. But this introduces a new problem: most public processes are not fixed structures of simple atomic messages, but flexible sets of complex extended transactions in which the number of possible process loops rapidly tends towards infinity. We illustrate this by reference to concurrent engineering, where interactions generally involve several parties, contain processing spread over multiple systems on different platforms owned by different organizations, and require a number of iterations to complete successfully. We describe the problem in detail, present a general solution as a pattern of transactions and dependencies, show the inherent requirement for a notion of transaction that is different from that found in database systems, and provide a process representation for the model which can be implemented directly.*

Keywords: Business Process Management, Collaborative Systems, Groupware, Role Activity Theory, Role Activity Diagrams, RADs, Playwright, RADRunner, Design Engineering.

1 Introduction

Collaborative Systems, especially when used by virtual teams in complex projects, require not just the "syntax" of communication (email, audio-conferencing, video-conferencing, virtual whiteboards, etc) but also the "semantics" of processes (inputs, outputs, objectives, sign-offs, etc) to support the work carried out. Such process support has had patchy take-up and mixed success.

The first generation of process support, Workflow, was based on Petri Nets which have a fixed connection structure. With these tools processes could be well-designed but not flexible - and open-ended human activities like problem solving and design could not be supported at all.

The second generation of process support, business process management (BPM) systems, is based on the pi-calculus ([1], [2]) which has a dynamic connection structure. Such tools have greater reach and they permit new kinds of flexibility which they constrain by distinguishing private processing from a shared 'public process', which is a kind of contract between the participants, consisting of an agreed and stable set of interactions.

But this introduces a new problem: in reality most public processes are not fixed structures of simple atomic messages, but flexible sets of complex extended transactions in which the number of possible process loops rapidly tends towards infinity. We illustrate this by reference to concurrent engineering where interactions like agreeing sub-contract terms, signing off specifications, concluding project stages, and so on [3] all involve several parties, contain processing spread over multiple systems on different platforms owned by different organizations, and require a number of iterations to complete successfully. (And this is an environment in which mistakes are expensive. If concurrently performed work unexpectedly turns out to be incompatible then rework is required and projects are delayed.)

In this paper we describe this problem in detail, present a general solution as a pattern of transactions and dependencies, show the inherent requirement for a notion of transaction that is different from that found in database systems, and provide a process representation for the model which can be implemented directly.

2 The Collaborative "Process" Problem

It is common to describe a collaborative effort as a "process". However, when we try to model such an effort using conventional techniques, the activities and their inter-relationships rapidly become impossible to define. At a high level there are usually recognisable stages and patterns but as we descend into the detail, every project is different and the number of possible permutations explodes. In effect conventional process models only describe a kind of management level fiction; the real

activities of people and the logic of how they go about them are rendered completely invisible and unmanageable.

For example, consider design engineering. The problem is not in the engineering discipline itself; it is in the modelling constructs used to describe it. They are derived from programming techniques and designed to describe processes that can be hard coded and then repeated dozens, perhaps millions of times with only very specific variations - the kind of process for which automation is the logical conclusion.

But human activities are usually very different in character. In particular:

- *The precise outcome is often not known.*

The unique value that humans add is in creating new solutions.

- *Inter-disciplinary collaboration is often fundamental.*

In a machine-like process, the objective is to eliminate boundaries, to integrate all the participating sub-processes into a seamless virtual system. In human collaboration by contrast, the different resources, skills and responsibilities of the various players are the very reason for involving them.

- *Action is directly linked to knowledge.*

In a machine-like process, once it is defined, there is no requirement for knowledge - only correct information. Humans, by contrast, add value in the way they apply abstractions (true knowledge) to particular situations.

- *Quality is often as important as efficiency.*

A machine-like process delivers a defined outcome at minimum cost and in minimum time. But in collaborative problem-solving there are more complex trade-offs between cost, time and the quality of the solution which have to be negotiated dynamically.

So, in the real world, engineering managers are dealing with activities that are only partly predictable when the "process" commences. It is a fundamental characteristic of the whole environment that it is organic and dynamic so hard coded process descriptions miss the point - even if they are supported by tools that make modification of that process very easy.

There is a parallel problem in the structure of the product itself. A complex product like an aircraft or a ship may appear to decompose neatly into systems, assemblies, sub-assemblies and parts but in fact it is a densely tangled

set of solutions to a huge and varied set of requirements and constraints. A single requirement may be met by a combination of structure, systems and software, for example, while a single part such as a wing rib may be constrained by structural, aerodynamic, fuel, electrical and manufacturing requirements.

Later modifications only increase the complexity. New connections and dependencies are introduced by tracing the impact of modifications, defining and implementing knock-on changes, testing the whole package and restoring the integrity of the product.

3 A Transactional Model of Collaborative Engineering

We present a general solution of this problem as a pattern of transactions and dependencies. The approach taken is in effect the opposite of conventional project planning. Instead of starting from the top with a well-defined process and product structure and then watching it break down as we descend into the detail, we start from the bottom with the most unstructured case and then see how structure might emerge from it.

Let us imagine that an unexpected issue has arisen in the middle of a large project. It is sufficiently serious to warrant specific management attention and potentially it impacts several design domains. Initially we have no plan or structure for this new sub-project but there are a number of significant things that we can say:

1. We have none of the usual elements of a "process" but what we can talk about is a *pattern*. We know how collaborative problem solving works.
2. A Collaborative Problem-Solution *pattern* has a number of recognisable aspects:
 - a. The input will be a *problem definition* which may require further investigation and development.
 - b. The (successful) outcome will be a *solution*, probably consisting of a set of *solution components* in different domains.
 - c. There will be a set of *models* and *tests* which demonstrate that the solution solves the defined *problem*.
 - d. There may well be interface agreements and other supporting documentation.
 - e. There will be set of dependency relationships connecting all these objects in different ways.

- f. There will be a number of participants in the solution process whose contributions have to be coordinated.
 - g. There may well be distinct phases of activity such as problem definition, concept design, detailed design and testing, integration and testing, etc.
 - h. Within each phase, a lot of activity will be iterative.
 - i. One of the outputs of each phase will be a refined definition of future activities.
3. The components of the *solution* are an interdependent set which must be protected from further change unless the whole solution is revisited and retested. The set of solution components forms a *transaction* - somewhat like a database transaction (see discussion below) - in which they are all committed to the product design together.

Moreover, the *solution transaction* forms part of a bigger *transaction* which includes the *problem definition*, the *models* and *tests* and all the other supporting documentation that is related to it.

4 Human Transactions are Significantly Different from Machine Transactions

The components of a design problem-solution need to be managed as a transaction, somewhat like a database transaction (as we suggested above) - but there are important differences. Database transactions need to be strictly ACID - *Atomic*, *Consistent*, *Isolated* and *Durable* - especially in a distributed context. Human transactions are less rigid:

- In a database transaction, all the elements must be committed together as a single, indivisible operation or not at all because if any part fails, the system as a whole will be left in an inconsistent state. Money could be made to appear or disappear, for example, if a bank transfer was only partially successful.

But the manager of a design project might proceed with an incomplete solution if they think the risk is low.

- The notion of rollback is not applicable. Incomplete solutions are not un-designed and important lessons may be learned from solutions that are rejected for the final design.

- It is extremely unlikely that any particular problem can be defined and solved in complete isolation from all others. Other external relationships have to be defined as part of the transaction and maintained from then on.

A collaborative interaction may not be ACID but it is necessary to regard it as a transaction in order to understand "where you all are" in the process. Without such control, the parties may differ in their interpretations of events and a process support system intended to facilitate cooperation will in fact derail it - especially in later rework and modification.

Existing proposals for extended transaction management in distributed, loosely-coupled computer systems include *BTP* (from OASIS, [16], [17]), *ASF* (from the OMG, [20]), and *WS-TX* (from Microsoft, BEA and IBM, [18], [19]). These proposals do cater for situations in which ACID requirements are relaxed, and deal with situations that may be long-running, but they do not transfer well to the "softer" situations typical of collaborative human processes:

- *Context*. Context propagation is an essential aspect of all 3 specifications - each provides a sophisticated mechanism for the creation and management of a "coordination context", a space that can be shared by the parties to a transaction. But in human-oriented transactions, coordination is driven more by periodic agreement than by continuous synchronisation, and even at synchronisation points the different parties may agree to retain different versions of shared information.
- *Compensation*. In a machine transaction, however long-running, it is necessary to cater automatically for failure of particular tasks (such as a payment) by invoking a "compensation" - a set of activities which ties up loose ends and restores consistency. With human transactions the compensation can rarely be defined in advance. The action that should be taken when a team member leaves a project, for example, or a design is rejected, will depend on the precise circumstances.
- *Completion*. All 3 proposals include a notion of "2-phase commit" drawn from database theory - although this can be relaxed somewhat in different circumstances. But the work done during a collaborative human transaction is generally persisted (archived) by default, whatever the final outcome. And the agreement itself is likely to be a set of approvals, rejections, qualifications, comments, requirement changes, revisions to plans and so on. Such outcomes cannot be usefully captured via the binary "Yes/No" notion of completion inherent in *BTP*, *ASF* or *WS-TX*.

We need a more appropriate modelling technique.

(including documents and other objects).

5 Role Activity Diagrams Represent Human Collaboration Well

Originally developed to describe collaborative software development, *Role Activity Theory* (RAT, [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]) has been in use in engineering for twenty years but while it has been used successfully by analysts for process modelling, its take-up by IT developers for process management has been hindered by lack of tool support. However, with the advent of new web technologies, a product *RADRunner* has become available which provides such support (<http://www.rolemodellers.com>, [15]). The theoretical advantages of Role Activity Theory can now be turned into new management capabilities.

The key features that differentiate RAT from other process representations are:

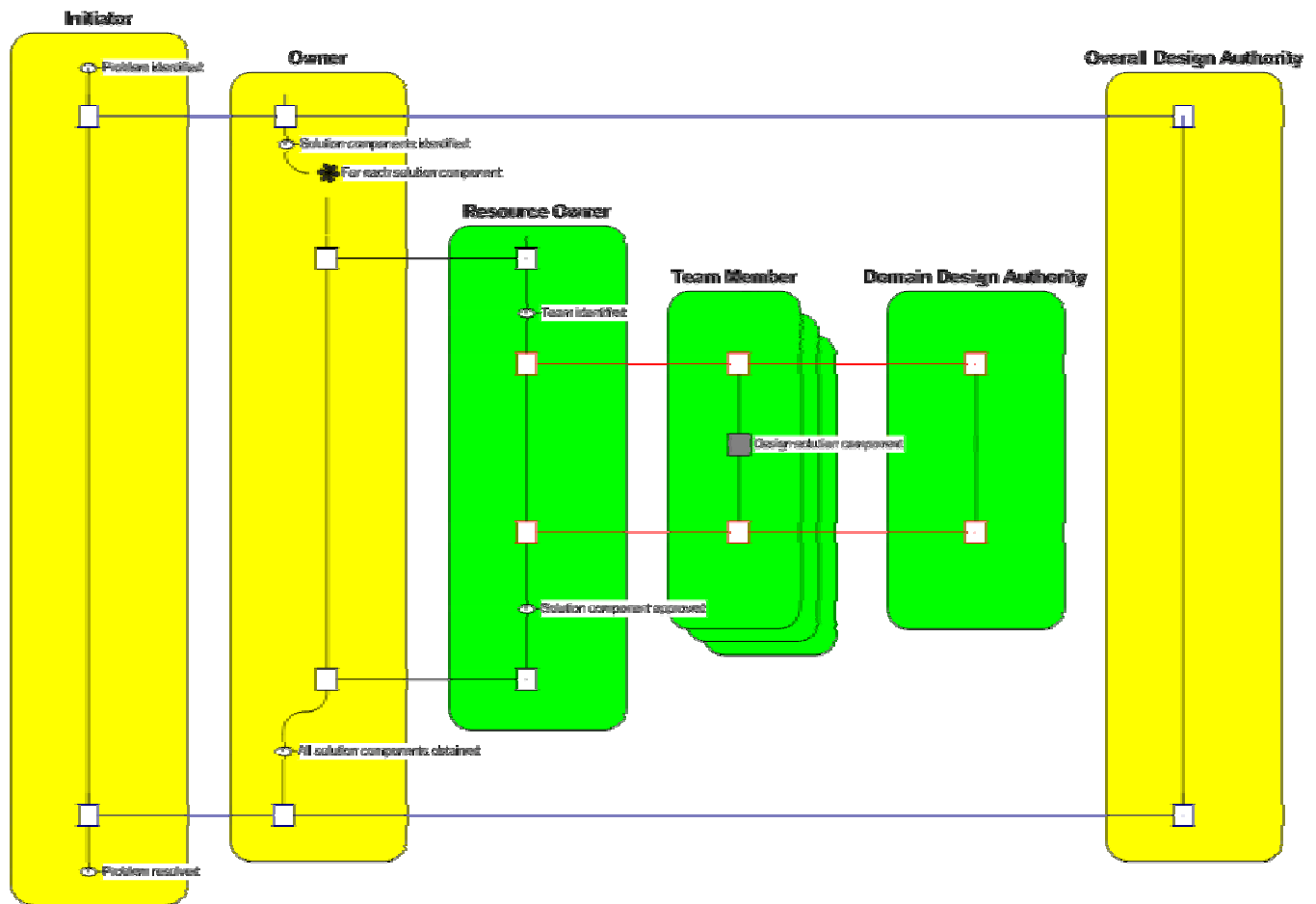
1. *It begins with a strong representation of different Roles.* Each one is assumed to have different responsibilities, authorities and information resources. They participate in process in different ways and they collaborate by passing messages

Processes are not strictly sequential. An expected sequence can be represented graphically but in fact every activity has a before and after condition and it is these conditions that trigger other activities. Processes will therefore loop automatically if the conditions dictate and free-floating process fragments such as monitoring and auditing can be triggered at any time.

3. *Roles can instantiate other Roles, and Processes can trigger other Processes.* The RADRunner tool, which provides digital support to processes based on RAT, allows this to happen from within a process. A process can thus be developed and modified by its participants in a visible and manageable way - exactly what design engineers actually do.

Note that RADRunner permits compliance with any of the above machine-oriented protocols (BTP, ASF, and WS-TX) for particular transactions, where they are fully automatable and/or where they communicate with systems utilising such a transaction model.

Figure 1: Simplified Role Activity Diagram of a collaborative design activity



6 Graphical Modelling With Role Activity Diagrams

Role Activity Diagrams (RADs) offer various benefits for modelling collaborative behaviour over related process depictions such as UML *swim lanes*. For instance:

- Roles are not generic objects, but groups of activity that map directly to human and organisational concerns (*goals* and *responsibilities*).
- A Role may be of any width, permitting parallel activities to be shown side by side.
- Interactions are highlighted as of primary importance and the representation of interactions is powerful and realistic - for example, more than 2 parties can be involved.
- State changes are clearly identified and used to govern future behaviour.
- The sequencing of activities is not procedural, but event-driven and declarative.
- Fragments of behaviour can be depicted in a realistic way.

Collaborative design might be represented as in Fig 1, with 3 roles at the overall design level, and another 3 at the level of the specific design domains or disciplines. There is a degree of correspondence between these levels which indicates that the model is natural:

- *Overall design level*
 1. *Initiator*. This role acknowledges that the problem is real and kicks off the solution process.
 2. *Owner*. This role is responsible for promoting and reporting on progress. They may or may not participate as a team member.
 3. *Overall Design Authority*. This role is responsible for the integrity and quality of the whole design. They give permission for the integration of the specific solution.
- *Domain design level*
 1. *Resource Owner*. This role assigns expert resource to the problem at the request of the problem Owner.

2. *Team Member*. This role participates in the collaborative solution activity and develops the domain-specific component of the solution.
3. *Domain Design Authority*. This role is responsible for the domain-specific aspects of the integrity and quality of the design. They give permission for the domain-specific component of the solution to be released for integration into the overall design.

Different roles, such as 1 and 3 in each level, may well be performed by the same person, although this is not shown explicitly. A more detailed explanation of this pattern is given in the Appendix below.

All the interactions may repeat ad infinitum, as requirements evolve and the solution shapes up. Iterations and loops of rework are not shown explicitly on the RAD but the modelling technique allows for it from the start. If a design is rejected, for example, the 'before' condition automatically becomes true and further design work is automatically enabled. One corollary is that any project plan derived from a RAD will be very dynamic; GANTT charts and so on will have to be regenerated on a regular basis. This is a normal part of project management, and must be catered for naturally if we are to link process modelling and project planning (see below).

7 Transactions and Dependencies

The RAD shows 2 *collaborative transactions* - one at overall design level, and one at domain design level, each bounded above and below by a 3-way interaction. In each case the interaction at the top represents an exchange of requirements whilst the interaction below represents an exchange of solutions - both of which will be complex compound objects. The lower bound interaction, for example will probably consist of:

1. The fully developed problem statement.
2. The objects which form the solution, contributed by different Roles.
3. The models and tests that justify the solution.
4. A map of the dependencies between all these objects (for use tracking the impact of changes).
5. An approval chart, showing who has approved what.
6. Revised process plans for the remainder of the collaboration.

The approval chart and other dependencies can be maintained automatically. Whenever a participating Role submits a new version of one of their deliverables, all the

approvals which are dependent on the new deliverable are removed and new approval activities are triggered. Processes and object relationships interact - as long as they are both represented.

We are developing an accompanying modelling technique which can be overlaid onto raw RAD notation, to show the transactions and the objects that are used and created within them. We wish to depict the dependencies between objects separately from the sequential relationship between activities and represent the relationships between them explicitly. These quite separate concepts are simply confused in conventional process diagrams.

8 Benefits of This Approach

Benefits that this approach can offer to managers include:

1. *Visibility of design progress* - even as the process evolves.
2. *The ability to track key decisions and approvals for audit purposes.* All necessary information is stored explicitly in the compound object passed via the lower bound interaction.
3. *The ability to manage transactions* - for example by combining them or splitting them into sub-transactions. This may be particularly useful when some parts of a solution can be created much faster than others, and it is desirable to free up some people to move on before the entire transaction is complete. In the short term we aim simply to make it easier for the manager to understand what is going on and to make manual changes but in the long term we aim to provide automated features.

"Partial sign-off" of deliverables can be managed by completing the original transaction and creating a new one to produce just the missing piece (like a part-paid invoice). The new transaction can then manage a simpler dependency relationship, between the missing piece and the overall solution.
4. *The ability to manage dependencies.* For instance, the dependency map can be used to identify mutual dependencies, which always cause process problems. Interface contracts can then be introduced to make design activities more independent.

9 Next Steps

We are setting up a case study with in the aerospace sector in order to:

1. Investigate the applicability of the general solution.

2. Develop the necessary overlay of new modelling techniques.
3. Identify how the approach can be integrated with planning and project management.

This would allow project and process management, making them both more realistic and more powerful.

4. Discover how engineering designers and managers can learn from design work, and incorporate relevant knowledge into their specialist domain.

Institutionalising such improvement involves both a deep understanding of the metrics appropriate to creative, exploratory, collaborative human activities (in order to improve future design processes) and a means of knowledge capture and transfer that can be used from within a design process.

10 Conclusions

Role Activity Theory (RAT) is an entirely generic way of representing processes and *RADRunner* is a similarly generic tool. The applicability of RAT to engineering is not based on any particular dedicated features but rather its ability to handle the required flexibility.

Other tools will be needed to manage product structure, models, dependencies and authorities but each of these has their place in a process pattern that can be instantiated and managed by *RADRunner*.

The approach allows process integrity to be guaranteed without the need to implement BTP, ASF or WS-TX - but compliance with any protocol is possible for particular transactions, where they are fully automatable and/or communicate with systems utilising such a transaction model.

The process pattern identified by reference to concurrent engineering is the starting point for development of a graphical methodology for Role Activity Diagramming, which not only provides the means for management of complex collaborative processes but also integrates this management activity with project planning.

Next steps include the development of extensions to Role Activity Diagrams which will identify transactions and show the objects that are used and created within them. This work offers the potential to object and task dependencies more manageable, to provide an automated means of improving efficiency in collaborative work and to track process enactment for audit and improvement purposes.

11 Appendix: A more detailed view of the collaborative design pattern

A Collaborative Problem-Solution pattern has a number of recognisable aspects and phases although these are probably not strictly sequential. In fact, in simple cases, where the problem is well understood and the form of the solution is obvious, then many of these stages can be short circuited.

Note that the terminology used here is not derived from any specific engineering discipline but is designed to lift the pattern out of the concrete details.

1. *Identification and recognition of the problem.*

The identification may come from anywhere but someone with appropriate authority has to acknowledge its existence, allocate resource to it and if necessary modify existing plans.

This points to a specific role that we might call *Initiator*.

An early priority is to appoint a manager who will manage the activity through its lifecycle and whose responsibilities will include promoting and reporting on progress.

This points to another specific role of *Owner*.

2. *Refinement of Problem Definition.*

The problem will need initial investigation and articulation. If the impact on different domains is not well understood then this in itself may be a collaborative activity.

If it is to be opened up to innovative solutions, it is important to *abstract the problem* and to root out unwarranted assumptions about how it should be solved.

3. *Assembling the Design Team.*

The design team will be assembled and set to work on the detailed design of the solution. The "Owner" must have the ability to appoint and remove team members, and to make and modify plans from within the "process" itself.

This points to another specific role of *Team Member*.

Negotiation for resource points to the specific role of *Resource Owner* within each domain.

The responsibilities of the team members include:

- a. Being the *Owner* of the problem within their own domain, promoting and reporting on progress - especially if other engineers are involved.
- b. Connecting their own specialist knowledge to the problem.
- c. Representing their own knowledge to other team members so that their potential contribution to a solution can be understood.
- d. Listening to other team members and understanding their potential contributions.
- e. Thinking collaboratively and creatively about the overall solution.
- f. Learning from the activity and incorporating relevant knowledge into their specialist domain.

4. *Agreement on the Concept.*

At the same time engineers will be looking for solution concepts. The abstract representation of the problem allows for multiple concepts which may be quite different in the way they involve different domains and which will compete with each other for adoption. The output of this phase will be an *agreed concept*.

Agreement on the concept allows a number of other activities to be completed:

a. *Development of Models and Tests.*

There will be a specific activity of developing the methods by which the solution will be proved. (These should be independent of the design of the solution but the details will probably be dependent on the chosen concept.) If the issue is safety critical, formal tests will be essential. "Real" tests on a physical prototype may also have to be added to the overall project plan.

b. *First estimates of time and resource consumption.*

These will be continuously updated and revised.

5. *Design of the Solution.*

The team activity has recognisable phases although in practice work is likely to be iterative:

- a. The overall *concept* has to be refined. In effect it now has to serve as a *contract* between the different engineering domains, documenting what

their contribution has to provide and how it will interact with the contributions of other domains.

- b. The team members develop their own specific *solution components* in detail. These contributions require approval from an appropriate authority within their domain.

This points to the specific role of *Domain Design Authority*.

- c. The separate components of the solution are *integrated and tested*.

6. *Implementation of the Solution.*

The solution is implemented and frozen. Somebody with appropriate authority includes the *complete solution* (as a transaction) in the overall product design so that it cannot be revisited without specific authorisation.

This points to the specific role of *Overall Design Authority*.

References

- [1] Milner, R., 1999, "Communicating and mobile systems: the pi calculus", Cambridge University Press, ISBN 052164320
- [2] Smith, H., Fingar, P., 2003, "Workflow is just a Pi process", <http://www.bpm3.com/picalculus>.
- [3] Murdoch, J., et al., 1999, "Modelling Complex Design Processes with Role Activity Diagrams", Journal of Integrated Design and Process Science, 1999.
- [4] Holt A. W., Ramsey H. R., Grimes J. D., 1983, "Coordination system technology as the basis for a programming environment", Electrical Communication, Vol. 57(4).
- [5] Greenspan S.J., 1984, "Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition", University of Toronto Report CSRG-155
- [6] Ould M.A., Roberts C., 1986, "Modelling iteration in the software process", Proceedings of the Third International Software Process Workshop, Breckenridge, Colorado, USA, IEEE Computer Society Press.
- [7] Ould M.A., Roberts C., 1987, "Defining formal models of the software development process", Software Engineering Environments, Pearl Brereton (ed.), Ellis Horwood.
- [8] Snowdon R.A. , 1988, "A Brief Overview of the IPSE 2.5 Project", Ada User, Volume 9, No. 4.
- [9] Roberts C., 1988, "Describing and acting process models with PML", Proceedings of the Fourth International Software Process Workshop, Moretonhampstead, Devon, UK, IEEE Computer Society Press.
- [10] Roberts C., Jones A., 1990, "Dynamics of process models in PML", Proceedings of the Fifth International Software Process Workshop, Kennebunkport, Maine, USA, IEEE Computer Society Press.
- [11] Snowdon, R.A., Warboys, B.C., 1994, "An Introduction to Process-Centred Environments," Software Process Modelling and Technology, A. Finkelstein, J. Kramer and B. Nuseibeh (eds.), Research Studies Press, Taunton, England.
- [12] Ould M.A., Huckvale T., 1995, "Process Modeling - Who, What, and How: Role Activity Diagramming", Business Process Change: Concepts, Methods and Technologies, Idea Group Publishing
- [13] Ould M. A., 1995, "Business Processes: Modelling and analysis for re-engineering and improvement", Wiley, ISBN 0-471-95352-0.
- [14] Warboys B.C., et al, 1999, "Business Information Systems: a Process Approach", McGraw-Hill, ISBN 0-07-709464-6.
- [15] Harrison-Broninski, K., 2003, "From Requirements To Roll-out Immediately", <http://www.rolemodellers.com/download/unlicensed/From Requirements To Roll-out Immediately.pdf>.
- [16] Cepenkus, A., et al, 2002, "Business Transaction Protocol - An OASIS Committee Specification", version 1.0, http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf.
- [17] Potts, M., et al, 2002, "Business Transaction Protocol Primer", version 1.0, http://www.oasis-open.org/committees/download.php/2077/BTP_Primer_v1.0.20020605.pdf.
- [18] Cabrera, F., et al, 2002, "Web Services Transaction (WS-Transaction)", <http://www.ibm.com/developerworks/library/ws-transpec/>.
- [19] Cabrera, F., et al, 2002, "Web Services Coordination (WS-Coordination)", <ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf>.
- [20] Object Management Group, 2003, "Transaction Service Specification", version 1.4, <http://www.omg.org/cgi-bin/apps/doc?formal/03-09-02.pdf>.