Modelling Parliamentary Workflows a Case Study in Belgian Parliaments

Christophe Ponsard[1], Gaetan Deberdt[2], and Joël Tournemenne[3]

[1] CETIC Research Center, Charleroi (Belgium) - cp@cetic.be
[2] Parlement de la Communauté Française (Belgium) - gaetan.deberdt@pcf.be
[3] Parlement Francophone Bruxellois (Belgium) - jtournemenne@pfb.irisnet.be

Abstract. Parliament work is regulated by a number of democratic rules about the way laws are proposed, discussed and finally voted. Despite a number variations, most parliaments share the same kind of workflow supported by one or two assemblies. Such workflows are most of the time described by a regulation stated in natural language and generally approved by the assemblies themselves. This document is subject to some interpretation, especially by the administration responsible of the day to day management. Currently this management is also on-going strong electronification with even a direct exposure of the parliamentary work on the internet for better transparency and control by the citizen. In this paper we report about our work of modelling the parliamentary workflows, starting from the official documents and in-place systems. The aim of this work is multiple: first, discover potential ambiguities and inconsistencies, then compare how similar are a number of parliaments and finally see how those models can be translated in the computer systems, especially in the perspective of the open-sourcing and mutualisation of such systems among different parliaments. Our practical experience of applying various modelling techniques is reported and discussed using two of the seven (!) parliaments running in Belgium. This comparison work relies both on a set of modelling requirements for such systems and on the SEQUAL reference framework for assessing the quality of models.

Key words: e-government, parliament, modelling, workflow, mutualisation

1 Introduction

All democratic countries of the world run some kind of parliamentary system whose main functions are to make law and control the work of the executive power, following the principle of separation of powers. Parliaments may consist of chambers or houses, and are usually either bicameral or unicameral. In bicameral systems, the lower house is almost always the originator of legislation, while the upper house is usually the body that offers the "second look" and decides whether to veto or approve the bills [23].

Law making also follows a general common process, starting from a bill either proposed by the executive or legislative body, then discussed by the assemblies.

After preliminary readings, it is generally sent to specialised committees which will work on it. This will result in a number of amendments and finally a vote. In case of adoption, the law is then promulgated by being officially signed by the authority (e.g the President or the King) and finally published. It is then generally followed by executive laws to enforce it. The following figure show this process for the Australian (bicameral) parliament.

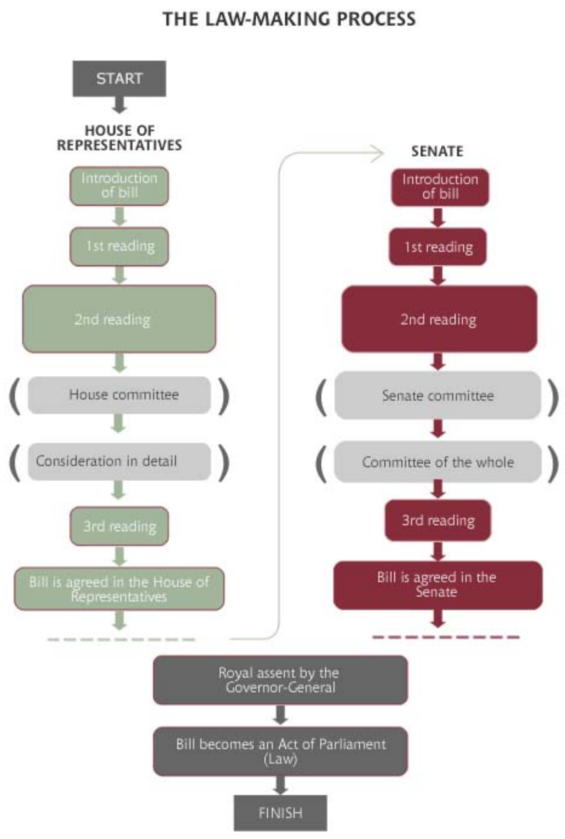

Fig. 1. Typically Parliamentary Workflow [15]

With the raise of ICT, e-democracy is on its way and is present at various levels: e-voting, e-forms, e-referendum... and among them e-legislation which is the part we are interested in here. The electronification process has already started in the 90's at the data and document level (scanning, OCR, meta-data, automated generation of documents, diffusion on web-site). More recently it is reaching the legislative processes themselves. After initial phases of incertainty and euphoria, this evolution is now reaching some maturity and being "institutionalised" [4][19]. The main goals identified in this process are to improve:

- *the procedural quality:* better modelling (less complex, less operational), supporting evolution and re-engineering;

– *the output quality:* drafting systems for improving the formal quality of legislation and regulatory impact assessment for improving the material quality of legislation;
– *the participatory quality:* introducing new communication tools into the representative system or even more visionary concepts for new democracy models.

Most parliaments have now a strong ICT department in charge of this work. As parliaments have the same business, this also means that they need the same kind of solution. Rather than reinventing the wheel, some assemblies have started to collaborate and mutualise their efforts, this is especially true in Belgium which has a complex organisation with many assemblies at region, community and federal levels. This effort also requires to be able to know precisely the commonalities and differences between those assemblies and thus to model them precisely.

This paper reports about a practical case-study done in two regional assemblies of Belgium in the context of mutualising their development with the longer term goal to open-source the resulting more generic software [7]. Those assemblies are the Parliament of the French Community (PCF in short) and the French Parliament of Brussels (PFB in short), which are respectively a medium-size and a smaller size parliament. The first step of this study was to precisely model those two assemblies, starting from the existing situation as documented in the regulation issued by the assemblies themselves and as observed on the field [16].

This paper is structured as follows. In section 2, we will discuss about the requirements on the adequate language to capture parliamentary workflow. Then, in section 3, we will see how a number of candidate languages fit those requirements by showing selected parts of our case studies. Section 4 will compare those models based both on the previous requirements and on a reference framework for assessing model quality. Based on this, a number of important lessons learned from those models will be discussed. Finally, section 5 will draw some conclusions and perspectives.

2 Requirements on the Modelling Language

The main requirements discovered during the study were the following:

– *[BEHAV] Ability to capture behaviors.* The language must be able to capture the dynamic nature of the parliamentary workflows.
– *[RESPO] Ability to capture responsibilities.* The language should be able to describe the various agents playing some role in the system and their responsibilities. More precisely what they control and under which circumstances.
– *[GOAL] Ability to capture the goals.* The language should be able to capture underlying goals of some operational construct. Goals can be either functional or non-functional (such as security, reliability, etc.)

– *[PRECISE] Precise language.* The language should be precise and unambiguous.
– *[UNDER] Easy to understand.* The language should be accessible to non specialist for validation purposes. Languages should preferably have a graphical semantics associated with it.
– *[TOOLS] Tool support.* The language should be supported by tools at modelling level and at run-time level, either directly or through some model transformation.

3 Study of Selected Languages

This section reports about various modelling techniques used to model parliamentary work. It does not claim to present all relevant techniques in an exhaustive way. A useful reference for this is [24].

3.1 Use Cases and Sequence Diagrams

A use case is a description of a system's behaviour as it responds to a request that originates from outside of that system. Use cases, stated simply, allow description of sequences of events that, taken together, lead to a system doing something useful.[2] Each use case describes how the actor will interact with the system to achieve a specific goal. One or more scenarios may be generated from each use case, corresponding to the detail of each possible way of achieving that goal (or possible exception/failure).

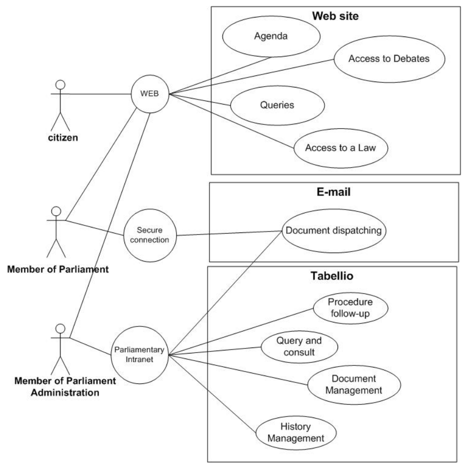

Fig. 2. UC Context Model of a Parliament Management System.

Notations for Use Case include UML Use Case (graphical) [11] and template-based descriptions (textual) [5]. They are usefully complemented by sequence diagrams for graphically describing the generated scenarios with a very comprehensive view of the system with the time on the vertical dimension and the interaction between agents structured horizontally. Note while UML 1.X sequence diagrams were limited to rough traces, UML 2.X supports many structuring operators like conditionals, options, even loops, with the danger to capture too much complexity in a single scenario.

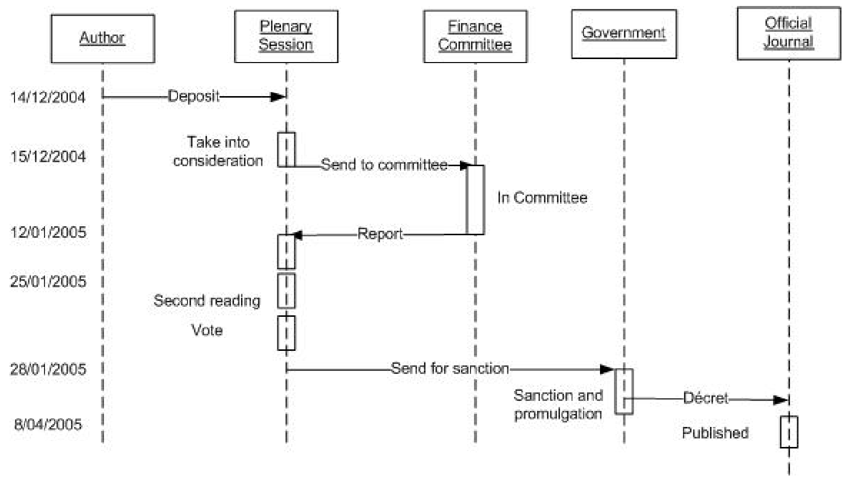

Fig. 3. Sequence Diagram for some procedure.

UML Use Cases diagrams have a good capacity to capture the context of the system and the general responsibilities but lacks the capacity to reflect the dynamic behavior. Textual templates can describe some part of the behavior but not very precisely. Sequence diagrams used together enable a more precise capture of behaviours but generally partial and at instance level. Goals can be captured using methods like [5] however generally mainly at functional level.

3.2 Goal Models

From [22], a goal is an objective the system under consideration should achieve. Goal formulations thus refer to intended properties to be ensured; they are optative statements as opposed to indicative ones, and bounded by the subject matter. Goals may be formulated at different levels of abstraction, ranging from high-level, strategic concerns (such as "Efficient Management of Parliamentary Work") to low-level, technical concerns (such as "Publishing of Voted Laws on Parliamentary Website"). Goals also cover different types of concerns: functional concerns associated with the services to be provided, and nonfunctional concerns associated with quality of service - such as safety, security, accuracy, performance, and so forth. Within the scope of this paper, we will use the KAOS goal-oriented language [8].

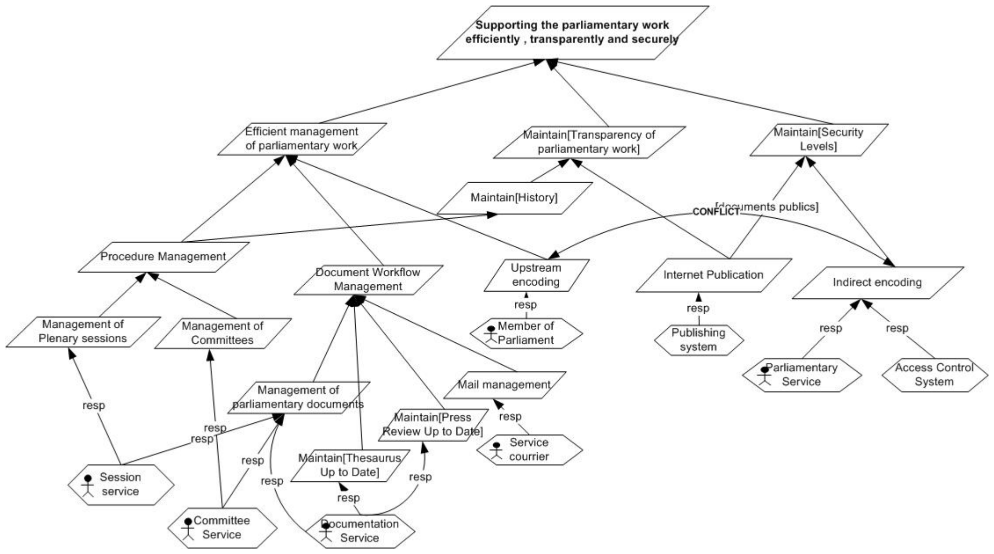
Supporting the parliamentary work efficiently , transparently and securely
Efficient management of parliamentary work
Maintain[Transparency of parliamentary work]
Maintain[Security Levels]
Maintain[History]
[documents publics]
CONFLICT!
Procedure Management
Document Workflow Management
Upstream encoding
Internet Publication
Indirect encoding
Management of Plenary sessions
Management of Committees
resp
Member of Parliament
Publishing system
Parliamentary Service
Access Control System
Mail management
Management of parliamentary documents
Maintain[Press Review Up to Date]
Service courrier
Maintain[Thesaurus Up to Date]
Session service
Committee Service
Documentation Service

Fig. 4. Goal Model of the Parliament Administration.

Goal models enable to capture, structure and reason about system properties and agent responsibilities. Languages like KAOS have precise semantics. The goal level is defined using temporal logics [14], semantics refinements and operations are also precisely defined [9][13]. Not however that the operational level is not very practical to use especially to describe workflows as the language is not designed for this level.

3.3 Final State Machines and State Diagrams

A finite state machine (FSM) is a model of behavior composed of a finite number of states, transitions between those states, and actions. FSM have been extended by Harel to statecharts to allow the modeling of superstates, concurrent states, and activities as part of a state. This notation is now standardised in UML State Diagrams [11].

State Diagrams are very popular. They are very easy to understand and thus to use to validate a behavior even with non-experts. The hierarchical structure allows also the system to be nicely described at progressive levels of details. There are precise semantics although several alternative semantics have been defined, leaving possible ambiguities but generally for specific cases. Goals can be associated with a FSM for example as invariant or obligation an FSM should enforce. This can be verified using model-checking tools.

FSM are also supported by tools for simulating the system or generating the behavioral part of the code (e.g. Rhapsody [21]). It is also easy to design such a generator. In our case study, the company responsible of the system development has such a framework, called XOooF which is now open-source [20]. The framework supports the partial generation of the application code

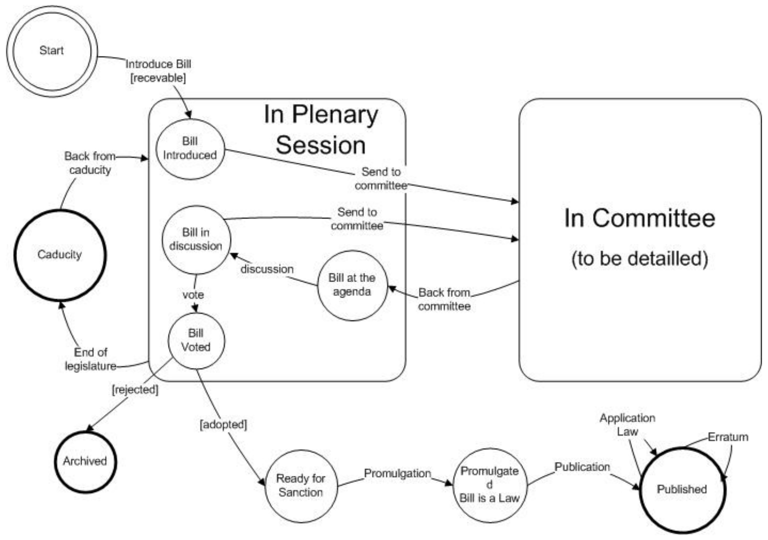

Fig. 5. Final State Machine for the Journey of a Bill.

from XML-based description of state machines. Some aspects not covered are persistency, advanced transactions and graphical user interfaces. Several target languages such as VB/COM, C#, Java and Python are supported.

3.4 Business Process Oriented Languages

Many notations have developed for modelling business processes, with different coverages (activities, products, decisions, context), specification levels (organisation, orchestration, web-services) and underlying semantics. To leverage this, BPMN (Business Process Modelling Notation) is a current standardisation effort aiming at unifying the expression of basic business process concepts (e.g., public and private processes, choreographies) as well as advanced modelling concepts (e.g., exception handling, transaction compensation) [1]. The connection of BPMN with more operational standard such as BPEL (Business Process Execution Language) is however not entirely solved as discussed in [17] but seems to be evolving favorably.

UML - more software-oriented - also support this kind of modelling through the activity diagram which can represents business and operational step-by-step workflows of components in a system. Activity diagrams can be unstructured or organised using swimlanes (somehow similar to sequence diagram lifelines) which enable a better capture of the action responsibilities. Figure 6 shows a typical process model of the parliament work using those notations.

At semantic level, BPMN remains semi-formal although quite complete. Some attempts have been made to more deeply formalise parts of it [3]. Back to our example described with UML, the semantics were changed between UML

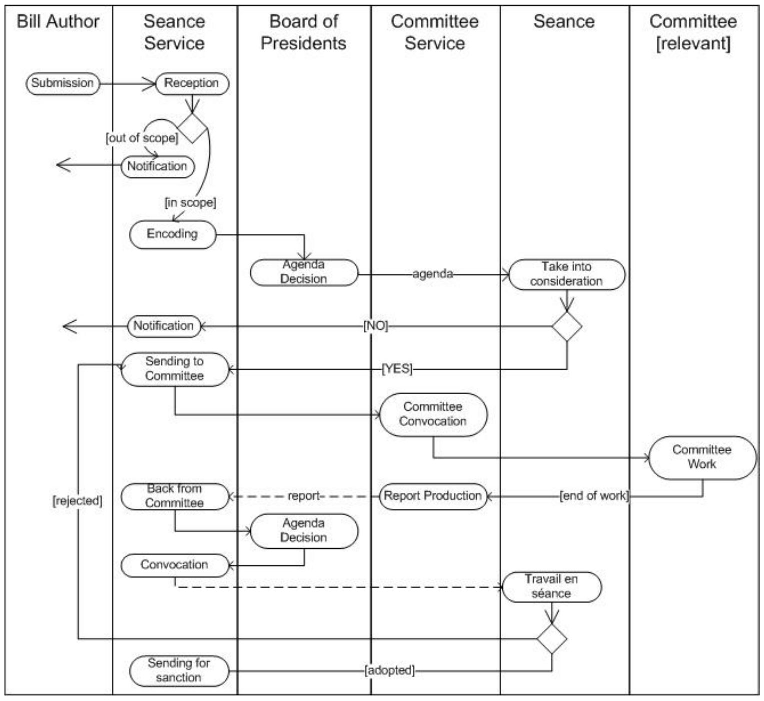

Fig. 6. Activity Diagram for the Journey of a Bill.

1.x w (variation of the UML State Diagram) and UML 2.x (semantics based on Petri nets) [18]. This is a good evolution as Petri nets have better mechanisms for controlling concurrency and synchronisation which is important in workflow management. Petri nets are frequently used as formal underlying model and are also supported by tools (e.g. Flexo was considered for the Belgian case study [10]).

4 Lessons Learned

In this section, we will first compare the qualities of the previous models w.r.t. the requirements described in section 2. This discussion will also rely on the SEQUAL reference framework for assessing the quality of models [12]. The rest of the section will put those conclusions in a wider perspective by going back to the e-government goals defined by Schefbeck [19].

4.1 Comparison Table of Modelling Languages

Table 1 summarises our comparative work based on our requirements described in section 2.

Model	Use Cases	Goal Trees	State Diagrams	Business Process Models
Behaviour	through sequence diagrams	partially	very good, hierarchical, scalable	very good
Responsibility	at context level	very good	poor	through swimlanes
Precision	semi-formal	formal (KAOS)	formal (FSM)	formal (petrinets)
Understand.	good	good	very good	very good
Tools	UML tools	Objectiver	UML tools, Rhapsody	BPEL tools, some UML tools...

Table 1. Modelling Languages Comparison Table (domain requirements)

To consolidate this comparison, we used the SEQUAL reference framework which defines a number of model qualities: empirical, syntactical, semantical, pragmatic, societal, knowledge and language [12]. Those quality factors are compared in table 2. Some of those qualities are already addressed in our requirements: empirical is understandability, semantical is precision. The organisational quality is defined as how well the goals of modeling are reached by the model. This is exactly the purpose of table 1, so this factor is a synthesis of that table.

Model	Use Cases	Goal Trees	State Diagrams	Business Process Models
Empirical	Poor-to-medium (depending on template used)	medium-to-good (depending on refinement checking strategy)	medium (difficult to structure)	good (control flow)
Semantical	semi-formal	formal (KAOS)	formal (FSM)	formal (petrinets)
Pragmatic	good	good	very good	very good
Knowledge	good (capture of scenario/functions)	very good (capture of system goals)	poor (states/transition not directly linked to domain)	good (business process level)
Organisational	medium	medium	medium	good
Language	generic	generic	generic	more specific

Table 2. Modelling Languages Comparison Table (SEQUAL)

The main lessons learned from those tables is that a single language does not fit all our requirements. Activity diagrams seem the most adapted for our purpose given the current evolution of methods and tools while in the past, state machines were more the reference framework.

Other notations are useful to use in a complementary matter. Especially in the reengineering, and comparative study it is important to make sure the goals are fully aligned because variation in goals will inevitably result in variation at the workflow level and it is important to understand if some variation is a design decision or more fundamentally bound to a goal.

4.2 Procedural Quality

The use of modelling techniques helped greatly in the process of understanding the way the assemblies are working, their commonalities and differences.

During the elicitation phase in the first assembly (PCF), the various models were built from a number of sources of domain knowledge: the official regulation of each parliament, interviews with the staff and the documentation of the existing system. Building those models allowed us to have guidelines for completeness (e.g. asking about missing transitions) and for conflict identification (e.g. different actions reported by different sources). It allowed us to discover a number of undocumented choices left open by the regulation and to understand the rationale behind those choices. This resulted in an improvement of the documentation of the procedures, which are not only meant for developing a new system but also helpful as training material for new collaborators.

The work in the second assembly (PFB) did not start from scratch but was carried out based on the models from the first assembly (PCF), assuming their would be only few differences. This assumption was confirmed with the following main differences:

- *Syntactic variations* in the vocabulary used (e.g. the term for a law, for the board of presidents...)
- *Small behavioral differences*, typically variations in some transitions. Those are easily implemented at specification level and propagated to the implementation by regenerating the impacted code.
- *A more fundamental difference is the distribution of roles*: as PFB is smaller, the same people would typically handle a several tasks. As the model was built using roles, this has however no impact on our models.

4.3 Output Quality

Prior to our study, a strong model-based approach was already in place in PCF (based on finite state machines) and partially at PFB (based on a document management workflow).

The impact on the output quality was especially visible at PCF with a chain of model-based tools supporting the whole parliamentary process, from the gathering of minutes to the diffusion of the reports on the website.

The traceability of the parliamentary process is also excellent based on the accumulation of state traces in the system.

4.4 Maintainability and Reusability

The long term goal initiated by the case study is to eventually be able to share common code between assemblies and even to open-source such code. The current closed source model has a number of limits, especially when a number of assemblies share common needs and have to develop their own solutions separately and at high cost. This process has already started under the Tabellio project [7]. A number of generic enough modules have been open-sourced together with the XOoof FSM-based framework.

For the process to be successful, the code quality should however be improved prior to its open-sourcing and this is currently on-going. A major evolution is the transition to a workflow management systems which is not based on code generation as before but on a workflow engine, based on the Plone framework and in coordination with other e-Government initiatives such as PloneGov [6]. Here again, the underlying model proves fundamental has it will drive the configuration of the new system and the definition of the data migration procedure between repositories.

5 Conclusions and Perspectives

In this paper, we explored various way to model parliamentary workflows using different languages. The comparison was driven by a real-world case study and performed using both specific requirements and the SEQUAL reference framework. As expected, a single language cannot fit all requirements and qualities. However business process models seem the most adapted for the needs of modelling parliamentary workflows. Other notations such as goal models allow the analyst to have a deeper insight of the system and to better understand variations between different assemblies and better manage the evolution of a given system.

Among the other lessons learned, the use of adequate models greatly helped in the understanding of the way each assembly was working and how similar they were. Models are also fundamental to deploy tool support. Firstly, in a model-driven architecture perspective, models greatly ease the development of solutions by removing the need to write and test substantial part of the system. Secondly, in a mutualisation perspective, those tools can even be shared together with some representative models and guidelines on how to adapt them. The reuse is then maximal, reducing maintenance costs and allowing easy tuning to the need of other assemblies, especially those of developing countries. This also opens a number of interesting perspectives to further improve the way democracy works: speeding the process, introducing more transparency, etc.

At the methodological level, there is room for many improvements. The comparison work done here is very coarse grained. In order to draw more conclusions about the models and the way to use them, more precise metrics have to be defined and measured together with the reference quality framework. This work will be considered in the current re-engineering phase of the workflow system.

Acknowledgements

This work was financially supported by the Walloon Region and European Union (ERDF and ESF). We also warmly thanks the staff of the respective parliaments.

References

1. *Object Management Group/Business Process Management Initiative*, http://www.bpmn.org/.
2. Kurt Bittnera and Ian Spence, *Use Case Modeling*, Addison Wesley Professional, 2002.
3. M. Brambilla, *LTL Formalization of BPML Semantics and Visual Notation for Linear Temporal Logic*, Tech. report, January 2005.
4. Daniel Brassard, *How can information technology transform the way parliament works ?*, Parliamentary Information and Research Service - Library of Parliamentarians of Canada, 2005.
5. Alistair Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2001.
6. PloneGov Consortium, *The PloneGov Project*, http://www.plonegov.org.
7. Tabellio Consortium, *Tabellio : an Open Source Collaboration for Assemblies*, http://www.tabellio.org.
8. A. Dardenne, A. van Lamsweerde, and Stephen Fickas, *Goal-Directed Requirements Acquisition*, Science of Computer Programming **20** (1993), no. 1-2, 3–50.
9. R. Darimont and A. van Lamsweerde, *Formal refinement patterns for goal-driven requirements elaboration*, 4th FSE ACM Symposium, San Francisco, 1996.
10. Denali, *FlexoBPM*, http://www.denali.be.
11. Martin Fowler, *UML Distilled - Third Edition*, Addison-Wesley, 2004.
12. J. Krogstie and A. Solvberg, *Information Systems Engineering: Conceptual Modelling in a Quality Perspective*, Kompendiumforlaget, Trondheim, 2003.
13. E. Letier and A. van Lamsweerde, *Deriving Operational Software Specifications from System Goals*, FSE'10, Charleston, November 2002.
14. Z. Manna and A. Pnueli, *The Reactive Behavior of Reactive and Concurrent System*, Springer-Verlag, 1992.
15. Australian National Audit Office, *Managing Parliamentary Workflow - Best Practice Guide*, April 2003.
16. C. Ponsard, *A Comparative Analysis of the French Community Parliament and the French Parliament of Brussels (in French)*, http://www.tabellio.org/documentation/manual/analyse-comparative-pcf-pfb-cetic, 2005.
17. J. Recker and J. Mendling, *On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages*, Proc. of 11th Int. Workshop on Exploring Modeling Methods in Systems Analysis and Design, June 2006.
18. W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag New York, Inc., New York, NY, 1985.
19. Gunther Schefbeck, *E-Parliament: Legislative Standards and Good Practice*, Proceedings of International Workshop on E-Parliament: Managing Innovation, Geneva, Switzerland, 2007.
20. SoftwareAG, *XOooF*, http://xooof.sourceforge.net, 2006.
21. Telelogic, *Rhapsody*, http://www.telelogic.com/products/rhapsody.
22. A. van Lamsweerde, *Goal-Oriented Requirements Engineering: A Guided Tour*, Invited minitutorial, Proc. RE'01, August 2001.
23. Wikipedia, *Parliament*, http://en.wikipedia.org/wiki/Parliament, 2008.
24. Michael zur Muehlen, *Workflow-based Process Controlling: Foundation, Design, and Application of Workflow-driven Process Information Systems*, Logos Verlag, 2004.