Preface

The YAWL Symposium, taking place on the 7th of June 2013 in Sankt Augustin, Germany, is an event dedicated to sharing experiences with the YAWL system.

YAWL is an open source Business Process Management environment. Based on the Workflow Patterns Initiative that started in 1999, work on the YAWL Workflow Management System began in 2003. Since then YAWL has become a full-fledged Workflow Management System that has made its way out of the purely academic world into real life applications.

The YAWL User Group, founded 2012 at Bonn-Rhein-Sieg University of Applied Sciences, is the organizer of this symposium and has created the website http://yaug.org as a platform for discussions about the ongoing development of the YAWL system.

The YAWL Symposium 2013 attracted ten submissions which were each reviewed by at least three members of the program committee and all of which were deemed to be of sufficient quality to be accepted for presentation on the day and for inclusion in the proceedings.

We are grateful to all who contributed to YAWL over the years and specifically to those who contributed to this symposium.

St. Augustin, June 2013

The members of the Organizing Committee

Thomas Freytag
Andreas Hense
Arthur ter Hofstede
Jan Mendling

Program Committee Members

Michael Adams	Queensland University of Technology
Carlo Combi	Universita' degli Studi di Verona
Massimiliano de Leoni	Eindhoven University of Technology
Patrick Delfmann	Eur. Research Center for Information Systems
Remco Dijkman	Eindhoven University of Technology
Marlon Dumas	University of Tartu
Marie-Christine Fauvet	Joseph Fourier University of Grenoble
Giancarlo Fortino	Universita' della Calabria
Thomas Freytag	Cooperative State University (DHBW) Karlsruhe
Chinab Hanachi	Toulouse 1 University
Andreas Hense	Bonn-Rhein-Sieg University of Applied Sciences
Manfred Jeusfeld	Tilburg University
Marcello La Rosa	Queensland University of Technology
Susanne Leist	University of Regensburg
Irina A. Lomazova	Russian Academy of Science
Jan Mendling	Wirtschaftsuniversität Wien
Oliver Müller	University of Liechtenstein
Chun Ouyang	Queensland University of Technology
Henderik Proper	Public Research Centre Henri Tudor
Pnina Soffer	University of Haifa
Arthur ter Hofstede	Queensland University of Technology
Wil van der Aalst	Eindhoven University of Technology
Petia Wohed	Stockholm University

Using YAWL in a Business Undergraduate Course on Process Management: An Experience Report

Joerg Evermann, Mary Furey, Terry Hussey

Faculty of Business Administration, Memorial University of Newfoundland, St. John's, Canada
jevermann@mun, mfurey@mun, thussey@mun.ca

Abstract. The paper reports on the use of the YAWL workflow management in a core undergraduate course in a business school. We describe the design of the course, and YAWL's position within it, learning activities and learning outcomes. The use of YAWL in an introductory, general business course, rather than an upper-level information systems course, entails several challenges. We report on these challenges and reflect on lessons learned and best practices.

Keywords. Workflow management, education, business, experience reports

1 Introduction and Background

Workflow management is an important aspect of business process management, and the wider business and management discipline. Approximately 1400 students are enrolled in three different undergraduate degree programs at the Faculty of Business Administration (FBA) at Memorial University of Newfoundland. Of these, the Bachelor of Commerce (BCOM) and the Bachelor of Business Administration (BBA) make up more then 95% of students. Approximately half of the students choose an accounting concentration, another third choose a marketing concentration, with the remainder in organizational behavior, entrepreneurship, or strategy. Due to lack of student interest and enrollment, the faculty does not offer electives in information systems (IS).

An academic program review in the mid 2000s recommended that a course on business process management and modelling should be introduced instead. We began development of this new course in 2009, with a first offering in the fall semester of 2011. The core course is required for the BCOM and BBA degrees, and is usually taken in the second year. The course is taught in classes of 75 minutes, twice a week, for 12 weeks per semester, with class sizes varying between 30 and 60 students.

The FBA is accredited by the AACSB and has, as part of the required assurance of learning process, defined the following learning goal and objectives, which are to be satisfied by and evaluated in this course:

- Students that successfully complete this course will understand business processes and the role of technology in enabling and supporting them. (Goal)
  - Our students will be able to identify, model, and evaluate the efficiency of an organization's main business processes. (Objective 1)

– Our students will be able to improve and redesign a business process using IT and non- IT-based solutions. (Objective 2)
– Our students will be able to identify points of a business process where IT support can improve the overall process, and be able to propose high-level requirements for suitable IT support. (Objective 3)

2 Course Design and the Choice of YAWL

The main challenge was to introduce the technical topic of workflow management into a general business curriculum where students are not enthusiastic about IS related topics. The approach to designing the course must take into account that it is not an IS elective, but is required for all business students and is typically taken in the first or second year of study. Students do not have a background in software technology, programming, databases, conceptual modeling, and other foundational technologies. Because no other courses in the curriculum include business process management, the topic had to be covered from all perspectives: strategic, managerial, and operational.

Experience with a computer science course that was replaced by this course showed that students did not enjoy learning about computer technology. Therefore, we adopted a "gentle" introduction from the managerial perspective to give students a foundation in the concepts of business processes and process-oriented enterprises before introducing them to workflow management. The course has to demonstrate the relevance of the topic and capture student interest as early as possible. Table 1 in the Appendix provides an outline of the typical course topics and reading list.

Our first challenge in course development was a dearth of textbooks that provide a balanced organizational and technical treatment of business processes and workflows. For example, some books cover workflow modeling and workflow systems [1-3], but do not provide a broader managerial perspective. These books are more suitable for IS majors than for general business students. Other books [4-5] focus on specific vendor's products. We wanted to avoid these dependencies, and focus on vendor-independent concepts instead. Further, we could not rely on the support of the faculty to cover the cost of access, training, etc. Yet other books [6] include processes as part of a general course on IS, which would overlap extensively with an existing introductory MIS course. We finally settled on a combination of management articles (see third column of Table 1) and the YAWL book [7].

While the YAWL book is also targeted at IS majors rather than general business students, the choice of YAWL was motivated by the following considerations:

- YAWL provides a vendor-independent perspective on the theoretical concepts of processes and workflows. This allows us, in a university setting, to concentrate on teaching theory rather than product-specific skills.
- YAWL was the only workflow system that is built on a sound theoretical foundation (Petri nets and extensions). We felt this to be especially important in a university setting that focuses on theoretical knowledge, not only on skills.
- The YAWL book and system are developed by leading researchers in workflow management, and thus provides best available knowledge at the "cutting-edge".

- YAWL provides a good integration of a system with a textbook. This combination cannot be found for any other system or product.
- The YAWL system is open-source and (mostly) multi-platform capable. This provided use with free access for the greatest number of students. However, platform-specific components of YAWL, such as the worklet designer, or platform-specific related tools described in the YAWL book, such as CPNTools, meant that these were in principle not suitable for use in the course.
- In contrast to other systems, YAWL allows the easy exchange of workflow and resource models, convenient for homework and exam submissions.

3 Using the YAWL system in the Course

Experience has shown that an early demonstration of the use and benefits of workflow management systems can considerably improve student interest and motivation. We introduce YAWL in the second class of the course as a group activity, using the Genko Oil Case in Appendix A of the YAWL book that is provided as an example with the YAWL system. Applying Kolb's experiential leaning approach [8] where no explanation of the system itself and only little information of the process is provided, students role-play one of the process participants in the company. This allows students to experiment with the system, observe its behavior, and notice the actions they can take with respect to work items. The lecturer maintains an updated administrator view of all cases and work items on the classroom projector. According to Kolb, this type of learning can be defined as knowledge creation resulting from the combinations of grasping and transforming experience. Students are generally impressed by the capabilities of a WfMS and convinced of its usefulness and value. By engaging students to consider the business value generated by WfMS the instructor can overcome the innate technology resistance that is likely present.

After students receive an introduction to Petri nets and workflow nets using the WoPeD tool, we introduce the three perspectives of YAWL, the flow, data, and resource perspective. For each of these, the first class discusses the workflow patterns, and a useful exercise is for students to identify an example for each workflow pattern from their own work experience. The second class on each perspective occurs in the computer lab. Students are asked to form groups of between two and four members. We have developed tutorial scripts that guide students step-by-step through the development and execution of an example process. Students access a shared YAWL server and are able to again experience the capabilities of a WfMS in a realistic environment.

Students also use YAWL in a major assignment. The assignment is based on a Harvard Business School Case ("Pharmacy Improvements at CVS", HBS number 9-606-015). As part of the assignment, students model the described process in YAWL, including all resource and data perspectives. This part of the assignment is assessed using the following criteria:

- Is the model valid and sound?
- Is the process well-structured?
- Are there unnecessary OR splits or joins?

- Are the major exceptional paths handled?
- Are process loops and iterations correctly modeled?
- Is net-level and task-level data defined using appropriate data types?
- Are net-level and task-level data mappings defined?
- Is net-level data used for flow-control for all XOR-splits?
- Are appropriate resources defined and assigned to each task?
- Does the workflow model load and execute on the YAWL engine?

Only when these formal criteria have been satisfied, do we examine the “correctness” of the model with respect to the case description.

4 Challenges

4.1 Student Preparedness and Motivation

The readings presented in the first five weeks focus on the genesis of the concept of process-orientation from the early 1990s to the mid-2000s. However, students often lack a basic understanding of fundamental concepts in organizational theory and organizational behavior. This requires instructors to delve relatively deeply into these concepts to give students a sufficient basis to comprehend the course readings.

Workflow management could be seen as an abstract concept on its own but when the potential business benefits are discussed the level of student interest rises. First- and second-year students are unlikely to immediately relate an improved business process to an increase in business value. However, the concept of a reduced wait in a line-up is clear to them. Expanding on practical examples of business processes and the impact of their improvements allows the instructors to slowly move towards the technology portion of the course while maintaining student interest.

Another course element to maintain relevance to students are weekly reflections (“journals”, “blog”) that focus on personal experiences with the material covered in that week. They encourage students to make sense of the material through relation to familiar situations, while at the same time demonstrating relevance.

4.2 Group Work

Because the benefits of workflow management can best be realized when multiple process participants access a common workflow system, the YAWL assignment was originally designed as a group assignment, so that each group can a process in a realistic shared environment. While this is beneficial to student understanding, it resulted in most student groups developing a single “YAWL expert”. This defeated our goal of all students having an understanding of the workflow management system. Thus, assignments are now done individually, whereas the lab work focuses on the cooperative group aspect. Another advantage of individual assignments is that they can now be used to satisfy the assurance of learning assessment required to maintain AACSB accreditation, which was not possible for group assignments.

4.3 The YAWL Data Perspective

Data management in YAWL is arguably the most difficult aspect of the system for general business students with no prior knowledge in computer science or programming. It is also the aspect that contributes least to the general understanding of business process and workflow management. However, modeling data in YAWL is required for the workflow to be realistic, in the sense that students are able to view and edit work items. Thus, omitting the data perspective is not a good option.

As a consequence, as seen in Table 1, we have to allocate significant time to teaching even simple XML schema development, and XPath expressions. Due to many hands-on exercises, the general feedback from students is relatively positive, despite the level of difficulty. However, we have found that it is important to discuss the YAWL data perspective with built-in data types before introducing XML schema, so that students know the purpose of the XML schema they are developing.

As a future alternative, we are considering providing suitable XML schema to students for their YAWL assignment, so that students can focus on the input/output mappings from net-level to task-level variables and the flow-control for XOR splits and less course time needs to be allocated to this topic.

4.4 Integrating YAWL with Process Mining

In earlier offerings of the course, we asked students to also execute their workflow 20 times for a follow-up assignment on process mining, to expose students to the full lifecycle of business process management. Students exported their execution log data using PromImport and mine the data using ProM. Technical and pedagogical issues forced us to abandon this. First, we experienced too many support issues with PromImport, where students were unsure how to install and use the program. Moreover, exporting data from YAWL with PromImport puts a considerable load on the YAWL engine, leading to frustration among students. We also found that the range of student's solution to the workflow modelling assignment made it difficult to provide a single relevant set of questions (and solutions) for the process mining assignment. We now omit this assignment and provide students with a hands-on introduction using the ProM tutorial developed by de Medeiros and Weijters [9].

4.5 Software Versions

When we developed the course in 2009 and 2010, the then current YAWL version 2.0.1 reflected the newly published YAWL book, making for an ideal combination. As we also integrated process mining into the course, we "froze" YAWL at version 2.0.1, to work with PromImport (version 7.0), and ProM (version 5.2). While the later decision to remove PromImport from the course provides flexibility in the use of different YAWL (and ProM) versions, the fit of 2.0.1 with the YAWL book is our main reason for remaining with this version. We hope that the book may be updated at some point in the future to again accurately reflect the latest system.

4.6 YAWL Hosting

The nature of WfMS means that the best learning experience is in groups of students, each representing its own organization. Ideally, this requires a separate YAWL system for each student group in the course. We explored the following options:

- Multiple servers host separate YAWL systems. As the YAWL system is not resource intensive for our application, this was not a preferred configuration.
- A single server runs multiple virtual machines, each with its own installation of the stock YAWL system. While the configuration of YAWL in this scenario is simple, the configuration of the VMs and dynamic provisioning of additional instances requires more administration resources than the faculty can provide.
- A single server and servlet container hosts multiple YAWL engines, databases, resource services, etc., each configured with its unique set of ports. While this is technically the most efficient solution, it requires considerable administrative expertise, especially for automatic provisioning of additional YAWL instances.
- A single YAWL instance, with the use of naming conventions for the organizational model for different student groups. This is administratively the easiest solution, but requires some effort from students.

Resource constraints meant that it was impossible to internally host a single (or multiple) YAWL systems and that we could not develop an experienced YAWL system administrator. Hence, the task of hosting any system falls to the instructor. To avoid configuration issues, a stock version of YAWL4Study is hosted on an Amazon EC2 micro-instance, which is easily sufficient for dozens of concurrent users.

5 Conclusions and Recommendations

In conclusion, we have successfully introduced the YAWL system and textbook into an undergraduate, required course for general business students. As this paper has demonstrated, the approach taken is different from one for a IS majors that have a technical understanding and interest in the topic. In light of the challenges we have encountered, we offer the following recommendations:

- Early demonstration of business use and benefits to achieve student "buy-in" into technology aspects of business process management.
- Demonstrate the importance and relevance of business processes by not focusing on technological aspects, but through managerial considerations.
- Constant focus on business applicability and relevance, through
  - Case studies
  - Personal reflections
  - Appropriate process examples

In summary, developing the course proved to be more of a challenge than had the course been offered as an IS elective to interested majors. The fact that the YAWL book, as most other books, does not present the topic in the larger business context requires a large amount of extra material. For this, we have mainly drawn on Harvard Business Review articles and Harvard business cases. Thus, there is a need for future pedagogical material on YAWL, and workflow management in general, to extend the scope. The newly published book by Dumas et al. [18] takes some steps in this direction and may be considered for a future course revision.

We believe that we have achieved a successful balance of managerial and technical issues that successfully introduces students to the area of business process and workflow management. All course materials, including slides, reading notes, lecture plans, student handouts, exercises and assignments are available from the authors in the hope that they may be useful to others.

6 References

1. Van der Aalst, W., van Hee, K.: Workflow Management. MIT Press (2004)
2. Van der Aalst, W., Stahl, C.: Modeling Business Processes. MIT Press (2011)
3. Weske, M.: Business Process Management. Springer, Heidelberg (2007).
4. Magal, S.R., Word, J.: Essentials of Business Processes and Information Systems. John Wiley & Sons, Hoboken (2009)
5. Magal, S.R., Word, J.: Integrated Business Processes with ERP Systems. John Wiley & Sons, Hoboken (2011)
6. Kroenke, D.M., McKinney, E.H.: Processes, Systems, and Information. Pearson, Boston, MA (2011)
7. Ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russel, N.: Modern Business Process Automation. Springer, Heidelberg (2010)
8. Kolb, D.: Experiential learning. Prentice-Hall, Englewood Cliffs, NJ (1984)
9. De Medeiros, A.K.A., Weijters, A.J.M.M.: ProM Framework Tutorial. http://tmpmining.win.tue.nl/_media/tutorial/promtutorialv2.pdf
10. Hammer, M.: Reengineering work: don’t automate, obliterate. Harvard Business Review 68:4, 102-114 (1990)
11. Hammer, M., Stanton, S.: How process enterprises really work. Harvard Business Review 77:6, 108-118 (1999)
12. Champy, J.: People and Processes. ACM Queue 4:2, 34-38 (2006)
13. Hammer, M.: The process audit. Harvard Business Review 85:4, 111-123 (2007)
14. Miers, D.: Best practice BPMN. ACM Queue 4:2, 40-48 (2006)
15. Reisig, W., Rozenberg, G.: Information introduction to Petri nets. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets. LNCS, vol 1491. Springer, Heidelberg (1992)
16. Bratosin, C.: A brief guide to XML, XML Schema, XQuery for YAWL data perspective. http://www.win.tue.nl/~cbratosi/yawl/YAWL-XML.pdf
17. Treese, W.: What’s all the noise about XML? ACM NetWorker 2:5, 27-29 (1998)
18. Dumas, M., La Rosa, M., Mendling, J. and Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2013)

7 Appendix

	Topic	Readings / Materials
1	Introduction to business process and workflow management	
2	Introduction to the YAWL workflow management system	YAWL book preface (pp. v-vii)
3	Business processes and process-oriented organizations	Hammer (1990) [10]
4	Organizational change and change management for implementing business process	Hammer and Stanton (1999) [11] Champy (2006) [12]
5	Process and Enterprise Maturity Model	Hammer (2007) [13]
6	Process and Enterprise Maturity Model	Harvard Business School Case "Siemens Rolm Communications" (HBS# 195214-PDF-ENG)
7	Business process metrics, analysis and process improvement	Harvard Business School Case "Body Scans and Bottlenecks" (HBS# KEL592-PDF-ENG)
8	Business process metrics, analysis and process improvement	
9	Business process management lifecycle	Miers (2006) [14]
10	Workflow management - YAWL overview and architecture	YAWL book chapter 1 YAWL book chapter 7
11	Petri nets	Reisig and Rozenberg (1992) [15]
12	Workflow nets	YAWL book section 2.3
13	YAWL Language and Design Environment – Control flow perspective	YAWL book Section 2.1 YAWL book Section 2.2.1 YAWL book Section 2.4 YAWL book Chapter 3 YAWL book Section 8.1 YAWL book Section 8.2 YAWL book Section 8.5
14	YAWL Language and Design Environment – Control flow perspective **(computer lab)**	
15	YAWL Language and Design Environment – The resource perspective	YAWL book Section 2.1 YAWL book Section 2.2.2 YAWL book Section 2.5 YAWL book Section 8.2 YAWL book Section 8.3
16	YAWL Language and Design Environment – Resource perspective **(computer lab)**	
17	YAWL Language and Design Environment – Data perspective **(computer lab)**	YAWL book Section 2.1 YAWL book Section 2.2.3 YAWL book Section 2.6
18	Web services, XML, and XML Schema	

	Topic	**Readings / Materials**
19	YAWL Language and Design Environment – Data perspective **(computer lab)**	YAWL book Chapter 10 YAWL book Section 8.4 Treese (1998) [16] Bratosin (2009) [17]
20	Process Mining using ProM **(computer lab)**	YAWL book Chapter 17
21	Other modelling languages – BPMN	YAWL book Chapter 13
22	Other modelling languages – EPC	YAWL book Chapter 14
23	Other modelling languages - BPEL	YAWL book Chapter 15
24	Business process examples – Logistics	YAWL book Appendix A

Table 1. Course schedule and reading list for the business process management course for the Winter 2013 semester

Developing a Realistic Workflow Management Environment for Teaching: An Interface from YAWL to OpenERP

Joerg Evermann

Faculty of Business Administration, Memorial University of Newfoundland, Canada
jevermann@mun.ca

Abstract. The paper describes an interface from the YAWL workflow management system to the OpenERP enterprise system. The interface is implemented as a codelet, and provides access to the full range of OpenERP information and functions. The paper provides an overview of the design of the codelet, the data types for its use, and an example application.

Keywords. YAWL, OpenERP, workflow management, enterprise system

1 Introduction

The YAWL (Yet Another Workflow Language) workflow management system is developed for teaching and research. Together with the YAWL book [1] it provides a state-of-the-art open-source environment for teaching business process management and automation, in an affordable and easily accessible way. The YAWL system is currently used in an introductory, core course on Business Process Management (BPM) in the general business degree curriculum at Memorial University. While this course covers all aspects of BPM, strategic, managerial, operational, and technical, YAWL plays a prominent part in the course to illustrate the capabilities and benefits, but also the complexities of process automation.

In the fall of 2011, Memorial University funded, through a $5000 Instructional Development Grant, a course improvement project aimed at introducing hands-on ERP systems experience into the business undergraduate curriculum. The initial targets of this initiative were three courses: Information Systems, Operations Management, and Business Process Management course. From the BPM course perspective, integrating the workflow management with ''legacy'' information systems provides for a more realistic environment for students to experience and learn about workflow management and process automation.

While the business benefits of integrating workflows and enterprise systems are easy to describe, they are difficult for students, especially those at an early stage in the degree program, to fully appreciate without hands-on experience. Experiential learning, defined as "the sense-making process of active engagement between the inner world of the person and the outer world of the environment" [2, pg. 2], is a "more effective and long-lasting form of learning" that "involves the learner by creating a meaningful learning experience," [2, pg. 1] and "learning from experience is one of the most fundamental and natural means of learning available," [2, pg. 15].

The project's intended learning outcomes are an improved understanding and appreciation of the capabilities and importance of integrated enterprise IT to business operations.

Given the size of the development grant, and the inability of the faculty to provide additional monetary or human resources to this project, the scale of the project was quite limited. The chosen enterprise system was OpenERP[1], an open-source system that provides core modules such as sales, purchasing, accounting, production management, as well as extensions for point-of-sales, project management, etc. Among open-source ERP systems, it is one of the most mature and feature complete systems. OpenERP provides its own process model and workflow engine. However, the configuration language is XML based and there is no recognizable formal underpinning for the workflow description language.

This suggested the development of an interface from YAWL to OpenERP, so that OpenERP functionality can be used in a YAWL workflow. The remainder of the paper describes the implementation of this interface as a YAWL codelet. The next section provides an introduction to the OpenERP system, followed by design choices for the interface codelet. This is followed by a description of the codelet parameters and data types, and an example workflow.

The codelet, associated XML Schema data type definitions, and YAWL example workflow definitions are available under the GPL v2 license[2].

2 The OpenERP System from the YAWL perspective

OpenERP is developed using the Python language and provides a business object model that abstracts, through an object-relational mapping layer, from the underlying physical data structures and functions. On top of this object model, OpenERP defines a process model and workflow mechanism for many of the business objects. The OpenERP workflow model is based on object states and transitions between them. Each object *state* is associated with a method, whereas transitions are either triggered by signals, or triggered by changes in attribute values.

OpenERP provides an XML-RPC based web-services interface to both its business objects and its workflow mechanism. This interface provides the following generic operations on all business objects:

- Create (returns the new business object ID)
- Search (returns a set of business object IDs that match a query)
- Read (returns a set of attribute values for a given list of business object IDs and a given list of attribute names)

[1] http://www.openerp.com
[2] http://www.yawlfoundation.org/pages/resources/contributions.html

- Write (updates the provided attributes of business objects with a given list of IDs with the provided new value)
- Delete ("Unlink")

OpenERP also provides a means to call any method defined on the business objects. However, calling the methods directly (outside the built-in, intended workflow) may lead to issues such as the (implicit) pre-conditions (as per the built-in workflow) not being met, or the consequent actions (as per the built-in workflow) not being executed. Thus, calling the business object methods directly is not recommended. Instead, OpenERP provides a mechanism to send "signals" to its workflows. These signals can be used to advance the built-in workflow for a business object. For example, a signal may be sent to confirm a draft sales quotation and transform it to a sales order. The OpenERP workflow mechanism then calls the appropriate methods on the business object.

While this is a "safe" mechanism to interact with the OpenERP system, the externally defined and controlled workflow (e.g. from YAWL) must be essentially isomorphic to the workflow configured in OpenERP and can only 'mimic' that workflow. Further, developing an external workflow requires a thorough understanding of the built-in workflow and the states of the business objects. However, the codelet is not limited to this "safe" way of interacting with OpenERP, and it can in principle be used to call any method in any order. This, however, requires a thorough understanding of the OpenERP pre-conditions and consequent actions.

3 Interface Design

YAWL provides different ways to integrate external systems, the three most prominent being the web-services invoker service, the codelet mechanism, and the external data gateway. Any of these can in principle be used to develop the interface.

As the YAWL web-service invoker service requires a valid WSDL file and uses SOAP rather than XML-RPC, this would have required a translation server that accepts SOAP requests and issues XML-RPC requests in turn. Alternatively, a new XML-RPC web-service invoker service could have been built as an alternative to the existing SOAP based one. Either of these alternatives was considered to be too technically complex for the limited time-frame of the project.

An external data gateway could be constructed to access either the business object information in OpenERP, or directly access the underlying relational data. Technically, one could also imagine that this might be used to access methods or send workflow signals, but this would be conceptually confusing, as the data gateway is intended primarily for data access.

Instead, a simple codelet was developed that accepts input and provides output using pre-specified data types. Two options were investigated:

- Offer access to specific OpenERP business objects, their data, methods, and workflow signals. In this scenario, XML data types would need to be developed that reflect the OpenERP business object model, e.g. a data type for the "sales order" object, a data type for “sales order line” object, etc. This would remove the burden of data type development from the YAWL process designer, but would at the same time limit the flexibility of the codelet to a fixed set of business objects, their data and methods as determined by the codelet design.

- Offer access to generic as well as specific OpenERP operations with no abstraction from the OpenERP XML-RPC API. This requires the codelet user, i.e. the YAWL process designer, to develop appropriate business object data types and deal with the specifics of data transformation to the OpenERP API on the YAWL side, e.g. as part of the input and output mappings for tasks. The benefit is that the codelet does not prescribe specific data types, and it can access any OpenERP business object or workflow. The codelet was developed based on this, second model.

The codelet itself is stateless and establishes a new connection to the OpenERP system for every call, thus requiring the OpenERP connection information with every call. While this may not be as efficient as returning a connection handle to the YAWL workflow, the fact that the YAWL workflow may be long-running means that a connection handle in the YAWL workflow data might expire. Further, as tasks in the YAWL workflow may be assigned to different (human) resources, maintaining a quasi-persistent connection handle would also force the same OpenERP user account for the entire workflow, which may not be desirable in practice.

4 Use and Example

Table 1 lists the input parameters for the codelet. The content of the `method` parameter is limited to the five generic methods for OpenERP business objects: `search`, `write`, `read`, `delete`, `create`, and the additional `action`, which is used for sending signals to OpenERP workflows. The results that the codelet returns depend on the invoked method. Alternatively, the codelet returns an error, either passed back from OpenERP, or an exception in the codelet, or an error encountered by the codelet, e.g. when the input paramters do not match the invoked method type.

Parameter	**Type**	**Description**
`URL`	`xsd:String`	Hostname for OpenERP
`Port`	`xsd:Integer`	Network port for OpenERP
`Database`	`xsd:String`	OpenERP database to select
`Username`	`xsd:String`	Username for OpenERP
`Password`	`xsd:String`	Password for OpenERP
`Object`	`xsd:String`	Type of OpenERP business object on which method or action is to be called
`Method`	`xsd:String`	OpenERP method name
`Parameters`	`ParameterType`	Parameters appropriate for the called method

Table 1. OpenERP codelet input parameters

Figure 2 below shows the YAWL workflow for creating and processing sales order. That workflow is based on managing sales orders in OpenERP, shown in Figure 1 above. The codelet design decisions have an impact on the usage of the codelet in two important ways. First, the direct representation of the basic OpenERP method calls (search, read, etc.) leads to typical combinations of search-read sequences as two automated tasks in the YAWL workflow. Second, the relatively low degree of abstraction of the codelet parameters suggests that the data transformations in the YAWL task input-/output-mappings are not trivial. Thus, the use of the codelet requires considerable XQuery expertise.

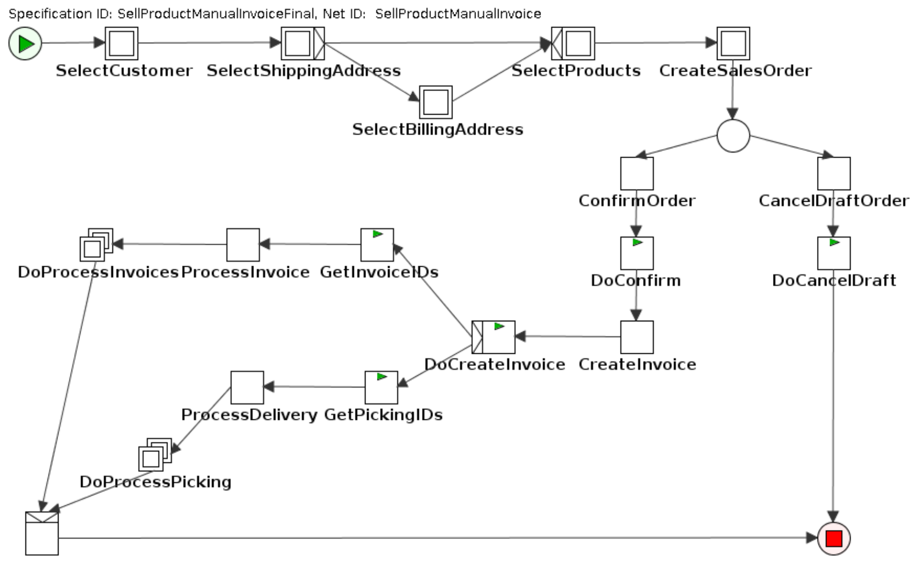


Fig. 2. Sales order process

5 Use in Teaching and Current Status

When the project was initiated, the intended use of the YAWL-OpenERP interface was to allow students to create realistic workflow definitions for simple processes like sales order processing, as part of an assignment or course project. It was hoped that by using a realistic integration with business data in the ERP system, the usefulness of workflow management could be demonstrated to students and lead to better appreciation and understanding of the business value of process automation. Specifically, the experiential learning is argued to:

- increase student engagement,
- improve student skill development,
- increase student learning and understanding.

As the codelet implementation is now complete and an example process (Figure 2) is implemented, we have found that the level of YAWL, OpenERP, and XML knowledge required to develop integrated workflows is beyond what can be taught in an introductory course that has no computer science or programming pre-requisites.

There are a number of possible responses to this situation. First, we are aiming to have students interact with the process, rather than create their own workflows. For the coming fall semester, we are working to make the defined workflow more robust to user error and to provide a better user interface. At the same time we are developing computer lab exercises for students to use the pre-defined workflow. Finally, we are developing workflows for processes other than sales order management.

A second possible response is to investigate the aggregation of lower-level tasks that access the codelet into higher-level workflow fragments, whose use does not require knowledge of OpenERP, XML, or the codelet design. One option for this is the use of worklets which can then be assembled by students. However, the current course design does not include worklets or declarative workflow design. Another alternative is to provide students with a workflow specification that contains a number of YAWL nets, which can either be copied-and-pasted into students' own work, or to which composite tasks that students create as part of their own workflows can be unfolded. The current example process (Figure 2) already contains many such abstractions as composite tasks. Again, a set of computer lab exercises for students will be developed around this idea in time for the fall semester.

5.1 Evaluation

The intended learning outcomes will be evaluated using questionnaires measuring student engagement, adapted from [3, 4], perceived skill development, adapted from [5]. Further, questions on student's understanding of workflow management principles and the role and capabilities of YAWL in process automation will be used:

- In your own words, describe what the YAWL system is. (Q1)
- In your own words, describe the place of the YAWL system in a company. (Q2)
- In your own words, describe how the YAWL system relates to other information systems in a company.(Q3)
- In your own words, describe why and how the YAWL system can be useful to a company. (Q4)

Additionally, Likert-type scales will be used for the following questions:

- I have a good understanding of workflow management (Q5a)
- I am able to explain workflow management to other students (Q5b)
- I am able to use workflow management systems (Q5c)
- I am able to make a business case for workflow management. (Q5d)

. These questionnaires will be administered using a pre- and post-test design before and after a computer lab class that involves the OpenERP interface. This allows us to measure the changes in students introduced by the realistic workflow environment. Additionally, on the post-test questionnaire, students motivation was assessed, using the following Likert-type scales:

- I would discuss related topics outside the class. (Q6a)

- I would do additional reading on related topics. (Q6b)
- I would do some thinking for myself about related issues. (Q6c)

5.2 Initial Results

To satisfy the requirements of the instructional development grant under which the project was funded, a preliminary evaluation of benefits had to be assessed, even though the OpenERP interface was not yet fully integrated into the pedagogics of the course. To meet the deadline, the integration between WfMS and ERP was demonstrated by the course instructor using the sales order management process in Figure 2. Students were shown the workflow definition, the OpenERP data, and the running workflow. As expected, a demonstration is not as engaging or interesting as experiential learning, and the initial results reflected this.

From a total of 77 students, 57 responses were received, of which 53 provided information on both the before and after questionnaire.

The understanding questions (Q5a-Q5d) were averaged as they all represent understanding of WfMS. There was no significant difference between the means for the before and after questionnaire (before = 3.80, after = 3.97, on a 7-point scale).

The motivation questions (Q6a-Q6c) were used only on the post-test questionnaire. The results indicate moderate motivation levels (approx. scale mid-point) for Q6a and Q6b, whereas Q6c shows good motivation levels. The difference is not surprising, as the Q6a and Q6b asked students whether they would take some action, whereas Q6c only asked whether they would "think about" the topic.

Finally, we examined the engagement [3, 4], perceived skill development [5] and perceived usefulness (single item). The descriptive results are shown in Table 2.

	Mean	**SD**
Perceived Engagement	3.33	1.39
Perceived Skill-Development	4.04	1.29
Perceived Usefulness	4.26	1.58

Table 2. Results for engagement and skill development, on 7-point Likert scales

The results indicate that the demonstration was not engaging to students (mean less than scale mid-point, but not significant as per t-test). However, the demonstration was perceived as improving skill development (mean significantly above scale mid-point, $p<0.01$) and useful (mean significantly above scale mid-point, $p<0.01$). The result with respect to engagement is not surprising as the demonstration required students to watch for 15 minutes rather than interacting with the system themselves in a true experiential way, as originally intended. The results with respect to skill development and usefulness are encouraging, especially given the low level of student engagement. We believe that this can be significantly increased once true experiential interaction with the system is available.

Only 18 responses were received with answers for Q1-Q4 differing between the before-demonstration and after-demonstration questionnaire. The answers were examined to identify improvements in understanding. Of the 18 respondents, only 12 showed improvements in understanding and even fewer showed a marked improvement across all four questions. The low response rate is likely due to the low level of engagement..

6 Discussion and Conclusion

This paper presented a YAWL codelet to access the OpenERP system. The codelet exposes low-level access functionality, rather than business-level objects or methods. This low level of abstraction requires the codelet user to have a thorough understanding of the OpenERP data model, methods and workflows. At the same time, this design makes the codelet useful for the widest range of applications. Codelet users and workflow designers may also use YAWL features to build additional layers of abstraction on top of this foundation. For example, YAWL worklets could be defined that aggregate some of the codelet functions, e.g. the search-read combinations, into assemblies that are meaningful at the business level.

Designed as part of a project to introduce experiential learning into an undergraduate business degree, the codelet is essentially feature-complete and robust, but the development of teaching material, example workflows, and associated documentation is still in progress. An evaluation of the usefulness of the OpenERP interface for teaching workflow management is planned, based on goals such as increased understanding, increased student engagement, and increased skill development. An evaluation that was not based on experiential learning has only shown limited improvements on these dimensions, but this is expected to improve once the interface is ready for its intended use in the classroom and computer lab tutorials.

While this project focused on the OpenERP system, we believe that the challenges we faced, and the design decisions we made, e.g. w.r.t. having to “mimic” the internal workflow or the choice between providing abstraction or a direct interface, are not unique to OpenERP and applicable to other open-source or commercial ERP systems.

7 Bibliography

1. Ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russel, N.: Modern Business Process Automation. Springer, Heidelberg (2010)
2. Beard, C.M., Wilson, J.P. (2006) Experiential Learning: A Best Practice Handbook for Educators and Trainers (2nd. Ed.). Kogan Page Limited, Philadelphia, PA.
3. Webster, J. and Ahuja, J.S. (2006) “Enhancing the design of web-navigation systems: The influence of user disorientation on engagement and performance,” MIS Quarterly, (30:3), pp. 661-678.
4. Webster, J. and Ho, H. (1997) “Audience Engagement in Multimedia Presentations,” The DATABASE on Advances in Information Systems, (28:2), pp. 63-77.
5. Alavi, M. (1994) "Computer-Mediated Collaborative Learning: An Empirical Evaluation," MIS Quarterly, (June), pp. 159-174.

Adapting a Generic Data Synchronisation Framework for YAWL to Access Clinical Information Systems at the Task Level

Holger Meyer[1], Sebastian Schick[1], Jan-Christian Kuhr[2], Andreas Heuer[1]

[1] University of Rostock {hme,schick,ah}@informatik.uni-rostock.de,
[2] GECKO mbH Rostock jan-christian.kuhr@gecko.de

Abstract. Based on a generic extensible data access framework (DAF) [10], we devised a domain-specific extension to support interoperability of YAWL workflow cases with hospital information systems (HIS). In the scenario considered, the HIS is the principal system that keeps master data and clinical data of the patient. Data transfer between HIS and the YAWL runtime environment is facilitated via exchange of standardized HL7 messages. The solution presented supports read and write task-level synchronization of workflow variables using the patient's case ID as correlation parameter. A first proof-of-concept has been carried out in a real clinical setting.

Keywords: Workflow, Data Access, YAWL, Healthcare, Perioperative Process

1 Introduction

Clinical value chains are in general well-structured and follow an a-priori known execution path. Thus, such scenarios lend themselves to be supported by process aware information systems in general and by business process management systems (BPMS) in particular.

When a BPMS approach is adopted, particular challenges arise if the process engine is external to existing hospital information systems. Patient's master data and visit-related clinical data such as diagnosis and scheduled treatments are usually handled within a HIS, which is the principal system. However, a subset of data is required to be known by the process execution engine to support execution of patient's workflow cases. Conversely, it may be necessary to map process runtime data, such as the duration of activities, back to the HIS. These scenarios call for a data synchronisation solution that integrates HIS with the process execution environment.

In the PERIKLES project we have introduced an extendable generic data synchronisation framework (DAF) [10] and then specifically adopted to the domain of clinical healthcare. The framework helps to avoid inconsistencies within redundantly maintained data and supports transactional aspects within the process and data perspective. Therefore, a layered architecture is used with facilities to access external data sources, to associate the control-flow perspective with transactional properties like isolation, serializability and recovery.

A second extension to YAWL is the *FlexY* approach [11, 12], which extends the control-flow perspective of YAWL with new concepts for handling process adaptation at run-time. *FlexY* combines the method of late modeling with declarative concepts and underspecification. The runtime adjustment of sub-processes is triggered at certain points of the process according to external data provided by the data access framework. Why flexibility is important and how it can deployed profitably in healthcare is described elsewhere [9, 2].

In this paper we focus on the DAF approach to extend YAWL to handle task-level read and write access to hospital information systems by exchanging standardized messages. The extension described in this paper makes use of YAWL concepts like workflow engine level extensibility (gateway mechanism), automated tasks provided as services, tasks and sub-net variables and parameters based on an XML type system, flexible enactment of sub-processes and exception handling.

2 A Generic Data Synchronisation Framework

The current version of the YAWL engine already provides a simple interface to populate task and net variables using a data source called *data gateway*. In case of using a data source some implementation effort is necessary, because YAWL provides no standard gateways. To provide a configurable, policy based integration of external data sources, we presented a framework called *D*ata *A*ccess *F*ramework in [10]. First, we give a short overview of the main framework components. Afterwards, we present the extensions of the framework which were developed.

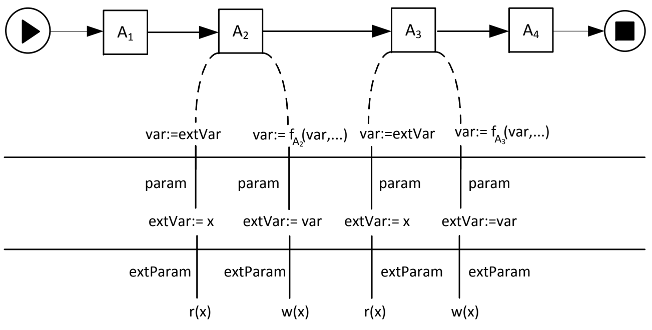


Fig. 1. Data access on external data sources in YAWL

External Variables Access to data in external sources should be transparent to the user who models and executes a YAWL process model. Therefor, we introduce a new variable type *external variable* in YAWL [10]. An external variable defines a view on a data source by specifying the data source and required configuration parameters. Additionally, each data source is encapsulated by a plug-in implementation. The plug-in integration is outlined in the following section.

Figure 1 shows the main idea behind the external variable concept. Each variable (c.f. Fig. 1: *var*) in a process model, can be bound to an external variable (c.f. Fig. 1: $extVar$), using a parameter mapping (c.f. Fig. 1: $param$). To bind an external variable (c.f. Fig. 1: $extVar$) to an plug-in, we use a new type of parameter called external mapping (c.f. Fig. 1: $extParam$). For each net we can define a set of external variables with the type Ext, a set of plug-in id's $PlugInID$ and a set of external parameter mappings $extParamID$.

Definition 1 (External parameter mapping). *A mapping is defined as* $(pID; distKey; map; rp; wp)$*, with:*

- *pID references a plug-in which should be used,*
- *distKey is the key attribute to identify a single XML fragment,*
- *map is an XQuery query which defines the transformation rule between an external variable and the respective data source (with the ID pID),*
- *rp is a local read policy (consistent, read-only) for the external variable and*
- *wp is a local write policy (consistent, immediate) for the external variable.*

In Def. 1 an external parameter mapping is defined, where pID is used to identify a plug-in. The $disKey$ and map parameters describe a data item in the source and the rp and wp parameters are used to define the access policies.

Framework Architecture To support external variables within YAWL, we extend the existing YAWL data gateway implementation with our own *data access gateway* [10]. However, in this scenario we use a simplified version which do not use extended transactional services. We use a data gateway which encapsulates the data access framework and manages the access to different plug-in implementations. A Data Source Manager configures further processing using the variable mapping of the external variable and selects the plug-in which have to be invoked.

3 Adaptation to Clinical Information Systems

In healthcare, interoperability between clinical information systems is generally accomplished via exchange of standardized HL7 messages. HL7 messages are made up of segments, fields and delimiters. The *MSH* segment in the example (cf. Fig. 2) indicates that the message is of type ADT (admission, discharge, transfer) and has been caused by an *A01* (patient admission) trigger event.

3.1 HL7 Plug-in

Supporting clinical healthcare workflows by YAWL requires integration of given hospital information systems with the process execution runtime. Following our

```
MSH|^~\&|SAP-ISH|ROD|ITB265||20101123025931|NP11I0|ADT^A01|0000688151|P|2.1
EVN|A01|20101123025000
PID|||265029061|265051944|Nachname001^Vorname001|Mädchenname001|19350128|F|||Musterstr1.
001^^Musterort001^^66976^DE||00000 SELBST||D|W|02
PV1|00001|I|26513.2^^^2651IA|NO||||||1A|||||||||||NO|||||||||||||||R|2651964931||||20101123025000
```

Fig. 2. Example of an HL7 message

general approach, we have thus extended the existing DAF by an HL7-plug-in to support read/write access of YAWL processes to HL7 enabled clinical systems. The basic architecture of our approach is shown in Fig. 3.

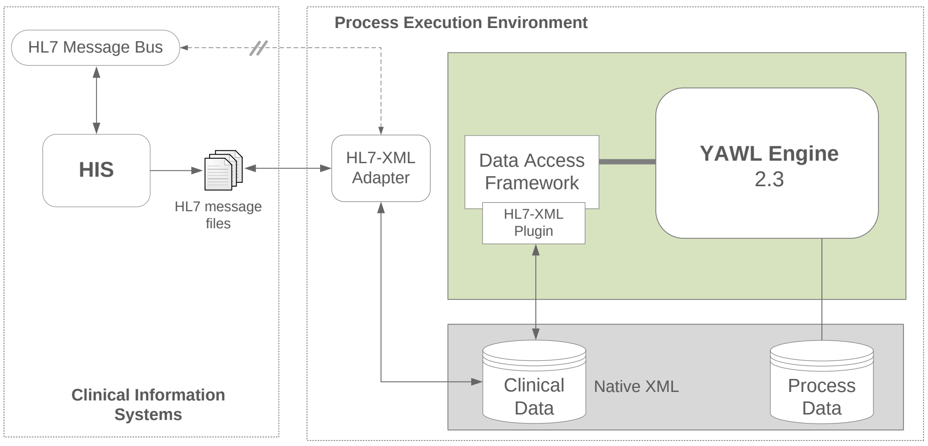


Fig. 3. General architecture of the HIS - YAWL integration

Extending the Process Specification By utilizing extended attributes one can specify at the task level data synchronization policies for individual variables. These policies instruct the DAF about the mode of the data access (read or write), the correlatable parameter to be used (e.g. the patient's medical case id), the plug-in that handles the request, and an XPath expression that extracts the relevant data from the XML representation of the HL7 message. By calling the plug-in the data synchronisation with HL7 sources is handled for all *external* variables.

Listing 1 shows an example of a definition of an external variable. Besides a plug-in ID (`pluginID=XML_PostgreSQL_ADT_Extract`), the source description is defined by an SQL table name (`xmlColumn=patientdata`) together with key attributes (`mappingKey=medcaseid,msgtype,datetime`) and the selection of an XML fragment (`mapping=//patientdata/pv1/patientclass`). We also define some initial values for two of the key attributes (`mappingKeyVal=,A01,[max]`) and a table for write access (`writeProcess=createHl7[public.hl7out;m1]`).

To identify HL7 messages in the database, a number of different attributes are necessary, which are used as key attributes. Some of the key attributes are known at design time, like *message type* or *aggregate functions* over an attribute. Other attributes are only known at runtime, like *medical case ID*. These compound primary keys are described by the *mappingKey* attribute.

Listing 1. External variable definition

```
extVar_pluginID=XML_PostgreSQL_ADT_Extract
extVar_readPolicy=readOnly
extVar_xmlColumn=patientdata
extVar_mapping=//patientdata/pv1/patientclass
extVar_mappingKey=medcaseid , msgtype , datetime
extVar_mappingKeyVal=,A01 , [max]
extVar_writeProcess=createHl7 [public . hl7out ;m1]
```

In most cases HIS assume the role of a data source. Upon the occurrence of predefined events the HIS emits HL7 messages that travel along a message bus or may be exported to the file system. In order to validate our approach in a real clinical setting without running the risk of interfering with production systems, we have used the latter method (cf. Fig. 3).

Since YAWL handles task and case variables as XML data types, every *incoming* HL7 message needs to be transformed into an XML representation that is compliant to the XML schema of the YAWL process specification. Conversely, all *outgoing* messages must be converted from XML to a proper HL7 representation. Therefore, depending on whether variables are specified for read or write access, we use different tables for read and write access. We therefore extend the source definition of an external variable, such that a second source for write access can be defined (cf. Lst. 1: `extVar_writeProcess`). These bi-directional message transformations are accomplished by an HL7-XML adapter (cf. Fig. 3). In read mode, the adapter polls the HL7 export file for new messages and transforms any of these into an XML representation. In order to avoid excessive transformations each time a message is accessed, all XML-formatted messages are stored persistently in a database (cf. Fig. 3: Database HL7Messages).

3.2 Proof of Concept

The validity of our approach for real clinical settings has been demonstrated by a single-day proof of concept (PoC) deployment in a German hospital. Figure 4 shows the principal design. While the PoC has been restricted to *read-only* synchronization, the figure shows also *write-mode* use cases that may be relevant for future evaluations. The underlying clinical scenario is a perioperative careflow, which, for the sake of clarity, has been greatly simplified.

From the perspective of the **treatment layer**, each patient follows a sequence of events, beginning with the admission to the hospital end ending with the post-operative transport to the intensive care unit (ICU). While this sequence is strongly aligned with the process layer, there are also important interactions with the hospital information system.

The **HIS layer** may be viewed as consisting of two distinct components: The patient management system (PMS) keeps the patient's master data such as name, address, gender, and date of birth as well as the visit-related *administrative* case data. For every visit a patient may make to the hospital, the PMS assigns a medical case ID to this visit. On the other hand, the clinical information system (CIS) is responsible for recording and managing the *medical* visit-related data, such as diagnosis, lab test results, and the planned treatment. Both subsystems are capable of emitting and receiving HL7 messages in order to communicate business events to other information systems or to process such events that have been emitted by other systems. HL7 messages may contain administrative data only (e.g. in case of an admission or transfer event) or primarily medical information (e.g. in case of communicating lab results).

The **process layer** has been implemented by the YAWL runtime environment and is thus external to the hospital information systems. The use case evaluated in the PoC required a read-mode synchronization of the workflow instance with the patient's master data as well as the visit-related administrative data, both of which are kept inside the PMS. Upon start of a workflow case, the patient's medical case ID is passed to the process as a parameter. The medical case ID, which is to be distinguished from the workflow engine's case ID, serves as the principal correlation link between the process layer and the clinical information systems. Once the task *Supply Master Data and Case Admin Data* becomes enabled, synchronization with the HIS layer takes place in read mode such that the required administrative data are extracted from an appropriate HL7 message that has been emitted by the PMS. This information may then be reviewed by the clinician that is executing the task before continuing with *Do Surgical Assessment*. In this way, synchronization relieves the stakeholders of re-entering the same data twice.

Possible Future Use Cases Data synchronization between process and HIS layer is by no means restricted to the read-only mode. Our approach is also capable of supporting write-mode scenarios that may be interesting from the clinical point of view and that lend themselves for future evaluation. In the figure, we give two motivating examples. In both of which, the workflow case acts as information source and a hospital information system as recipient of that information. Compared with the former scenario, the situation is thus reversed.

The workflow tasks *Transfer To OR* and *Transfer To ICU* correspond to events that are relevant from an *administrative* point of view. So rather than capturing this information twice, it may be desirable to run a write-mode synchronization on completion of these tasks, respectively, to communicate the time stamp of the event to the PMS. In terms of the HL7 messaging concept, this would also amount to automatically executing a *transfer patient* request inside the patient management system.

The task *Do Surgical Procedure* may have captured *clinical* data such as duration and description of the surgical procedure. However, this information may also be required by the CIS. Hence, one could think of enabling the workflow system to write this information back to a HIS layer by sending an appropriate HL7 message to be consumed by the clinical information system.

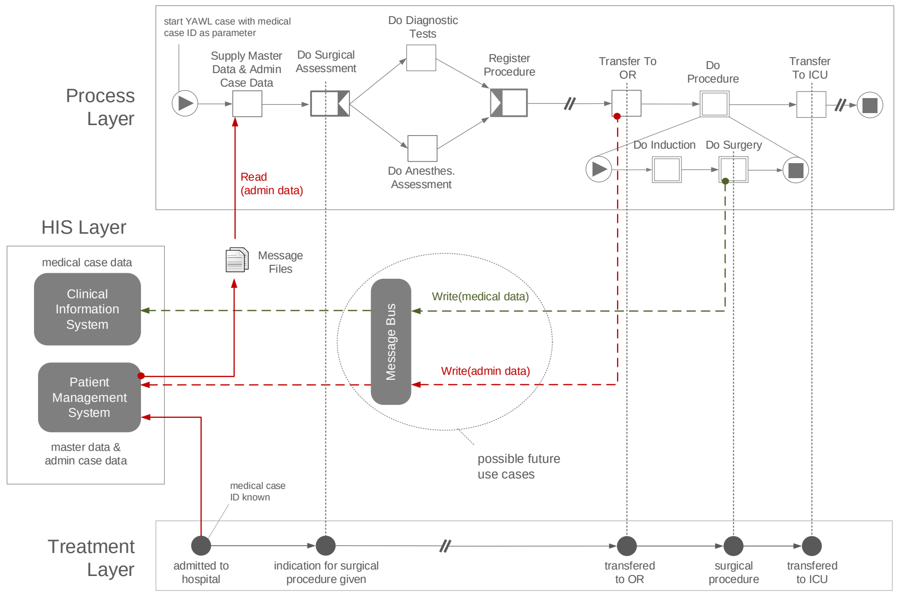


Fig. 4. Simplified clinical scenario of the proof of concept. The figure shows a use case that has been evaluated as in a real clinical setting (solid red line) as well as possible future uses cases (dashed red and green lines).

4 Related Work

YAWL is used and extended by several research initiatives including but not limited to clinical workflows. The framework supports the data patterns defined by Russel et al. [8]. The *Proclet* approach [5] extends YAWL to use process fragments, which can communicating via channels and ports. The support of different types of flexibility was demonstrated in several frameworks. The *Worklet* approach [1] extends YAWL with a concept of late-binding process fragments. In [3] an extension to YAWL implementing configurable process models is presented.

In the context of data-driven processes, different approaches for data integration exist. *SIMPLE* [7] describes strategies for accessing external data sources, by providing an abstraction layer for data management. Frameworks like *PHILharmonicFlows* [4] or *Corepro* [6] provide an extensive integration of object behavior, object interactions and process execution within the process model.

5 Conclusion

In this paper, we present an adaptation to a data synchronisation framework for YAWL and its application to a clinical information system. We showed that an efficient way to access data from different clinical information systems during runtime is necessary. We then adapted an existing data synchronisation framework to support accessing these data at the task level. In future, we intend to use YAWL in different scenarios supporting also flexibility.

Acknowledgments Part of this work has been funded by the German Federal Ministry of Education and Research (BMBF) under grant 01IS099009. We thank Markus Bandt and Viktor Baidinger for their contribution to the implementation. Validation of the key concepts has been done in cooperation with the Hetzelstift hospital in Neustadt/Weinstraße, Germany. The authors wish to thank especially Dierk Vagts and Bernd Wössner for supporting the proof-of-concept in a real clinical setting.

References

1. M. Adams, A. ter Hofstede, D. Edmond, and W. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Conferences*, pages 291–308, 2006.
2. M. Bandt, R. Kühn, S. Schick, and H. Meyer. Beyond flexibility — workflows in the perioperative sector of the healthcare domain. *Electronic Communications of the EASST*, 37:146–157, 2011.
3. F. Gottschalk, W. van der Aalst, M. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
4. V. Künzle and M. Reichert. Philharmonicflows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.
5. R. S. Mans, N. Russell, W. van der Aalst, P. Bakker, A. Moleman, and M. Jaspers. Proclets in healthcare. *Journal of Biomedical Informatics*, 43(4):632–649, 2010.
6. D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *CAISE*, pages 48–63, 2008.
7. P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, and F. Leymann. Simpl - a framework for accessing external data in simulation workflows. In *BTW*, pages 534–553, 2011.
8. N. Russell, A. ter Hofstede, D. Edmond, and W. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *ER 2005*, volume 3716 of *LNCS*, pages 353–368. Springer Berlin / Heidelberg, 2005.
9. S. Schick, H. Meyer, M. Bandt, and A. Heuer. Enabling yawl to handle dynamic operating room management. In *BPM Workshops (2)*, pages 249–260, 2011.
10. S. Schick, H. Meyer, and A. Heuer. Enhancing workflow data interaction patterns by a transaction model. In *ADBIS*, pages 33–44, 2011.
11. S. Schick, H. Meyer, and A. Heuer. Flexible publication workflows using dynamic dispatch. In *ICADL*, pages 257–266, 2011.
12. S. Schick, H. Meyer, and A. Heuer. FlexY: Flexible, datengetriebene Prozessmodelle mit YAWL. In *BTW*, pages 503–506, 2013.

YAWL4Industry: Reflections on using YAWL for Industry Projects

M. T. Wynn, C. Ouyang, and M. Adams

Queensland University of Technology, Brisbane, Australia.
{m.wynn,c.ouyang,mj.adams}@qut.edu.au

Abstract. The Yet Another Workflow Language (YAWL) language and environment has been used to prototype, verify, execute and analyse business processes in a wide variety of industrial domains, such as telephony, construction, supply chain, insurance services, medical environments, personnel management and the creative arts. These engagements offer the YAWL researcher community a great opportunity to validate our research findings within an industry setting, as well as discovery of possible enhancements from the end user perspective. This paper describes three such industry projects, discusses why YAWL was chosen and how it was used in each, and reflects on the insights gained along the way.

Keywords: YAWL, Case Studies, Industry Experience, Deployment, Uptake

1 Introduction

The YAWL language and environment is well-known in the areas of research, for topics such as verification, configuration, exception handling, visualisation, risk-awareness and cost-awareness, and teaching, having been taught in the courses of at least 36 universities across 20 countries. Originally intended as a reference implementation of workflow patterns, the open-source YAWL environment provides an ideal testbed for novel research ideas. In the decade since the YAWL language was first developed, we, as YAWL researchers, have modelled numerous YAWL processes that reflect various scenarios in different application domains – ranging from structured transactional processes (e.g., insurance claim, travel booking, conference paper review processes) to manual and flexible processes (e.g., scheduling of medical operations). In most cases, our main focus is on *evaluating a specific research artefact*. Hence, we abstract from things that are less relevant to our purpose, for example, in a particular research project we might not spend much time on the development of intuitive user interfaces or we might not fully integrate a YAWL process with external systems. In addition, there may be limited stakeholder involvement in terms of user acceptance testing of the results of particular projects.

In recent years, a number of organisations have initiated YAWL deployments to prototype, verify and execute their business processes. This paper summaries the insights gained from our involvement in such projects in three different domains: construction, the creative arts, and telephony, whereby we *demonstrated how YAWL can be used to model and automate processes within a particular domain* in collaboration with domain experts who were actively involved in requirements analysis and design, process modelling, implementation and acceptance testing of the resulting system.

2 Case Studies

Each of the selected case studies was undertaken in close collaboration with domain experts, and each case delivered a YAWL environment tailored to organisational needs. The prototype process for construction is used for demonstration purposes, the YAWL4Film system was trialled and deployed in the real film production setting, and the system developed by first:utility was for ongoing production. A brief overview of the three case studies is given below.

2.1 Construction

Construction processes are complex, involving planning of activities from design to build, coordination between various teams (e.g. designers, contractors, builders, engineers, inspectors), and intensive use of building materials, equipment, and tools. In recent years, off-site manufacture (OSM) has been recognized as an effective way to reduce cost at the procurement stage of construction. Currently, construction management practices are predominantly manual and founded on the project manager's experience. This case study aimed to demonstrate how business process automation techniques can assist in capturing OSM requirements within a construction project.

A collection of process models (using BPMN notation) that represent the generic construction value chain was developed [2]. Based on the input from the domain experts from the Australian construction industry, key activities, resources, data and stakeholders involved in each activity within the construction value chain were identified. A prototype YAWL application was then developed to showcase the ability of workflow systems to support and coordinate OSM-related process activities [7] and demonstrated to stakeholders. The YAWL environment was chosen due to its ability to coordinate multiple parties involved in a construction project as well as its comprehensive support for rapid prototyping.

2.2 YAWL4Film

Processes in the field of screen business are characterised with high demands for creativity and flexibility. A typical process is one that supports film production, an important phase that encompasses when the majority of cast and crew are contracted and the majority of the equipment and resources are utilised. A film production is a highly manual process that requires handling large amounts of heterogeneous data on a daily basis and coordinating many geographically distributed stakeholders. The benefits offered by applying automation to such a process are twofold: it may ultimately reduce the production costs by minimising errors and delays; and by saving time otherwise spent in costly and tedious procedural tasks, the production team can focus more on creative activities thus increasing the quality of the final product [4].

The development of a prototype system, namely *YAWL4Film* demonstrated that the benefits stated above can be brought to the field of screen business through the application of workflow technologies [3]. The process model captures the control flow (see Figure 1), data and resource perspectives of a standard film production process. An important feature of YAWL4Film lies in the modelling of intensive data associated with the process, which had led to the development of *customised user forms* to support templates used in professional film making, and implemented using JSP, XSLT, and related Web technologies. A trial application of the system was conducted during two student productions at the Australian Film, Television and Radio School in 2007, which was followed in 2008 by real deployment of the system in the production of a medium-budget feature film by Porchlight, an independent film production company.

2.3 first:utility

First Utility Ltd (first:utility) is an independent utilities group based in the United Kingdom. Amongst other interests, it provides a variety of competitive telephony services to its customers, through the leasing of wholesale telephony lines from British Telecom (BT), which it on-sells to customers at competitive rates [1]. BT provides a *Service Providers Gateway* (SPG) through which providers can access BT's internal systems. The gateway may be accessed via a browser based session, which is slow and open to input and transaction errors, or via an API, which is suitable for high volumes of order processing. Of particular interest is the use of the SPG to place an order to have a customer's telephone line transferred from BT to the provider. However, processing an order can involve a number of potential errors, or *exceptions*, which require suitable recovery and retry processing to ensure an order can be successfully completed.

An implementation of the YAWL environment was chosen by first:utility to provide workflow support for both back-office automation and the human-controlled work activities of their Customer Relationship Management (CRM) systems. For back-end automation such as the line transfer order process, YAWL provided a number of advantages. One was the ease with which a custom service could be developed to interface with the SPG. Another was the various mechanisms available to build-in exception detection and recovery into long-lived processes by design. For CRM workflows, first:utility developed their own worklist extension that generates web-based forms using runtime attributes and XSLT transformations.

3 Insights

3.1 The YAWL Language

For all three case studies, the YAWL language was found to be capable of supporting all the required control flow behaviours for the processes, making use of various complex control flow constructs including multiple instances, cancellation, advanced synchronisation, composite and timer tasks [6]. For example, Figure 1 shows the YAWL model of the film production process. Using an OR-join together with loop constructs, we were able to specify precisely the process logic in a less 'crowded' way. The graphical notation of YAWL was found to be easily understood by designers, domain experts and stakeholders. For the resource perspective, most tasks were modelled using the role-based allocation mechanism with additional constraints making use of separation of duties, retain familiar and pile execution resource patterns where appropriate. For the data perspective, the support for user-defined complex XML datatypes made the task of defining the data requirements straightforward.

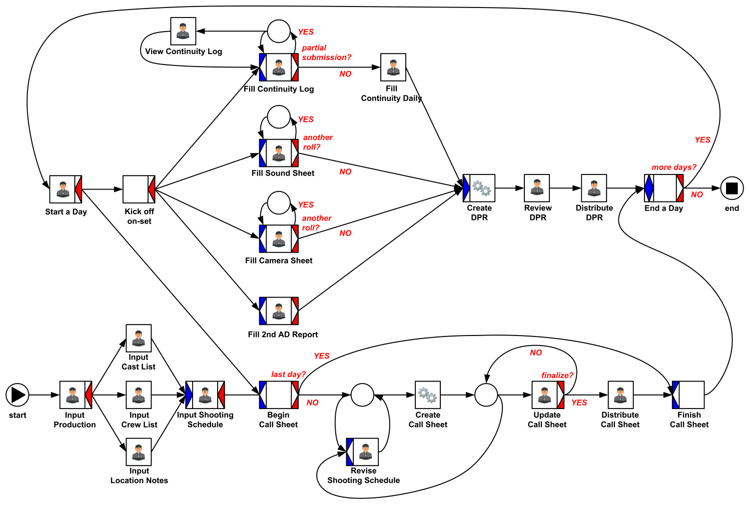
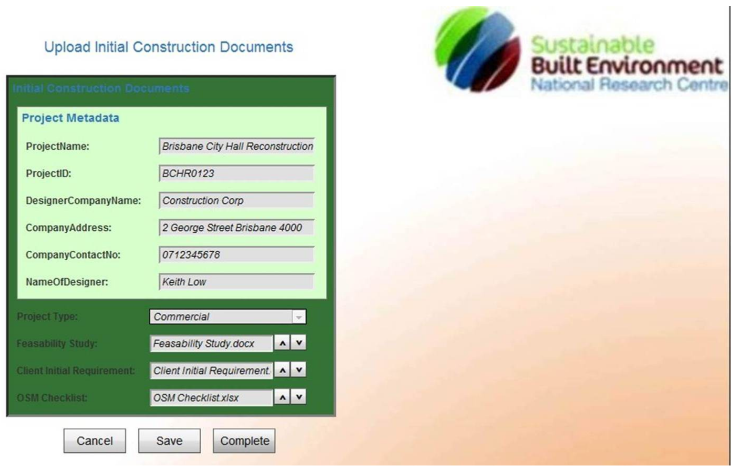


Fig. 1. The YAWL model of a film production process [3].

3.2 The YAWL Framework

Because the YAWL environment is built on a Service Oriented Architecture, it allows the central workflow engine to be readily extended through the development of value-added services [5]. A number of services are included in each YAWL core deployment, including those that extend the environment to include the resource perspective, flexibility and exception handling, inter-process interoperability, resource and activity scheduling, cost-awareness, document sharing, email services and simulation. The environment can be further extended by organisations developing their own custom services to perform specific activities, such as the first:utility SPG interfacing service described earlier. In addition, by introducing a small number of extensions to the environment, first:utility was able to suspend and resume cases and manually raise compensation processes to manage exceptions.

Another advantage of the YAWL environment that made it attractive to first:utility was its scalability. With many thousands of concurrent process instances, first:utility was able to scale the environment across a number of servers, running several engine instances in parallel. They also created a 'quarantine' server, where processes that had experienced a failure could be ported and closely examined to determine the root cause of the problem. Once rectified, the process could be moved out of quarantine and back into the production environment.

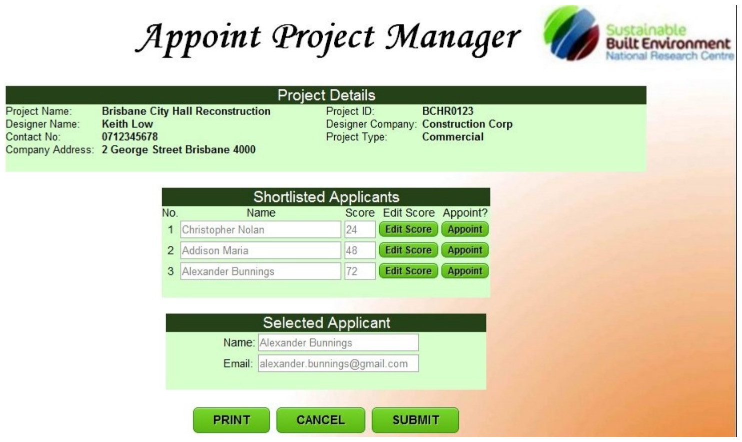


ProjectName:	Brisbane City Hall Reconstruction
ProjectID:	BCHR0123
DesignerCompanyName:	Construction Corp
CompanyAddress:	2 George Street Brisbane 4000
CompanyContactNo:	0712345678
NameOfDesigner:	Keith Low
Project Type:	Commercial
Feasability Study:	Feasability Study.docx
Client Initial Requirement:	Client Initial Requirement.
OSM Checklist:	OSM Checklist.xlsx

Fig. 2. An example dynamic form (with extended attributes) created for the construction workflow case study.

3.3 User Customisations

Although the auto-generated web forms provided by the YAWL environment are very useful for rapid prototyping purposes, we found that most end users prefer customised web forms in the final product. The YAWL environment supports this customisation requirement in two ways. First, by using the extended attributes feature, the colour, font and label names of auto-generated forms can be customised at design time. This enables a YAWL consultant to quickly change the look-and-feel of auto-generated forms.



Appoint Project Manager

Project Details

Project Name:	**Brisbane City Hall Reconstruction**	Project ID:	**BCHR0123**
Designer Name:	**Keith Low**	Designer Company:	**Construction Corp**
Contact No:	**0712345678**	Project Type:	**Commercial**
Company Address:	**2 George Street Brisbane 4000**		

Shortlisted Applicants

No.	Name	Score	Edit Score	Appoint?
1	Christopher Nolan	24	Edit Score	Appoint
2	Addison Maria	48	Edit Score	Appoint
3	Alexander Bunnings	72	Edit Score	Appoint

Selected Applicant

Name: Alexander Bunnings

Email: alexander.bunnings@gmail.com

Fig. 3. An example custom form created for the construction workflow case study.

Figure 2 illustrates a screen shot of a dynamic form created for the construction workflow case study where the colour theme of the SBEnrc centre was utilised. Second, by allowing a custom web form to be associated with a YAWL task, a YAWL consultant is free to create custom forms from scratch and also add new functionalities to the form. In Figure 3, the user interface for the "Appoint Project Manager" task for the construction process is shown (where three additional functionalities were added - "Edit Score", "Appoint" and "Print"). A significant amount of time was spent to develop intuitive and easy-to-use user interfaces in all three case studies. This customisation capability is found to be an essential feature in improving usability and user acceptance of the YAWL environment within an industry setting.

4 Conclusion

Based on our experience in applying YAWL within industry settings, the key strengths of the YAWL environment can be summarised as follows.

– The YAWL language is expressive and supports all necessary constructs for industry projects.
– The YAWL environment is an excellent tool for rapid prototyping.
– The extensibility of the open-source YAWL environment makes it possible for organisations to develop their own customised services.
– The YAWL environment enables the development of customised user interfaces allowing organisations to ensure consistent look and feel across their applications.

Some of the key lessons learned from our experience to develop and deploy a YAWL system are as follows. First, although it is quick to develop a working

YAWL prototype, a significant amount of project time is spent on developing customised user interfaces and custom services in most cases. Secondly, to develop a YAWL system that makes appropriate use of complex control flow constructs supported by the YAWL environment, a YAWL consultant should be well versed in the semantics of the YAWL language. The same is true for the data and the resource perspectives supported by the YAWL environment. Finally, a YAWL consultant should take into account the fact that the YAWL environment is an open-source research environment with functionalities being added by many participants over time and that some of these functionalities may not have been extensively tested before their release.

Work currently in progress, including a redesign of the YAWL Editor and an automated system update facility, may encourage more widespread adoption of the YAWL language and environment in industry. More extensive training materials have also been identified by designers and domain experts as desirable.

References

1. Michael James Adams. *Facilitating dynamic flexibility and exception handling for workflows.* PhD thesis, Queensland University of Technology Brisbane, Australia, 2007.
2. Russell Kenley, Sittimont Kanjanabootra, Chun Ouyang, and Moe Wynn. Procuring osm: Base-line models of off-site manufacture business processes in australia. In *Proceedings of the 16th Pacific Association of Quantity Surveyors Congress*, Brunei, Darussalam, Jul 7-10 2012.
3. Chun Ouyang, Marcello La Rosa, Arthur HM ter Hofstede, Marlon Dumas, and Kathrine Shortland. Toward web-scale workflows for film production. *Internet Computing, IEEE*, 12(5):53–61, 2008.
4. Chun Ouyang, Kenneth Wang, Arthur ter Hofstede, Marcello La Rosa, Michael Rosemann, Katherine Shortland, and David Court. Camera, set, action: Process innovation for film and tv production. *Cultural Science*, 1(2), 2008.
5. Arthur HM ter Hofstede, Wil MP van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation: YAWL and its support environment.* Springer, 2010.
6. Wil MP van der Aalst and Arthur HM ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.
7. Moe Wynn, Chun Ouyang, Wei Zhe Low, Sittimont Kanjanabootra, Toby Harfield, and Russell Kenley. A process-oriented approach to supporting off-site manufacture in construction projects. In *Proceedings of CIB World Building Congress*, Brisbane, Australia, 5-9 May 2013.

Supporting the Workflow Management System Development Process with YAWL

R.S. Mans[1], W.M.P. van der Aalst[1]

Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{r.s.mans,w.m.p.v.d.aalst}@tue.nl

Abstract. In order to address particular needs from the healthcare domain, the open-source YAWL Workflow Management System (WfMS) has been extended with features for scheduling support and inter-workflow support, the YAWL4Healthcare WfMS. As part of this undertaking, a WfMS development approach has been followed in which the same conceptual model is used for *specifying*, *developing*, *testing*, and *validating* the operational performance of a new system. In this paper, we elaborate on the features of YAWL that were essential for realizing our development approach.

1 Introduction

Nowadays, hospitals are investigating the introduction of a Workflow Management System (WfMS) in order to support their care processes. However, due to the complex nature of healthcare processes, these systems need to be enriched with additional functionality. Furthermore, the introduction of new technology requires a seamless integration with running operational processes and no unexpected break-downs may occur.

For ensuring above mentioned aspects, as shown at the bottom of Figure 1, we propose a WfMS development approach consisting of five phases. First, in the *requirements* phase, the required functionalities are identified. Second, during the *design* phase, a conceptual model is developed which is a formal, complete, and executable specification of the WfMS to be developed. Subsequently, the WfMS is developed in the *implementation* phase. Finally, during the *testing* and *simulation* phase, the conceptual model and the operational WfMS are used to both test and validate the operational performance of the WfMS.

The conceptual model has been defined in terms of a *Colored Petri Net* (CPN) [3]. CPNs provide a well-established and well-proven formal language suitable for describing the behavior of systems exhibiting characteristics such as concurrency, resource sharing, and synchronization. Moreover, we can use CPN Tools for specification, verification, and simulation. As concrete WfMS, the open-source *YAWL* WfMS [2] has been used. In this paper, we focus on the role of YAWL in the WfMS development approach. That is, we elaborate on the specific features of YAWL that were essential for realizing our approach.

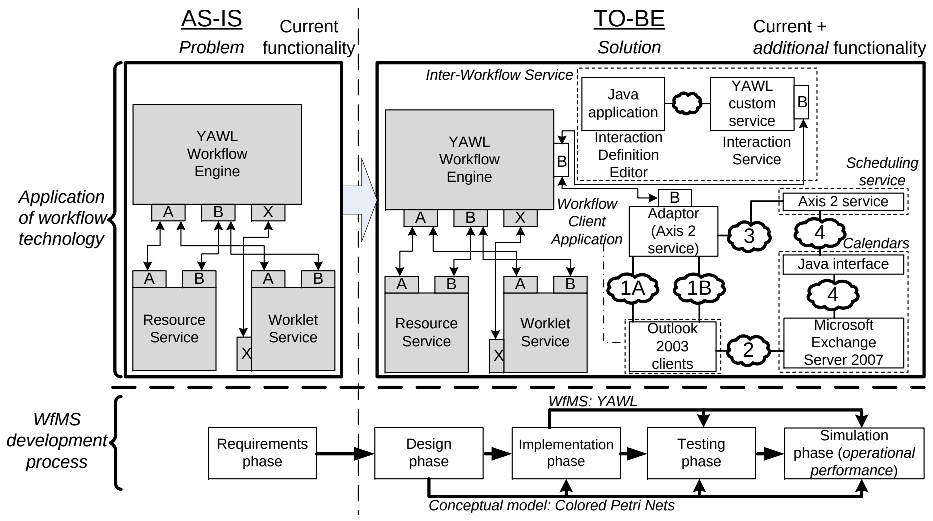


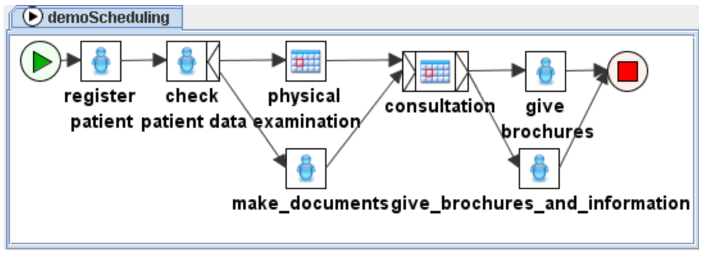
Fig. 1. Workflow Management System development approach using YAWL and the CPN conceptual model.

When applying the development approach, in the *requirements* phase it has been identified that for optimally supporting healthcare processes, WfMSs need to be extended with facilities for both *scheduling support* and *inter-workflow support* [4]. Based on the conceptual CPN model defined in the *design* phase, YAWL has been extended with the aforementioned two facilities resulting in the *YAWL4Healthcare* system and which is discussed in Section 2. Next, in Section 3 we elaborate on the *testing* and *simulation* phase in which YAWL4Healthcare is both tested and validated. Finally, we conclude in Section 4.

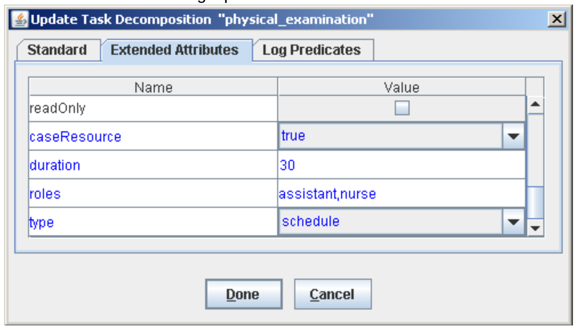
2 Scheduling Support and Inter-Workflow Support

YAWL4Healthcare offers both scheduling support and inter-workflow support. In this section, both facilities are discussed in detail and it is indicated which features of YAWL were essential for realizing them.

With regard to scheduling support, instead of only offering a workitem to a user via a worklist, it also needs to be possible to make an appointment for a workitem. This appointment has a specific duration and only appears in the calendars of these users that are involved in the execution of the workitem. The main scheduling features will be illustrated using a small scenario. The process definition specified in the YAWL editor can be seen in Figure 2a. The “physical examination” and “consultation” tasks are annotated with a calendar icon as for both an appointment is needed. Moreover, they are called *schedule* tasks. Via the “extended attributes” feature of the editor, additional attributes are defined for them. These attributes are also illustrated in Figure 2a in which for the “physical examination” it is defined that the presence of the patient is required during

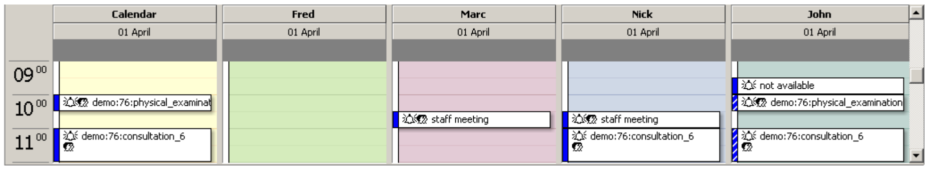


a) Defining a model in the YAWL editor. For tasks annotated with a calendar icon an appointment is needed whereas for tasks annotated with a single person icon this is not needed.



Name	Value
readOnly	☐
caseResource	true
duration	30
roles	assistant,nurse
type	schedule

b) The details for the 'physical examination' task are defined via the extended attributes feature of the editor.



	Calendar	Fred	Marc	Nick	John
	01 April	01 April	01 April	01 April	01 April
09^{00}					not available
10^{00}	demo:76:physical_examinat		staff meeting	staff meeting	demo:76:physical_examination
11^{00}	demo:76:consultation_6			demo:76:consultation_6	demo:76:consultation_6

c) State of the calendars after scheduling. The calendars of assistant 'Jane', assistant 'Fred', doctor 'Marc', doctor 'Nick', and patient 'John' are shown respectively. When scheduling the availability of resources is taken into account.

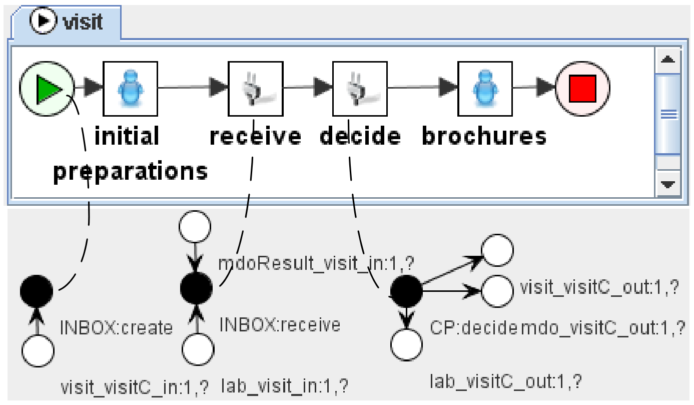
Fig. 2. Scheduling of appointments within the YAWL4Healthcare system.

the appointment (caseResource), the average duration of the appointment is 30 minutes (duration), an assistant and a nurse are required during the appointment (roles), and the task is a schedule task (type). The tasks for which workitems need to be offered via a worktray are called *flow* tasks and are indicated by a person icon. Also, for this kind of task, additional information is defined via the "extended attributes" feature of the editor (e.g. the average duration).

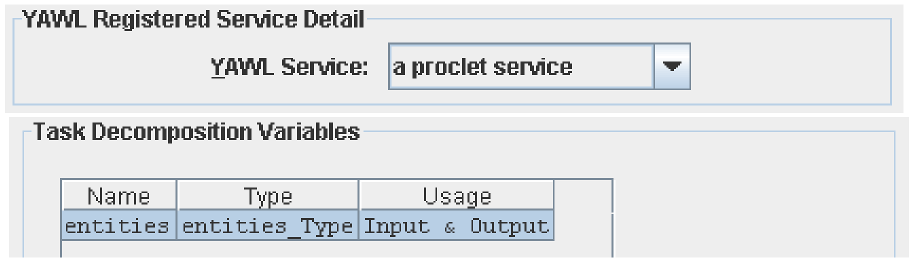
Once an instance of a process is started, appointments are automatically booked in the calendars of these persons that are involved in the execution of a schedule task. This is shown in Figure 2b in which an appointment is booked for the "consultation" task which appears in the calendars of doctor "Nick" and patient "John". Also, an appointment is booked for the "physical examination". Note that the system ensures that the final scheduling of tasks occurs in the same order as the sequence of schedule tasks in the accompanying process definition for the case. Moreover, sufficient time is reserved between two scheduled tasks. In case it is found out that too little time is left for performing preceding work-items for a scheduled schedule task, the corresponding appointment is automatically rescheduled. Also, users are able to express their dissatisfaction with the nominated scheduling by requesting: (1) the rescheduling of the appointment, (2) the rescheduling of the appointment to a specified date and time, or (3) the reassignment of the appointment to another employee.

In order to offer scheduling facilities (see Figure 1), YAWL has been extended with three components. The *Calendars* component provides a view on the calendars of users and allows for manipulating them. Based on the calendars, the *Scheduling Service* component provides the scheduling facilities to the WfMS. The workitems and any appointments for them can be seen via the *Workflow Client Application*. The "Calendars", "Scheduling Service", and "Workflow Client Application" components have respectively been realized by developing a java service, using Microsoft Exchange Server 2007, and using Outlook 2003 clients. Via a specific adaptor, communication takes place between the YAWL workflow engine and the "Scheduling Service" and the "Workflow Client Application". Moreover, in this way, only Interface B is needed for communication with the engine, i.e. starting and canceling instances, and the checking in and out of workitems. Note that the scheduling algorithm used by the Scheduling Service can easily be replaced by another algorithm, i.e., it is *pluggable*. Currently, a "naive" algorithm is implemented which searches for the first opportunity in which one of the resources of a role can be booked for the respective work-item.

The need for inter-workflow support emerged from the fact that the process of treating a patient typically consists of many smaller interacting workflow fragments that run in conjunction with each other. These fragments may also operate at different levels of granularity. The main inter-workflow support features will be illustrated using a small scenario. Note that these features are based on the Proclets framework [1] which allows for modeling and executing lightweight workflows that may interact with each other and reside at different levels of granularity. At the top of Figure 3a, the process definition of a workflow fragment, describing a visit to the hospital, is defined in the YAWL editor. At the bottom, the associated definition in the Interaction Definition Editor is shown. Via this editor it is possible to define for a workflow fragment which interactions with other fragments are necessary. That is, for tasks and input conditions for which interactions with other fragments are necessary, a so-called interaction point (visualized by a black dot) is defined. Via ports (visualized by a white dot), interaction points are connected with interaction points of other fragments. For example, for the "decide" task it is possible to instantiate a workflow fragment in order to perform a lab test or to instantiate a fragment arranging the next visit of the patient. Moreover, the patient can be registered for a multidisciplinary meeting in which the status of multiple patients is discussed. Note that at run-time for a certain entity (e.g. a patient or a lab test) an *interaction graph* is kept. This graph stores for an entity (e.g. a patient) all the interactions that need to take place between (future) fragments. As part of this, for a task for which interactions are needed, in the YAWL editor it is indicated that its exe-



a) Defining a workflow fragment and the corresponding interactions in the YAWL editor (top) and the Interaction Definition Editor (bottom). Via dotted arcs, the tasks and the interactions with other fragments that are defined for them are indicated.



YAWL Registered Service Detail

YAWL Service: a proclet service

Task Decomposition Variables

Name	Type	Usage
entities	entities_Type	Input & Output

b) For the 'decide' task it is indicated in the 'Task Decomposition' details that its execution is delegated to the Inter-Workflow Service. Also, the 'entities' variable allows for indicating at run-time the names of the entities for which the process is executed.

Fig. 3. Definition of interactions between workflow fragments.

cution is delegated to the Inter-Workflow service (see Figure 3b). Additionally, information about the involved entities is exchanged via the "entities" variable.

In order to offer inter-workflow support (see Figure 1), YAWL has been extended with the *Inter-Workflow Service*. The "Interaction Service" is responsible for the interactions between fragments at run-time and has been set-up as a YAWL custom service. As such, it communicates with the YAWL engine via Interface B. The "Interaction Definition Editor" offers tools for defining interactions between fragments at both design-time and run-time.

So, it can be concluded that *the service oriented architecture of YAWL was of great help for realizing the desired extensions*. That is, via the extended attributes feature of the editor, additional attributes of a task that need to be filled in, could easily be defined. Furthermore, due to the fact that the YAWL engine is agnostic to its external services, it was possible via a single interface (Interface

B) to focus only on implementing the functionalities that were needed for the desired scheduling support and inter-workflow support. So, we did not need to bother about basic workflow management functionalities as they were already provided by YAWL itself (e.g. a workflow engine and process editor).

3 Replacement of Functionalities

As illustrated in Figure 1, the conceptual model developed using CPN Tools has been used for both testing and validating the operational performance of the implemented YAWL4Healthcare WfMS in respectively the *testing* and *simulation* phase. In Figure 4 it is schematically depicted how this has been realized.

The CPN conceptual model provides a complete, formal description of the functionality of the system. As this model is executable, it also serves as a prototype implementation of the system. So, in the *testing* phase, one or more parts of the CPN model can be *replaced* by the concrete implementation of these parts. So, in the figure, the light grey-colored rectangle represents the parts of the actual system that are tested. This is done by establishing connections between the CPN model and parts of the actual YAWL4Healthcare WfMS which need to be tested. During the testing, parts of the system, which are not tested, may be simulated using CPN Tools (the dark grey-colored rectangle). This allows for the testing of numerous scenarios facilitating the discovery of potential flaws in both the architecture and the corresponding implementation.

In the *simulation* phase, our focus is on investigating the operational performance of processes that are supported by both the *designed* and *realized* system. Here we can benefit from the fact that CPN models can also be used for simulation-based performance analysis in order to evaluate systems and to compare alternative configurations of a system [3]. So, for processes supported by our system, it can be investigated whether they are negatively impacted or not (e.g. introduction of bottlenecks into the process). By setting up a reference environment, the operational performance can be investigated in a structured manner. Moreover, parts of the realized YAWL4Healthcare WfMS can be included in the CPN simulation model in order to analyze the implemented system.

At the time the above mentioned approach was applied, the "Inter-Workflow Service" was not developed yet. So, in the conceptual model, we only had com-

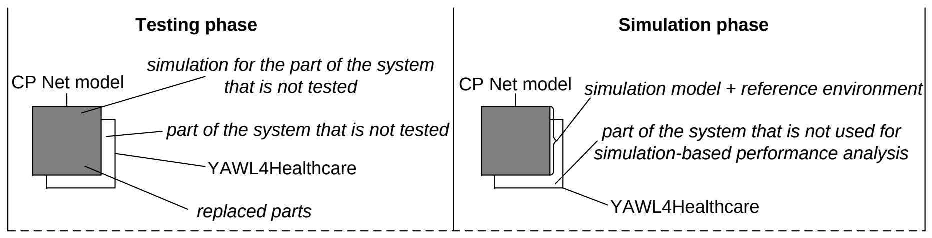

Testing phase
CP Net model
simulation for the part of the system that is not tested
part of the system that is not tested
YAWL4Healthcare
replaced parts
Simulation phase
CP Net model
simulation model + reference environment
part of the system that is not used for simulation-based performance analysis
YAWL4Healthcare

Fig. 4. The role of the CPN conceptual model and the YAWL WfMS in the testing and simulation phase.

ponents for the "Workflow Engine", "Workflow Client Application", "Calendars", and "Scheduling Service". In the testing phase, we could easily replace the scheduling service and both the workflow engine and scheduling service by their implemented YAWL4Healthcare counterparts. Furthermore, the engine was provided with a simple process definition. Also, four users where created in the "Workflow Client Application" together with the associated empty calendars in the "Calendars". Amongst others, 6 errors were identified in the "Scheduling Service" component and 6 errors were identified in the adaptor component. In the simulation phase, also both the "Workflow Engine" and "Scheduling Service" were replaced by their implemented counterparts. Furthermore, the engine was provided with a gynecological healthcare process such that a series of simulation experiments could be carried out in which 143 patients followed the process. For the schedule tasks, the corresponding appointments were made by our system. Therefore, the calendars of the simulated system were filled with appointments, etc. to mimic the true availability of the medical staff. As a result it was possible to explore different scenarios. For example, in one of the experiments we simulated the situation that the appointments for a CT, MRI, and pre-assessment are scheduled on the same day. This resulted in a considerably increased waiting time for both the pre-assessment and examination under anesthetic appointments.

Similarly, as in Section 2, also here we benefited from the service-oriented architecture of the YAWL WfMS. Due to Interface B, components in the conceptual model could easily be replaced by their implemented counterparts.

4 Conclusion

In this paper, we focussed on the role of YAWL in a development approach in which the same conceptual model is used during the design, implementation, testing, and simulation phase. By applying this approach, YAWL has been extended with facilities for scheduling support and inter-workflow support.

As part of this undertaking, we greatly benefited from the service oriented architecture of YAWL. As it turned out, this provided a powerful point of extensibility, allowing for the extension of the engine with additional desired functionalities. Moreover, it allowed for the applicability of our development approach in which there is a tight coupling between the conceptual model and the YAWL4Healthcare WfMS. While testing and validating, parts of the system may be simulated while connected to the actual system components.

References

1. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001.
2. A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment.* Springer-Verlag, 2010.

3. K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.
4. R.S. Mans. *Workflow Support for the Healthcare Domain.* PhD thesis, Eindhoven University of Technology, June 2011. See http://www.processmining.org/blogs/pub2011/workflow_support_for_the_healthcare_domain.

YAWL in the Cloud

D.M.M. Schunselaar*, T.F. van der Avoort, H.M.W. Verbeek*, and W.M.P. van der Aalst*

Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar, h.m.w.verbeek, w.m.p.v.d.aalst}@tue.nl

Abstract. In the context of the CoSeLoG project (which involves 10 Dutch municipalities), we realised a proof-of-concept implementation based on YAWL. The municipalities want to share a common IT infrastructure and learn from one another, but also allow for local differences. Therefore, we extended YAWL to run in a cloud-based environment leveraging on existing configuration possibilities. To support "YAWL in the Cloud" we developed load-balancing capabilities that allow for the distribution of work over multiple YAWL engines. Moreover, we extended YAWL with multi-tenancy capabilities: one municipality may effectively use multiple engines without knowing it and one engine may safely run the processes of multiple municipalities.

1 Introduction

Within the Netherlands, more and more municipalities seek cooperation to cut costs. One of the directions to cut costs is to share infrastructures supporting the processes of municipalities. Every municipality has a server/ business process management system (BPM system)/ case handling system/ etc. to support them. However, these systems are not used to their fullest capacity, and also capacity might change, e.g., decrease, during the year. Municipalities can cut costs by sharing infrastructure and have an adaptive system to handle the peaks. Within the CoSeLoG project, we are cooperating with 10 municipalities who want to cooperate with each other and learn from each other[1] [1]. One of the elements of the project is a shared infrastructure where the municipalities share IT-solutions and processes.

In recent years, we created many (configurable) YAWL models [2–4] for the processes of the municipalities. Therefore, we want to use YAWL to create a proof-of-concept implementation, showing that municipalities can share a common infrastructure and still allow for the necessary "colour locale". Unfortunately, YAWL has been designed to be used on a single machine. If we are using a single machine, this requires the owner of this machine to share (part of) her infrastructure which might not always be desirable. Furthermore, we are dealing

* This research has been carried out as part of the Configurable Services for Local Governments (CoSeLoG) project (http://www.win.tue.nl/coselog/).

[1] http://www.win.tue.nl/coselog/

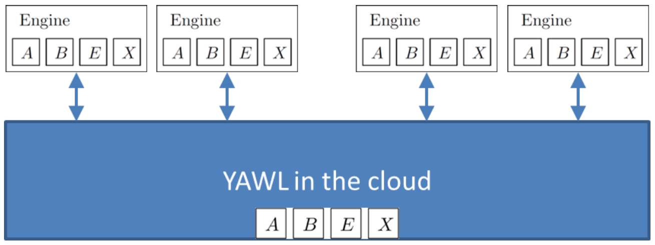


Fig. 1: The positioning of YAWL in the cloud.

with different organisations which do not want to expose some or all of their information to the other parties involved. Finally, by combining different organisations onto a single machine, this machine might become overloaded by a shared peak caused by the organisations.

In order to overcome the mentioned problems, we propose *YAWL in the cloud*. The cloud is tailored towards scalable resources to increase the computing power when necessary and to decrease the computing power when possible. Since the cloud can be maintained by a third party, there is no need to employ a person to maintain the systems. However, in order to make YAWL run in the cloud, we have to extend the standard YAWL architecture to deal with problems introduced by having multiple YAWL engines running simultaneous (e.g., non-unique case numbers amongst different engines). Therefore, we first introduce the general framework built around YAWL. Afterwards, we present the implementation and show some innovative features of YAWL in the cloud. Finally, we discuss the limitations and future work.

2 Architecture

An early design decision was to not change the YAWL system itself. By not changing YAWL itself, we are not bound to a specific (modified) version of YAWL. Moreover, to avert having to learn a new system, the end-user should not notice that she is working in the cloud. Finally, there has to be an additional component to control and observe the current state of YAWL in the cloud. This resulted in the high level architecture as depicted in Fig. 1, note that A, B, E, X are the interfaces of a YAWL engine. For each of the constraints, we list part of the effect it had on the architecture. Please note (as Fig. 1 shows), we are running multiple YAWL engines on multiple machines.

Instead of running a single YAWL server on a single machine, we assume that we are running multiple YAWL servers on multiple machines. The combination of these servers/machines is YAWL in the cloud.

No change to YAWL: By having multiple YAWL engines, it is no longer apparent to which engine to connect. Therefore, we introduce a *router* component. This router component routes the requests to the correct YAWL engine.

By not changing YAWL, each YAWL engine is unaware of the other YAWL engines. Therefore, it can no longer be guaranteed that every case identifier is unique. In order to overcome this, we introduce unique global cloud identifiers and maintain a mapping between global (cloud assigned) identifiers and local (engine specific) identifiers. This mapping is stored in a central database, and the transformations of the global/local identifiers to local/global identifiers (and vice versa) is done by the router.

We can now receive a request and route it to the correct engine(s). However, when multiple engines have to be consulted, we also obtain multiple responses. Therefore, we need to merge these responses in a single response before sending it back to the requestor. The merging of the responses is handled by the router.

Finally, it might be the case that not all the information in a response is intended for the requestor. This can be the case when an engine is running multiple cases for different tenants. Since the YAWL engine does not know the notion of multiple tenants, it sends information about all the tenants back as a response. Therefore, we also include filtering functionality in the router.

Invisible cloud infrastructure: YAWL is decomposed into components, e.g., engine, resource service, process designer. This decomposition yields that we only have to take care that the resource service is directed to *YAWL in the cloud* instead of an actual engine. This change is only a configuration of the resource service. By using the resource service as is, we do not change the front-end view of the end-user.

Management component: For the management component, we require a number of views on the cloud. One of these views is a functional view denoting per tenant which specifications and cases are loaded. Apart from a functional view, we also want a software view denoting the hierarchy of servers, engines, tenants, specifications, and cases. Furthermore, this management view should be usable to create new tenants, and bring new engines into the cloud.

Complete architecture: Taking all the constraints into account, we obtain the more detailed architecture depicted in Fig. 2. At the back, the communication with the different engines is situated. At the front we have a single point of entry for the resource services. Inside of the architecture, we have the routers for routing, translating, merging and filtering of requests and responses. These routers use the central database (DB) for the lookups of identifiers. There is a management component at the centre communicating with the engines and the central database. Finally, we have included a load balancer for both the incoming and outgoing requests. This load balancer is cloud-based and offers the option to enable/disable routers if necessarily. Furthermore, if a router cannot handle the requests, then the load balancer can initiate an extra router.

Within cloud computing there are different deployment models: public, private, community, and hybrid. In our architecture, we do not pose any restrictions on the used deployment model, i.e., our architecture is applicable to all of these

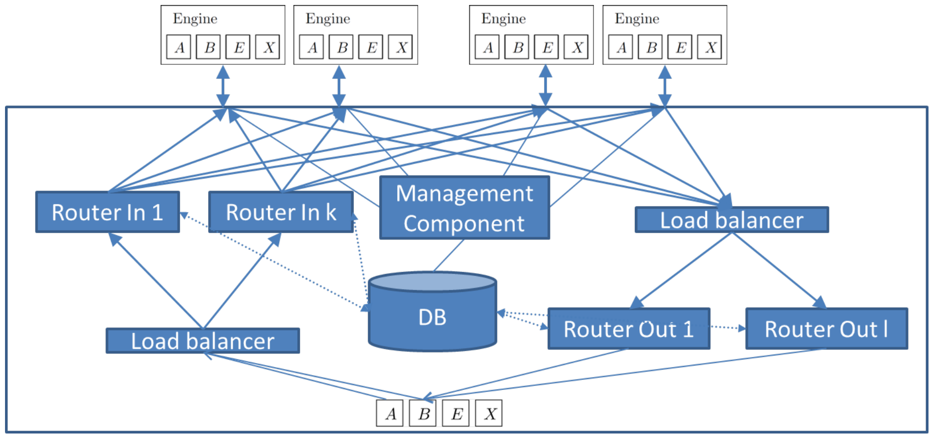


Fig. 2: The architecture for YAWL in the cloud

deployment models. Apart from different deployment models, there exist different service models, i.e., abstraction levels from the underlying hardware. These service models come in 4 different flavours (in ascending abstraction level): Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Business Process as a Service (BPaaS). Our architecture has been designed to be used in a PaaS service model. For a more extensive discussion on service models and deployment models, see [5].

Having presented our architecture, we now present the YAWL in the cloud implementation. For each of the newly introduced components, we show some of the implementation details.

3 Implementation

Using the architecture shown in Fig. 2, we implemented the various components connecting to standard YAWL. Due to space restriction, not all implementation details can be covered. See [5], for a complete overview of the implementation of YAWL in the cloud.

The router: As mentioned, we had to introduce a routing component for routing, translating, merging and filtering of requests and responses before sending them to the requestor. In Fig. 3, the communication between the different components is depicted. First a request is sent to the router, then the router consults the database for (amongst others) translating global identifiers to engine specific identifiers. Afterwards, the router contacts the engines of interest for this request. After the engines have sent their responses, these responses are merged, filtered, and the database is consulted for translation of engine specific identifiers to global identifiers. Finally, the created response is forwarded to the client.

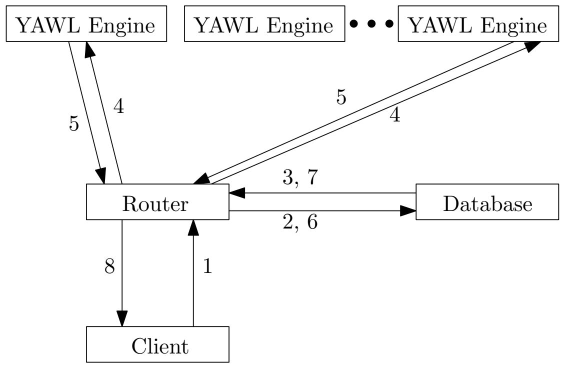


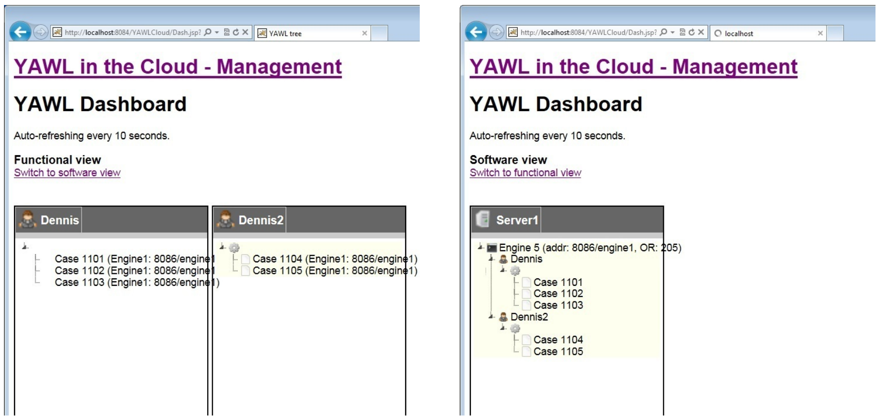
Fig. 3: The communication between the different components when a request is made.

Based on the type of request, different types of merges had to be introduced and rules for forwarding the request to specific engines. For instance, for the action *getAllRunningCases*, which gives all the running cases for a particular tenant, we have to merge the results per specification (i.e., multiple engines can have the same specification and we do not want to duplicate the specification to the user). Furthermore, this request has to be forwarded to all engines running specifications for this tenant. Finally, the cases for specifications not owned by the tenant have to be filtered out.

If we consider the action *getCasesForSpecification*, which gives all the running cases for a particular specification, we only need to forward this request to the engines running this specification. The engines return all the cases for that specification, the cases in the responses have to be merged into a single response. However, the filtering step is not required as this specification belongs exclusively to a specific tenant. Other tenants may use the same specification, but the identifier of this specification is different for different tenants.

The database: In the central database, we store the different local YAWL identifiers and the global cloud identifiers. We have local YAWL identifiers for specifications, cases, and work-items. Apart from storing the identifiers, we also maintain the different tenants and which specification a tenant has at her disposal. Finally, the database stores the settings for YAWL in the cloud, e.g., which constraints are present within YAWL in the cloud. A constraint can be that a tenant can have at most 5 cases per specification.

The management component: The management component is a view on the database and on the engines. In Fig. 4a and Fig. 4b two different views are presented on the systems. The first shows a separation of specifications and cases per tenant. The second shows a hierarchical view of servers, engines, tenants, specifications, and cases.



YAWL in the Cloud - Management

YAWL Dashboard

Auto-refreshing every 10 seconds.

Functional view
Switch to software view

```
Dennis
  Case 1101 (Engine1: 8086/engine1
  Case 1102 (Engine1: 8086/engine1
  Case 1103 (Engine1: 8086/engine1)

Dennis2
  Case 1104 (Engine1: 8086/engine1)
  Case 1105 (Engine1: 8086/engine1)
```

YAWL in the Cloud - Management

YAWL Dashboard

Auto-refreshing every 10 seconds.

Software view
Switch to functional view

```
Server1
Engine 5 (addr: 8086/engine1, OR: 205)
  Dennis
    Case 1101
    Case 1102
    Case 1103
  Dennis2
    Case 1104
    Case 1105
```

(a) Functional view, showing which cases and specifications are loaded per tenant.

(b) Software view, showing the servers, engines, tenants, specifications, and cases, and the hierarchy of them.

Fig. 4: Two different views on the cloud in the management component; per tenant, and hierarchical per server.

Apart from providing a view on the engines and database, the management component also allows to add/remove engines, and to add/remove tenants. Note that this functionality is currently semi-automated as the configuration of the different components still requires human involvement.

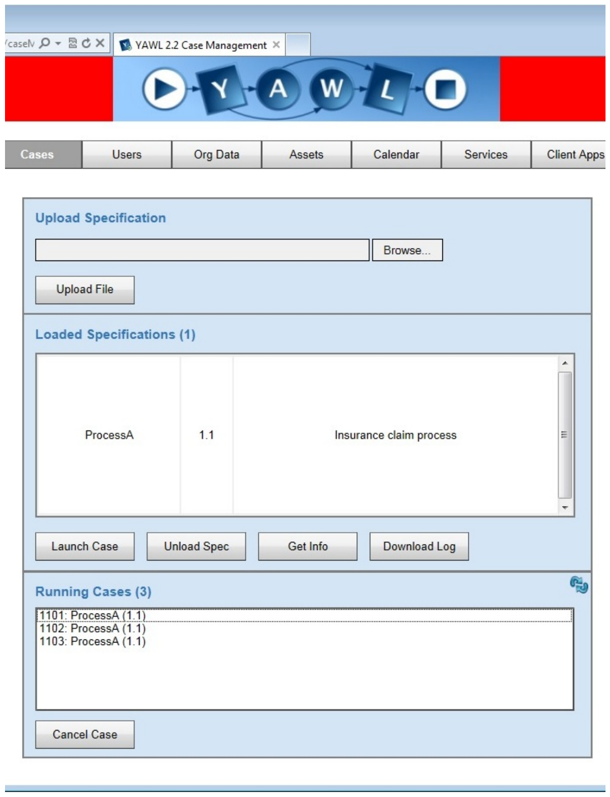
The front-end: Figure 4 shows the administrator view on two tenants. The views the different tenants have are depicted in Fig. 5 and Fig. 6. We have changed the name and colour of the YAWL admin view in order to stress that both tenants have indeed different views. Figures 5 and 6 also show the filtering step of the routers as a tenant only sees her cases, and not the cases of the other tenants.

4 Limitations and Future Work

We have presented our architecture for bringing YAWL in the cloud using the following design decisions: no changes to YAWL, and the end-user should be unaware to the cloud back-end. Furthermore, there had to be a management component to control the cloud. Along with our architecture, we also have shown some of our implementation details.

The implementation of bringing YAWL in the cloud is not complete. For every action on every interface, specific transformations had to be made. Therefore, we have focussed on the main interfaces: interface A inbound, and interface B in- and outbound. In future implementations, we want to extend the support to also include interface E and interface X.

YAWL in the cloud now mainly works in a semi-automated fashion, i.e., some actions require human involvement. In the ideal implementation, we want to have



ProcessA	1.1	Insurance claim process

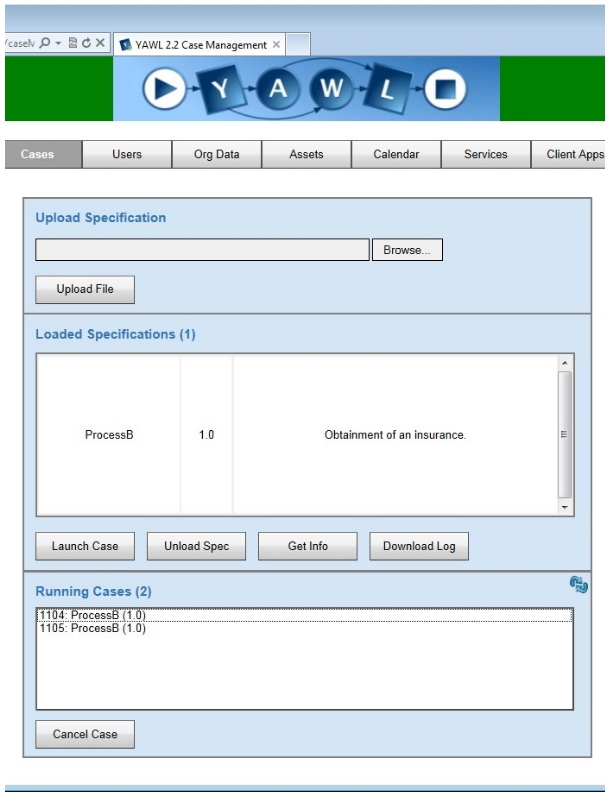
```
1101: ProcessA (1.1)
1102: ProcessA (1.1)
1103: ProcessA (1.1)
```

Fig. 5: The management view of tenant 4.

full automated support for all the features. Using this full automated support, it also allows for automated increases and decreases in computing power. In our current implementation, we have added functionality to optimise the computing power. Unfortunately, the added overhead of YAWL in the cloud is not compensated by the scalability. In the future, we want to profile our implementation and solve the bottlenecks currently in the implementation.

With bringing YAWL in the cloud, we mainly focussed on the engine. The next step would be to also bring the resource service in the cloud. By bringing the resource service in the cloud, we allow for an extra layer of flexibility and a clearer separation between organisations. Thanks to the decomposition of YAWL into different components, one can employ a similar approach to bringing the resource service in the cloud as we have done for bringing the engine in the cloud.

Finally, this research was started with collaboration and knowledge sharing amongst municipalities in mind. In the next step, we plan to use configurable YAWL as the base for the different process models in use by the municipalities. With configurable YAWL, the best practises are captured in a single model,



ProcessB	1.0	Obtainment of an insurance.

1104: ProcessB (1.0)
1105: ProcessB (1.0)

Fig. 6: The management view of tenant 5.

allowing the municipalities to cherry-pick the practises best fit for their organisation.

References

1. van der Aalst, W.M.P.: Business process configuration in the cloud: How to support and analyze multi-tenant processes? In Zavattaro, G., Schreier, U., Pautasso, C., eds.: ECOWS, IEEE (2011) 3–10
2. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N., eds.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
3. La Rosa, M.: Managing variability in process-aware information systems. PhD thesis, Queensland University of Technology (2009)
4. Gottschalk, F.: Configurable Process Models. PhD thesis, Eindhoven University of Technology, The Netherlands (December 2009)
5. Avoort, T.F.v.d.: BPM in the Cloud. Master's thesis, Eindhoven University of Technology, The Netherlands (2013)

Process Monitoring Using Sensors in YAWL

Raffaele Conforti[1], Marcello La Rosa[1,2], and Giancarlo Fortino[3]

[1] Queensland University of Technology, Australia
{raffaele.conforti,m.larosa}@qut.edu.au
[2] NICTA Queensland Lab, Australia
[3] Università della Calabria, Italy
g.fortino@unical.it

Abstract. This article describes the architecture of a monitoring component for the YAWL system. The architecture proposed is based on sensors and it is realized as a YAWL service to have perfect integration with the YAWL systems. The architecture proposed is generic and applicable in different contextes of business process monitoring. Finally, it was tested and evaluated in the context of risk monitoring for business processes.

1 Introduction

The growing number of cases in which workflow management systems are utilized to execute business processes, created the need for companies to monitor the execution of their process instances [4]. Being able to monitor a process instance is a basic requirements for companies that want to prevent the eventuation of risks, be aware of the processing cost of process instances, or simply verify if a process instance is being delayed.

Several commercial workflow management systems already provide monitoring functionality for their systems, e.g. WebSphere[4], Oracle BAM[5], Sybase[6]. This type of functionality is not yet available in open-source workflow management systems, such as the YAWL workflow management system.[7]

In this article we will illustrate how to realize a multi-purpose monitoring component for the YAWL system. The component will be realized as a service for the YAWL system, to guarantee a perfect integration with the system.

This article is structured as follows. Section 2 provides a briefly description of the YAWL systems. Section 3 shows how to realize the monitoring component, which is evaluated in Section 4. Section 5 discusses related work and finally Section 6 concludes the article.

[4] http://www-142.ibm.com/software/products/au/en/subcategory/SW920
[5] http://www.oracle.com/technetwork/middleware/bam/overview/index.html
[6] http://www.sybase.com.au/products/financialservicessolutions/complex-event-processing
[7] http://www.yawlfoundation.org/

2 Requirements and Preliminaries

In order to monitor a business process, being able to have a complete overview of process instances is a requirements. The status of a process instance is fully provided by information about instances of tasks (work items) and subprocesses (nets) belonging to such a process instance.

When we consider a work item, to properly describe its status is essential to know: i) if the work item was performed or not (status in the life-cycle of a work item); ii) who (resource) performed the work item; iii) when the work item was performed (time stamp); and iv) how it was performed (data). A similar set of information is required for an instance of a net, except for the resource.

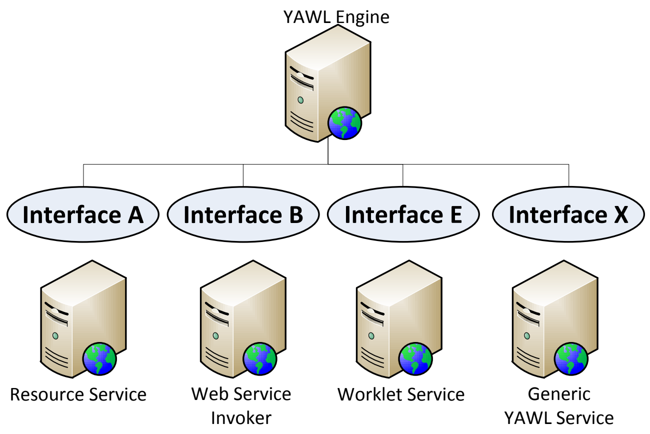


Fig. 1. YAWL Architecture [8]

A clear understanding of the YAWL environment [8] is required if we want to be able to access the status of work items and nets. The YAWL system is a service-oriented architecture built using Java. Figure 1 provides a simplified overview of the architecture of the YAWL system. The core element of this architecture is the YAWL engine, it is in charge of managing the instantiation of process instances and work items. The engine provides four interfaces to allow services to interact with the engine, for example allowing the resource service, which manages resources, to perform a work item instantiated.

These four interfaces are: i) interface A, which provides connection capabilities and allows process models to be uploaded and unloaded, and external services to be registered and unregistered; ii) interface B, which allows process instances to be lunched, work items to be checked-out for their execution, and process information to be retrieved; iii) interface E, which allows process logs to be retrieved; and iv) interface X, which provides a terminal for detecting and handling exceptions.

3 Monitoring Component

We can now describe how to create a monitoring component for the YAWL system, as the one realized for [1]. The best way to realize a new component for the YAWL system is to realize it as a YAWL service. Our service will use the interfaces provided by the system, described in section 2, to receive notifications from the engine about the initialization of the system and the starting of new instances.

Two are the interfaces that will provide information that may be relevant for us, they are interface B and interface X. The first interface will notify us with events related with the life-cycle of a process instance and the initialization of the YAWL system. Interface X will notify us when a process instance is canceled, and a case is enabled or completed.

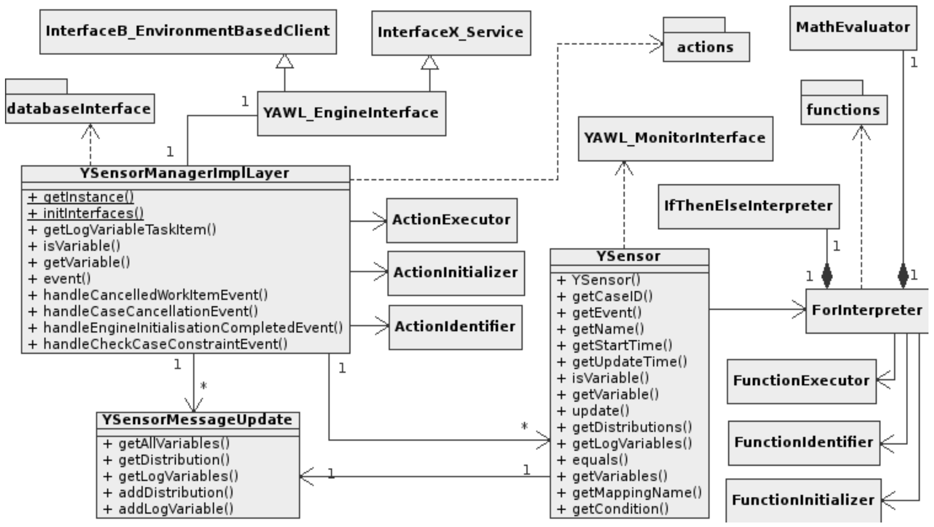


Fig. 2. Monitor Component: Class Diagram

In order to receive the notifications sent by these two interfaces, our monitoring component must extend the java class *InterfaceBWebsideController* and implements the java interface *InterfaceX_Service*. We are only interested in a subset of all notifications that interface B and X will send, for this reason we only need to implements these four methods: i) *handleCancelledWorkItemEvent*; ii) *handleCaseCancellationEvent*; iii) *handleEngineInitialisationCompletedEvent*; and iv) *handleCheckCaseConstraintEvent*. Figure 2 shows the UML [5] class diagram of how our monitoring component is built. In the diagram are only shown the classes required for the realization of the monitoring component and the methods that are relevant for us.

Our monitoring component works through the use of sensors (*YSensor*). Each sensor monitors a specific condition, composed of variables that are the result of functions and actions, and is managed by the sensor manager (*YSensorManager-ImplLayer*). The sensor manager manages the creation of sensors and notifies them when changes occur in a process instance. The sensor manager creates new sensors each time a new process instances is started. Information about the initialization of the system and the starting, completion and cancellation of process instances are retrieved by the sensor manager using the *YAWL_EngineInterface*.

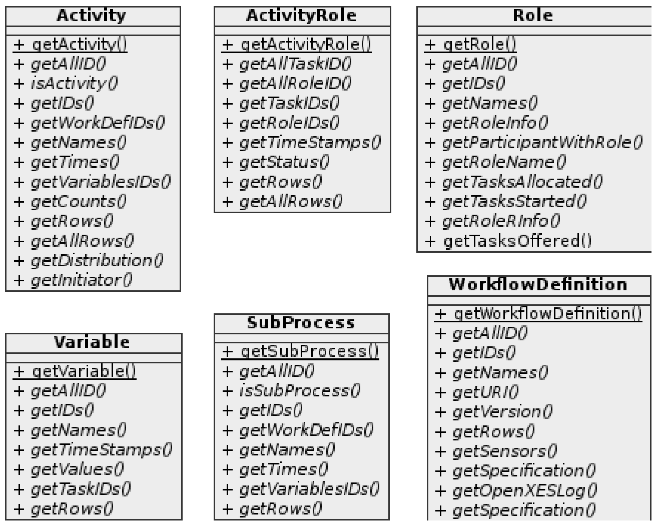


Fig. 3. DatabaseInteface Package Class Diagram

Changes in the process instance are discovered using the classes provided by the package *databaseInterface* (see Figure 3). This package provides an abstraction layer on top of the database which allows queries to be executed through the invocation of java methods. Executing queries gives to the system the possibility of retrieving information from different cases, and then have monitoring conditions defined across cases. In order to know which changes are relevant for a sensor, the sensors manager retrieves from it the list of *LogVariables*. Each of this logVariable is associated with an action that using the *ActionIdentifier* and the *ActionExecutor* is identified and executed, retrieving from the log the information of interest. Each action captures a specific aspect of a work item or a net, the list of all possible actions is shown in figure 4.

Changes in a process instance are then notified to a sensor using messages (*YSensorMessageUpdate*). Every time a sensor receives a message, it checks its monitoring condition using the updated information. The condition is checked using a specific interpreter (*ForInterpreter*) which interprets the languages defined in [1] and verifies if the condition is violated or not. In case the condition is violated the sensor will notify the administrator by sending a notification through the *YAWL_MonitorInterface*. The condition that a sensor can monitor is a boolean expression which may contain nested loops, if-then-else constructs, and algebraic operators.

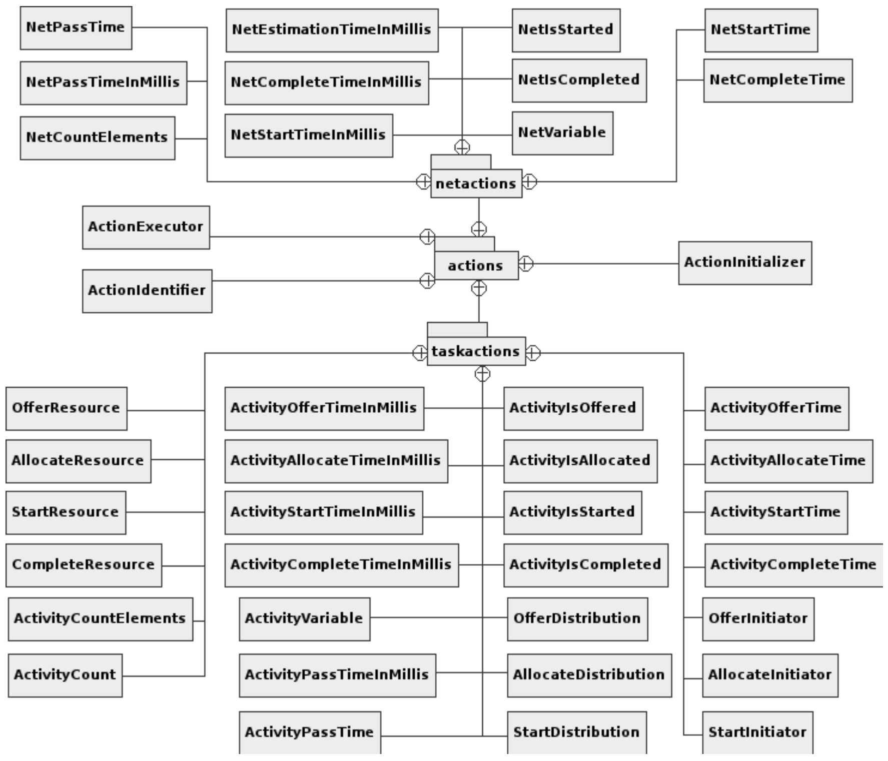


Fig. 4. Actions Package Class Diagram

4 Evaluation and Possible Uses

The architecture here proposed was evaluated in [1]. In the experiment we measured the time required to retrieve the result of the execution of an action. In table 1 we show the result grouped for type of action, e.g. NetStartTime and NetCompleteTime are grouped under the name NetXTime.

The results show that in general a result is produced in few milliseconds (ca. 20ms). Retrieving a net or an activity variable and retrieving information about the distribution set and the initiator of an activity require more time, this because they require the parsing of XML strings since the data are not directly stored in the database.

Actions	**Description**	**time [ms]**
NetIsX	functions checking if a net status has been reached	18.9
NetXTime NetXTimeInMillis	functions returning the time when a net status has been reached	18.8
NetVariable	returns the value of a net variable	432.6
ActivityCount	number of times a task has been completed	19.8
XResource	functions that return the resources associated with a task	20.9
ActivityIsX	functions checking if a task status has been reached	30.5
ActivityXTime ActivityXTimeInMillis	functions returning the time when a task status has been reached	22.3
ActivityVariable	returns the value of a task variable	96.7
XDistribution	functions returning the resources associated with a task by default	243
XInitiator	functions returning the allocation strategy for a resource association	249.6

Table 1. Performance of basic functions.

Risk monitoring is not the only context in which our component can be used. It is a multi-purpose monitoring component that provides the possibility of adding new actions in order to make it usable in other context such as for example cost monitoring, resource monitoring, or time monitoring.

5 Related Work

The idea of monitoring business processes is not new in the area of business process management. Academics explored the possibility of monitoring business processes using Complex Event Processing (CEP) systems [2, 3]. Commercial workflow management systems in general provide integrated monitoring features, e.g. Oracle Business Activity Monitoring (BAM) [6], webMethods Business Events[8], and SAP Sybase [7].

The monitoring component here discussed was used in the approach proposed in [1] for risk monitoring. In this approach business processes are integrated with aspects of risk management, specifically risk monitoring. Risk conditions are composed of two elements, a risk likelihood which monitors the likelihood of a risk to occur and a risk threshold which defined level of risk which the company is willing to accept before detecting the eventuation of a risk.

Gay et al. [2] propose the use of complex event processing for workflow monitoring on Petri nets. They identify six events that can represent the basic activities that a workflow can perform (i.e. Transition activation, Resource allocation, Resource liberation, Advance token, Start workflow, and End workflow). Using these simple events they have created six complex events that represent unwanted situations: i) Lack of resource; ii) Activity delay; iii) Lack of resource delay; iv) Transition delay; v) Workflow delay; vi) Interruption warning. This

[8] http://www.softwareag.com/au/products/wm/events/overview

approach, compare to our approach, does not take in consideration the data prospective. This limitation produces as consequence the possibility of defining conditions that are mainly related to the performance of a process instance.

Finally, Hermosillo et al. [3] propose a framework for dynamic business process adaptation in the context of BPEL processes. In this approach they use the monitoring functionality obtained using a CEP engine to detect conditions that will then trigger an adaptation, i.e. the add or change of a service.

6 Conclusion

In this article we showed how to realize a monitoring component for the YAWL system using the interfaces provided by the system itself. The monitoring is done using sensors, which monitor conditions that can be defined using information across cases.

The main contribution of this work is the identification and documentation of a minimal set of classes required for the realization of a monitoring component for the YAWL system.

The component is realized as a custom YAWL service, in order to guarantee a perfect integration with the YAWL system. The structure of the component also results to be independent from a specific monitoring purpose, consenting its application in different contexts could they be risk monitoring, or cost monitoring.

Finally, the architecture was implemented and tested. The results of the test show that the retrieving of information can be computed efficiently and without requiring additional work engine side.

References

1. R. Conforti, G. Fortino, M. La Rosa, and A. H. M. ter Hofstede. History-aware, real-time risk detection in business processes. In *Proc. of CoopIS*, volume 7044 of *LNCS*. Springer, 2011.
2. P. Gay, A. Pla, B. López, J. Meléndez, and R. Meunier. Service workflow monitoring through complex event processing. In *Proc. of ETFA*, pages 1–4. IEEE, 2010.
3. G. Hermosillo, L. Seinturier, and L. Duchien. Using complex event processing for dynamic business process adaptation. In *Proc. of IEEE SCC*, pages 466–473. IEEE Computer Society, 2010.
4. A. Kronz. Managing of process key performance indicators as part of the aris methodology. In *Corporate Performance Management*, pages 31–44. Springer, 2006.
5. Object Management Group (OMG). *OMG Unified Modeling LanguageTM (OMG UML), Superstructure version 2.4*. Object Management Group (OMG), Jan. 2011.
6. Oracle. *BPEL Process Manager Developer's Guide*, http://download.oracle.com/docs/cd/E15523_01/integration.1111/e10224/bp_sensors.htm. Accessed: Jun. 2011.
7. Sybase. *Sybase CEP Implementation Methodology for Continuous Intelligence*, http://www.sybase.com.au/files/White_Papers/Sybase_CEP_Implementation_Methodology_wp.pdf. Accessed: Jun. 2011.
8. A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment.* Springer, 2010.

Editor 3.0: Redesigning the YAWL User Interface

Michael Adams

Queensland University of Technology, Brisbane, Australia.
mj.adams@qut.edu.au

Abstract. It has long been a concern that the wider uptake of the YAWL environment may have been hindered by the usability issues identified in the current Process Editor. As a consequence, it was decided that the Editor be completely rewritten to address those usability limitations. The result has been the implementation of a new YAWL Process Editor architecture that creates a clear separation between the User Interface component layer and the core processing *back end*, facilitating the redesign of the default user interface. This new architecture also supports the development of multiple User Interface *front ends* for specific contexts that take advantage of the core capabilities the new Editor architecture has to offer.

Keywords: YAWL, Editor, Workflow, User Interface

1 Introduction

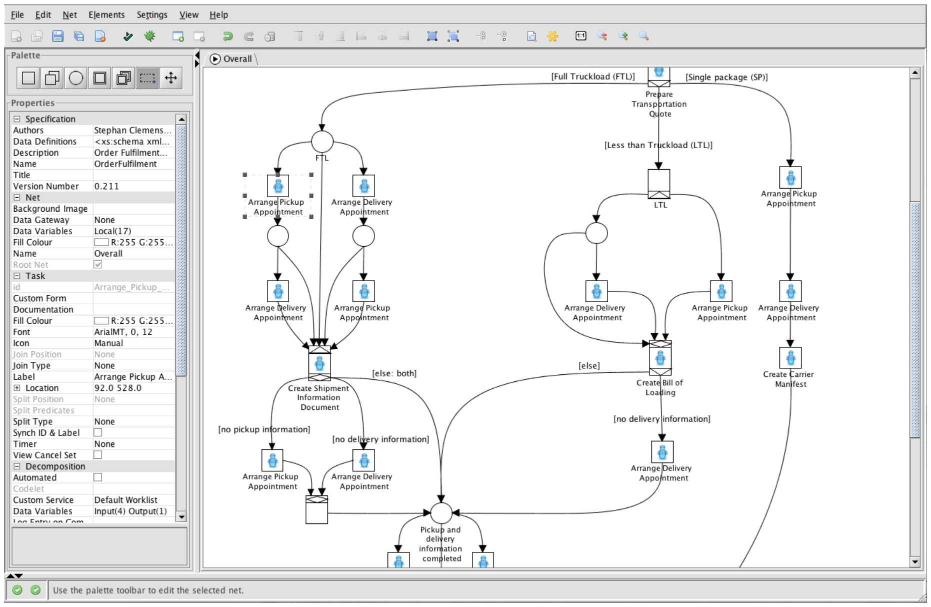
For any software, user acceptance must be of primary concern, and for most users the design and accessibility of the user interface (UI) has a much greater bearing on uptake and continued use than any functionality the software provides. For existing products, redesign of a user interface can lead to substantial improvements in learning time, performance speed, error rates and overall user satisfaction [1].

The first version of the YAWL Editor appeared almost a decade ago as a prototype, with the objective of demonstrating that it was possible to graphically design a process model in the YAWL language and have it transformed into a format that could be interpreted and executed by the YAWL Engine. Originally supporting only the control-flow perspective, many versions have since been released to include support for the data perspective and later the resource perspective, then timers, configuration, extended attributes and many other features. In each case, new functionality was 'grafted' onto the core prototypical architecture. Over time the result was a code base that had become difficult to maintain and extend, and a user interface that revealed both its age and its usability limitations.

To address these shortcomings, the YAWL Editor has been completely redesigned and is being rewritten from the ground up. The overriding driver of the project has been to make the Editor easier to use, and therefore more accessible to a wider audience. This paper outlines the main features of the new Editor.

2 Code Redesign

The original Editor architecture has a flattened package structure that mixes the user interface components with the underlying data manipulation and transportation. This meant that the user interface components were very tightly coupled to their data, which made it difficult to introduce changes to either aspect without disturbing the other as a side-effect. In addition, on opening a specification file, the Editor would first use Engine classes to parse the XML description contained in the file, then use those objects to populate its own 'mirror' classes (the Engine classes contain the code to (un)marshal to and from XML). When saving a model to file, that process was reversed. This policy introduced a large degree of redundant and error prone code, an artefact of its prototypical heritage.


File
Edit
Net
Elements
Settings
View
Help
Palette
Properties
Overall
[Full Truckload (FTL)]
[Single package (SP)]
Prepare Transportation Quote
[Less than Truckload (LTL)]
FTL
LTL
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Arrange Pickup Appointment
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Arrange Pickup Appointment
Arrange Delivery Appointment
[else: both]
[else]
Create Shipment Information Document
Create Bill of Loading
Create Carrier Manifest
[no pickup information]
[no delivery information]
[no delivery information]
Arrange Pickup Appointment
Arrange Delivery Appointment
Arrange Delivery Appointment
Pickup and delivery information completed

⊟ Specification	
Authors	Stephan Clemens...
Data Definitions	<xs:schema xml...
Description	Order Fulfilment...
Name	OrderFulfilment
Title	
Version Number	0.211
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(17)
Fill Colour	☐ R:255 G:255...
Name	Overall
Root Net	✓
⊟ Task	
id	Arrange_Pickup_...
Custom Form	
Documentation	
Fill Colour	☐ R:255 G:255...
Font	ArialMT, 0, 12
Icon	Manual
Join Position	None
Join Type	None
Label	Arrange Pickup A...
⊞ Location	92.0 528.0
Split Position	None
Split Predicates	
Split Type	None
Synch ID & Label	☐
Timer	None
View Cancel Set	☐
⊟ Decomposition	
Automated	☐
Codelet	
Custom Service	Default Worklist
Data Variables	Input(4) Output(1)

Use the palette toolbar to edit the selected net.

Fig. 1. An Overview of the New Editor User Interface

In the redesigned Editor there is complete separation of the UI from the underlying framework. The main advantages are twofold:

1. The completely separated core framework can now be deployed in its own jar file. This allows external developers to create new and multiple user interfaces that interact directly with the core framework through a concise API, and frees them from concerns regarding how to store and manipulate specification control flow, data and resourcing descriptors, and file handling techniques.

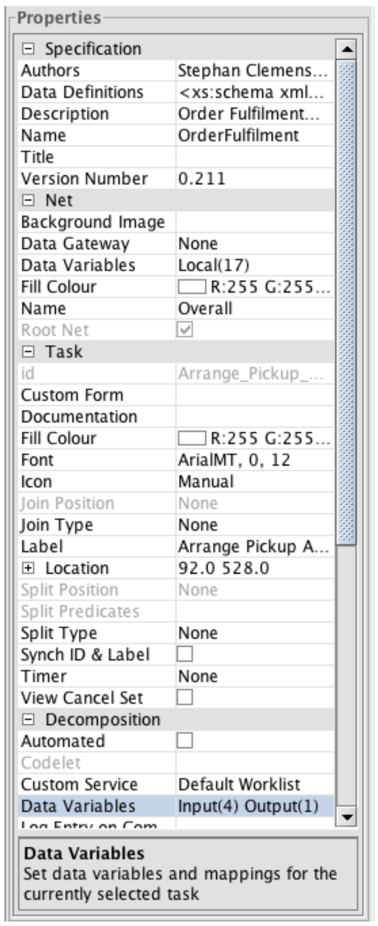
2. The core framework interacts with the YAWL Engine classes only. The negates that need for transformations when opening and saving specification files, which results in much faster file reading and writing times, and removes the scope for introducing errors into the ‘backend’ code when developing user interfaces.

3 Interface Redesign

A large factor in the usability of a software product, and particularly an graphical editor, is the ease with which the properties of an object can be set, updated and maintained. The original Editor used a number of different methods to update properties, found in various places within the interface. For example, specification properties were set via a dialog accessed from the *File* menu, and net properties from the *Net* menu, while task properties were generally accessed via a right-clicked context menu on the task itself. Adding a split or join to a task was accomplished via a box under the palette. Selecting an icon for the task was done in a different location again. Having various locations for these similar capabilities necessarily reduces usability and control.

The new Editor interface places all of the properties, for all aspects of the active process model, within a single expandable *Properties* window to the left of the graphical canvas (Figure 1). The relevant set of properties and their values for the currently selected object or objects are displayed in their appropriate form. Such properties windows are common in today’s editors, and knowing that all properties can be set in one location greatly aids ease of use.

The detailed view of the Properties window in Figure 2 shows that the currently selected object is a task with a decomposition, which therefore allows all properties of the Specification, Net, Task and Decomposition to be viewed simultaneously and easily selected for editing. Values are displayed in a format dependent on their data type. For example, simple values (e.g. Name, Version, Label) are shown as is, boolean types (e.g. Root Net, Sync, Automated)



Properties

⊟ Specification	
Authors	Stephan Clemens...
Data Definitions	<xs:schema xml...
Description	Order Fulfilment...
Name	OrderFulfilment
Title	
Version Number	0.211
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	Local(17)
Fill Colour	□ R:255 G:255...
Name	Overall
Root Net	✓
⊟ Task	
id	Arrange_Pickup_...
Custom Form	
Documentation	
Fill Colour	□ R:255 G:255...
Font	ArialMT, 0, 12
Icon	Manual
Join Position	None
Join Type	None
Label	Arrange Pickup A...
⊞ Location	92.0 528.0
Split Position	None
Split Predicates	
Split Type	None
Synch ID & Label	☐
Timer	None
View Cancel Set	☐
⊟ Decomposition	
Automated	☐
Codelet	
Custom Service	Default Worklist
Data Variables	Input(4) Output(1)
Log Entry on Com	

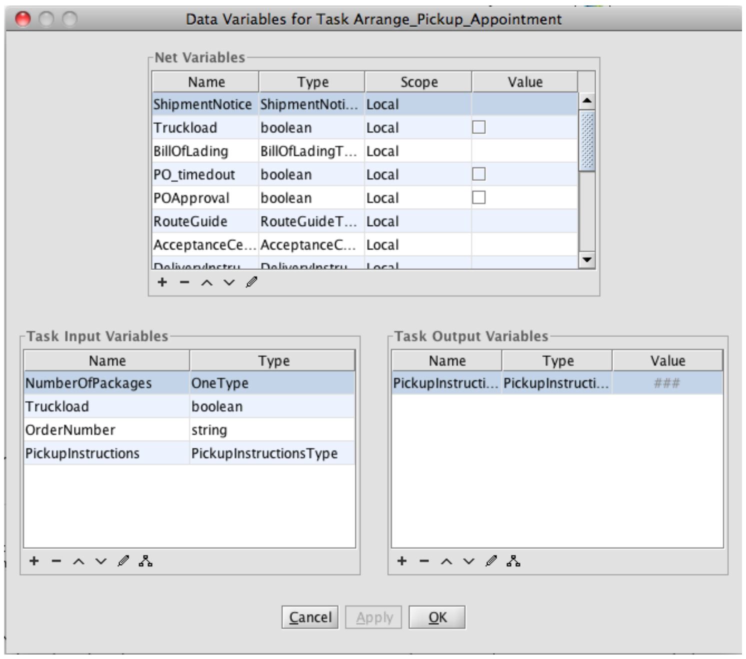
Data Variables
Set data variables and mappings for the currently selected task

Fig. 2. The Properties Window (detail)

as check boxes. More complex values, such as Fill Colour, Font and Data Variables, are shown in a summary form appropriate to their type.

Values for properties such as Data Gateway, Icon, Split and Join Types and Positions, and Decomposition are selectable via a dropdown list. Any property modification that represents a visual change is instantly updated on the canvas, and vice versa. For example, the current location coordinates for the selected task can be changed manually in the Properties window, which will immediately move the selected task to its new location on the canvas. A description of the currently selected property is displayed at the bottom of the window.

Data Perspective Arguably the most significant impediment to usability for the casual user in the original Editor was the requirement of a knowledge of XQuery for all but the most trivial of process designs. Typically, net-level variables were added via the *Net* menu, task-level variables via the *Decomposition Update* dialog (accessed from the right-click context menu) and then mapping input and outputs with XQuery parameters was achieved via the *Update Parameter Mappings* dialog (again form the context menu). Thus, a greater learning curve was needed, with each step providing more scope for error.



Name	Type	Scope	Value
ShipmentNotice	ShipmentNoti…	Local	
Truckload	boolean	Local	☐
BillOfLading	BillOfLadingT…	Local	
PO_timedout	boolean	Local	☐
POApproval	boolean	Local	☐
RouteGuide	RouteGuideT…	Local	
AcceptanceCe…	AcceptanceC…	Local	
DeliveryInstru…	DeliveryInstru…	Local	

Name	Type
NumberOfPackages	OneType
Truckload	boolean
OrderNumber	string
PickupInstructions	PickupInstructionsType

Name	Type	Value
PickupInstructi…	PickupInstructi…	###

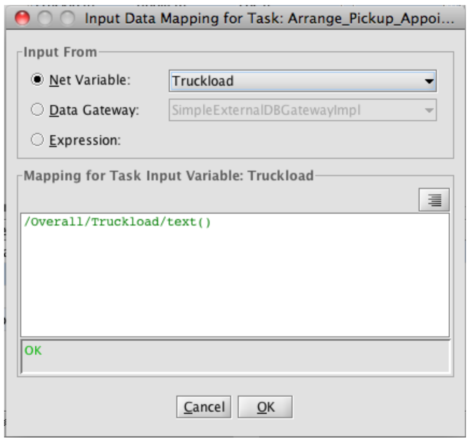
Fig. 3. The New Data Variables Dialog

In the new Editor, great care was taken to simplify this process as much as possible, through the introduction of a consolidated *Data Variables* dialog

(Figure 3). This dialog, invoked from the Task section of the Properties window, provides for the definition of net-level and task-level input and output variables all *within the same simplified dialog*. Each of the three areas in the dialog provides columns for name, type, scope and initial or default value, as appropriate, and each contains it own mini-toolbar, for adding, removing, editing and re-ordering variables. The confusing notion of *Input&Output* scoped variables has been dropped; it is now clear that a task may have a set of inputs and a set of outputs, aligning the Editor's data perspective with that of the Engine.

Notably, a net-level variable can be mapped to a task variable *simply by drag and drop*, and the required XQuery mappings are automatically created and applied. If a mapping other than the default is needed, it is able to be edited at any time with the XQuery dialog (Figure 4), invoked by the mapping button on the mini-toolbar. This dialog has also been completely redesigned to remove the unnecessary complexity of the original Editor's dialog.

An even more complicated data design exercise in the original Editor was in the definition of XQuery parameters for multiple-instance tasks, which served to inhibit the use of multiple-instance tasks for all but the most determined designers. In the new Editor, the XQuery parameters for multiple-instance tasks are also automatically generated and applied via drag and drop in the same dialog as single-instance tasks, making the capabilities offered by multiple-instance tasks available to expert and novice user alike.


Input Data Mapping for Task: Arrange_Pickup_Appoi...

Input From
- [✓ Net Variable:] Truckload
- Data Gateway: SimpleExternalDBGatewayImpl
- Expression:

Mapping for Task Input Variable: Truckload

```
/Overall/Truckload/text()
```

OK

Fig. 4. The XQuery Mapping Dialog

The definition of complex data types using XML Schema can also prove problematic for novice designers. The new Editor will introduce a new, simplified data definition language, loosely based on *Pascal* syntax, allowing for more easily expressed data type definitions that will automatically be transformed to/from XML Schema, thus providing two discrete, user-switchable views of the same type definitions (Listings 1.1 & 1.2).

Resource Perspective The 'Wizard' metaphor used by the original Editor to capture resourcing requirements has proven less than effective in terms of user interface design. In addition to being dated in terms of UI constructs, the metaphor as deployed suffered from three main issues:

1. It was necessary to always begin at the first page and work your way through, even if you wanted only to make a small change on the final page.
2. There was no facility to undo or cancel a change.

3. There is too much information displayed on each screen. While potentially useful for novice users, the information becomes unwanted ‘noise’ as experience is gained. A better approach is to have the information available separately through a ‘Help’ dialog that can be invoked via a button.

Listing 1.1. Type Definition Expressed as XML Schema (example)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="BookOrder">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="price" type="xs:double"/>
      <xs:element name="inStock" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="BookList">
    <xs:sequence>
      <xs:element maxOccurs="unbounded" name="order" type="BookOrder"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Listing 1.2. The Same Type Definition expressed in Simplified DTD Syntax

```
type BookOrder = record {
    title: string;
    price: double;
    inStock: boolean;
}
type BookList = order(1..n) of BookOrder;
```

All three concerns have been addressed in the new Editor through the use of a tabbed dialog that shows the various resourcing properties, any of which can be navigated to directly, with verbose help information removed to separately invoked dialogs. Also, the new Editor will support the *swim-lane* metaphor, allowing resources to be assigned to tasks graphically via their placement in the appropriate lanes, which will aid in the readability of models by all stakeholders.

4 Conclusion

This paper briefly outlines a few of the main features of the new YAWL Editor, however almost every aspect has been updated and improved. It is hoped that the reward for the redesign will be realised through increased deployment of the YAWL environment, and a positive affect on the teachability of YAWL, since novice users will no longer face such a steep learning curve in the use of the Editor before positive results can be achieved.

An alpha version is currently scheduled for limited release in August 2013.

References

1. B. Shneiderman, C. Plaisant, M. Cohen, and S. Jacobs. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 5th edition, 2009.

Web-based Editor for YAWL

Felix Mannhardt

Bonn-Rhein-Sieg University of Applied Sciences, Germany*
felix.mannhardt@smail.wir.h-brs.de

Abstract. This paper presents a web-based editor that offers YAWL editing capabilities and comprehensive support for the XML format of YAWL. The open-source project Signavio Core Components is extended with a graphical user interface (GUI) for parts of the YAWL Language, and an import-/export component that converts between YAWL and the internal format of Signavio Core Components. This conversion, between the web-based editor and the official YAWL Editor, is lossless so both tools may be used together. Compared to the official YAWL Editor, the web-based editor is missing some features, but could still facilitate the usage of the YAWL system in use cases that are not supported by a desktop application.

1 Introduction

Software that is available only as a traditional desktop application is unlikely to receive substantial adoption nowadays, in a time when these applications are more and more replaced by web applications, and mobile apps. Major companies from the industry even work on operating systems solely based on a web browser (e.g. Google Inc. with Chrome OS and Mozilla Corp. with Firefox OS). The YAWL System provides a complete tool-chain for Workflow Management: the YAWL Editor for process design, the YAWL Control Center for configuration and management of organizational resource, and the YAWL Engine for process execution. Excluding the YAWL Editor all components of the YAWL system are web applications designed with a service-oriented architecture. The YAWL Editor, instead, is only available as a rather monolithic desktop application, which is built with Java using Swing and the JGraph library. Additionally, new features cannot be added to the YAWL Editor in a pluggable way, due to the software architecture. However, it is possible to create a new YAWL editor that mitigates these shortcomings and retains compatibility to the official YAWL Editor by using the well-defined YAWL XML format.

This new editor for YAWL is based on Signavio Core Components (SCC) and called web-based YAWL Editor hereafter. SCC is an open-source project started by Signavio GmbH that is based on the discontinued Oryx project [2] and the commercial, web-based process modeling tool Signavio Editor. The SCC editor provides basic process modeling functionality, a file-based repository and is

* Parts of the work done while visiting Queensland University of Technology, Australia

extensible with new process modeling languages through plug-ins. Unfortunately, also the SCC project was discontinued in 2011, shortly after the first release. Nevertheless, the source-code is still available[1], and efforts are made to continue the development as part of the Apromore[2] project [7]. The Apromore project also includes the YAWL editing capabilities that are presented in this paper. Most of the SCC editor is implemented with JavaScript that manipulates Scalable Vector Graphics (SVG) to show and edit process models in the browser window. For each supported process modeling language of SCC a so called Stencil Set [2] needs to be defined.

The new web-based YAWL Editor consists of SCC together with three added components: the YAWL Stencil Set, a client-side component and a server-side component. The YAWL Stencil Set includes SVG graphics of all YAWL elements (e.g. Tasks, Conditions, Edges), all their properties (e.g. description, routing conditions, task variables) and basic rules regarding their interconnection. Additional rules that cannot be expressed within the Stencil Set are added through the client-side component with JavaScript. The server-side component, which is realized with Java, is responsible to convert between the YAWL XML format and the JSON format of SCC. This conversion is lossless, therefore workflows created with the original YAWL Editor can be imported, modified with the web-based YAWL Editor and exported back to the original YAWL Editor.

In Section 2 the functionality of the new web-based YAWL Editor is presented, then limitations of the current implementations are discussed in Section 3, and before concluding this paper related work is described in Section 4.

2 Web-based YAWL Editor

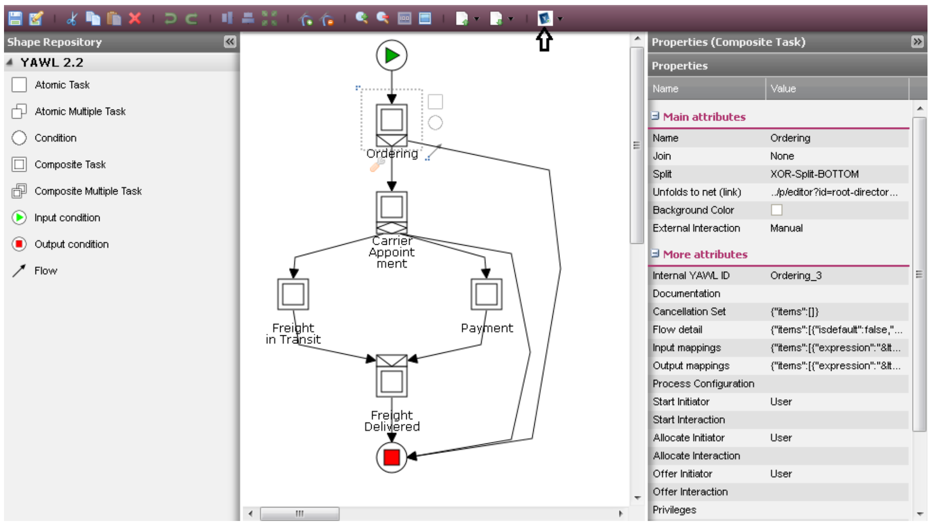
Figure 1 shows the web-based YAWL Editor displaying the top level net of the *Orderfulfilment* process [5] that was imported using the XML file that is distributed as an example together with YAWL. On the left side of the browser window, there is the *Shape Repository* containing all available tasks (Atomic, Composite, Atomic Multiple Task, Composite Multiple Task) and conditions (Normal, Input, Output) of YAWL. A new instance of a shape can be added to the diagram area in the middle either by using drag and drop from the *Shape Repository* or by clicking on one of the context menu icons showing the follow-up element. The context menu is visible in Figure 1 next to the focussed *Ordering* task. The web-based YAWL Editor enforces the connection rules of YAWL, for example that tasks without attached split gateway must not have more than one outgoing edges or that an Input Condition must not have an incoming edge. As a result without the XOR-split on the *Ordering* task, there would no follow-up task or follow-up condition be displayed in the context menu.

On the right side of the browser window, all properties of the currently selected node can be edited. Please note that the original identifier of the YAWL element is preserved in the property *Internal YAWL ID*, and will be re-used upon

[1] `http://code.google.com/p/signavio-core-components/`

[2] Apromore is a process model repository, available at: `http://apromore.qut.edu.au/`

export of the workflow back to the YAWL XML format. The properties *Split* and *Join* determine whether a task has split or join routing behavior. These properties provide selectable options for all combinations of routing behavior (AND, XOR, OR) and placement of the visible counterpart on the task node (Top, Left, Right, Bottom).

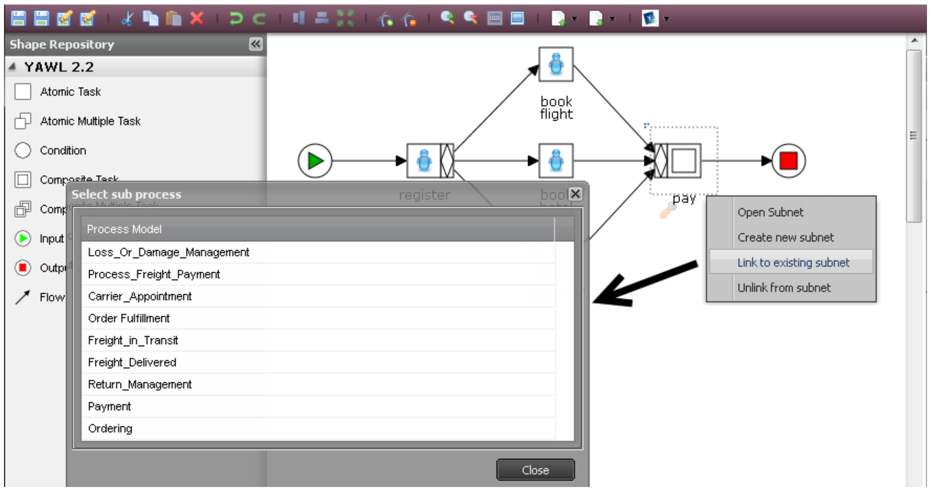


Name	Value
Main attributes	
Name	Ordering
Join	None
Split	XOR-Split-BOTTOM
Unfolds to net (link)	../p/editor?id=root-director...
Background Color	☐
External Interaction	Manual
More attributes	
Internal YAWL ID	Ordering_3
Documentation	
Cancellation Set	{"items":[]}
Flow detail	{"items":[{"isdefault":false,"...
Input mappings	{"items":[{"expression":"&lt...
Output mappings	{"items":[{"expression":"&lt...
Process Configuration	
Start Initiator	User
Start Interaction	
Allocate Initiator	User
Allocate Interaction	
Offer Initiator	User
Offer Interaction	
Privileges	

Fig. 1. Web-based YAWL Editor showing the Orderfulfilment example process

In contrast to the original YAWL Editor, which stores subnets of a workflow embedded in the workflow specification, upon import of the *Orderfulfilment* process all subnets have been stored separately in the repository. All composite tasks, like the task *Ordering* in Figure 1, just store a reference to the target subnet, as visible in the property *Unfolds to net (link)*. The reference is stored in form of an URI that points to the subnet in the SCC repository. When using another repository (e.g. Apromore), then this reference will be an URI that points to the process model in the external repository. This way, the same subnet can be used in multiple YAWL workflow specifications that are stored in the web-based YAWL Editor. A subnet that is created in the web-based YAWL Editor may also be exported as the rootnet of a YAWL workflow specification. This allows for more flexibility in the creation of workflows, for example the *Payment* subnet could be exported as a stand-alone workflow. Of course, during generation of the YAWL XML format all referenced subnets are embedded again in order to generate valid YAWL XML.

The features *Highlight Cancellation Region* and *Unfold to Subnet* have been implemented to show that the web-based YAWL Editor can be extended with more advanced features of the official YAWL Editor. *Highlight Cancellation Region* emphasizes the nodes and edges that are in the Cancellation Region of a selected task. YAWL specific features, like *Highlight Cancellation Region*, can be activated under the YAWL symbol in the menu as indicated by the arrow shown in Figure 1. The second feature, *Unfold to Subnet*, supports the user with the handling of subnets. Upon selecting a Composite Task a context menu appears, as shown next to the task *pay* in Figure 2. The user can either open the already linked subnet, create a new subnet, link the task to an existing subnet that was created before, or unlink the currently assigned subnet. The selection of an existing subnet is supported by showing a list of all available YAWL workflows in the repository of the web-based YAWL Editor.



Select sub process

Process Model
Loss_Or_Damage_Management
Process_Freight_Payment
Carrier_Appointment
Order Fulfillment
Freight_in_Transit
Freight_Delivered
Return_Management
Payment
Ordering

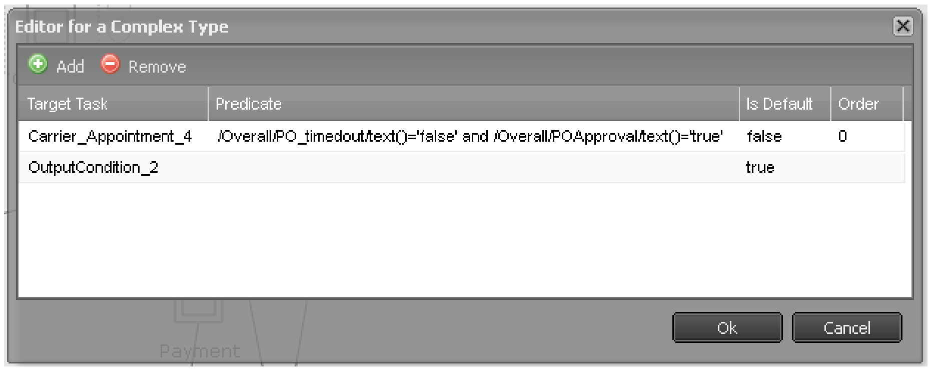
Fig. 2. Handling of YAWL subnets within SCC

The source code of the web-based YAWL Editor is currently available as part of the Apromore Editor: `http://code.google.com/p/apromore/`. Additionally, more information about future development and a demo version is available on the website: `http://www.yaug.org/WebYAWLEditor`.

3 Limitations

The current implementation of the web-based YAWL Editor provides fewer features than the official YAWL Editor. The most significant limitations are in the area of the resource and data perspective of YAWL. At this stage of the development, the web-based YAWL Editor does not connect to the Resource Service of YAWL to look up organizational resources (e.g. Roles, Participants, Secondary Resources), but the information about the resource perspective of an imported YAWL workflow is preserved using fragments of the original YAWL XML format. In a future version, the web-based YAWL Editor could connect to the Interface

R of YAWL to look up organizational resources and manipulate the stored XML fragment according to the selection. Regarding the data perspective, there is some support through structured editors, but the web-based YAWL Editor is missing *look up* and *validation* features. Figure 3 shows the structured editor for the routing predicates. New routing predicates can be added in a structured way, but the editor in Figure 3 still provides no means of automatically picking the target task, or checking the XSD expression for validity. The editor for task variables has similar limitations.



Target Task	Predicate	Is Default	Order
Carrier_Appointment_4	/Overall/PO_timedout/text()='false' and /Overall/POApproval/text()='true'	false	0
OutputCondition_2		true	

Fig. 3. Property editor for the routing predicates

Furthermore, there are still some minor limitations regarding the visual appearance of the YAWL workflows in the web-based Editor. SCC does not support bended edges with bezier curves or splines and it is missing support for exact positioning of labels. As a result, in the web-based YAWL Editor all labels are positioned directly under the corresponding YAWL elements and edges using bezier curves or splines are shown as straight lines.

4 Related Work

There have already been a few attempts bringing support for new process modeling languages to the Oryx Editor (predecessor of SCC) similar to this approach. For example, support for Coloured Petri Nets and UML class diagrams[3]. The web-based jBPM Designer[4], which is based on the Oryx Editor, is developed as part of the jBPM project. As it is solely dedicated to BPMN 2.0 it was not considered as starting point for the development of the web-based YAWL Editor. Regarding to YAWL and SCC/Oryx the implementation of an automatic export of BPMN models to YAWL in the Oryx Editor[5] using the approach described

[3] `http://bpt.hpi.uni-potsdam.de/Oryx/OryxScreencasts`
[4] `http://www.jboss.org/jbpm/components/designer`
[5] `http://bpt.hpi.uni-potsdam.de/Oryx/BpmnToYawl`

in [1] could be seen as related. The approach is limited to a subset of BPMN 1.2 and can not be used to edit existing YAWL workflows on the web. There is a YAWL Android Client[6] that supports the execution of a YAWL workflow on mobile devices, but this project does not provide support for the design time of a workflow. To the best of my knowledge there is currently no other web-based YAWL Editor available.

5 Conclusion

With the web-based YAWL Editor, it is possible to create new, and modify existing YAWL workflow specifications using solely a web browser. The web-based YAWL Editor is fully compatible with the official YAWL Editor. It has been tested with the publicly available workflow specifications, which are distributed together with YAWL. The web-based YAWL Editor can be offered as a service without the need of a local installation, and it can be integrated in existing web applications. In comparison to the original YAWL Editor some use cases are simplified, like the usage of YAWL in a classroom during an exercise, and new use cases are made possible, like the interactive presentation of YAWL workflow specifications on a web site.

The web-based YAWL Editor is, furthermore, already integrated into the process model repository Apromore. In the future, using Apromore and the web-based YAWL Editor features like, for example, version control [4] and process merging [6] may be available to the YAWL community. Still, there are some shortcomings with the current implementation of the web-based YAWL Editor that need to be mitigated. Most of all, better support for the resource and data perspective of YAWL is needed. Overall with some additional development effort, the web-based YAWL Editor could evolve to a beneficial addition to the YAWL ecosystem.

References

1. Gero Decker, Remco M. Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Transforming bpmn diagrams into yawl nets. In Dumas et al. [3], pages 386–389.
2. Gero Decker, Hagen Overdick, and Mathias Weske. Oryx - an open modeling platform for the bpm community. In Dumas et al. [3], pages 382–385.
3. Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors. *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*. Springer, 2008.
4. Chathura C. Ekanayake, Marcello La Rosa, Arthur H. M. ter Hofstede, and Marie-Christine Fauvet. Fragment-based version management for repositories of business process models. In Robert Meersman, Tharam S. Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng Chin Ooi, Ernesto Damiani, Douglas C.

[6] `http://www.yaug.org/tryOutYAWLAndroidClient`

Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh K. Mohania, editors, *OTM Conferences (1)*, volume 7044 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2011.

5. Marcello La Rosa, Stephan Clemens, Arthur Hofstede, and Nick Russell. Appendix A The Order Fulfillment Process Model. In Arthur H M ter Hofstede, Wil M P van der Aalst, Michael Adams, and Nick Russell, editors, *Modern Business Process Automation*, pages 599–616. Springer Berlin Heidelberg, 2010.
6. Marcello La Rosa, Marlon Dumas, Reina Uba, and Remco M. Dijkman. Merging business process models. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *OTM Conferences (1)*, volume 6426 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2010.
7. Marcello La Rosa, Hajo A. Reijers, Wil M.P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano Garca-Bauelos. Apromore: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029 – 7040, 2011.

Usability Extensions for the Worklet Service

Michael Adams

Queensland University of Technology, Brisbane, Australia.
mj.adams@qut.edu.au

Abstract. The YAWL Worklet Service is an effective approach to facilitating dynamic flexibility and exception handling in workflow processes. Recent additions to the Service extend its capabilities through a programming interface that provides easier access to rules storage and evaluation, and an event server that notifies listening servers and applications when exceptions are detected, which together serve enhance the functionality and accessibility of the Service's features and expand its usability to new potential domains.

Keywords: YAWL, Worklet, Exception Handling, Flexibility

1 Introduction

Since its introduction, the YAWL Worklet Service has proven to be an effective approach to facilitating dynamic flexibility and exception handling in otherwise static process instances [2, 3]. The Service provides an extensible *repertoire* of self-contained selection and exception-handling processes, coupled with an extensible *ripple-down* rules set. At runtime, the Service provides functionality that allows for the substitution of a task with a sub-process, dynamically selected from its repertoire, depending on the rules set and the context of the particular work instance. In addition, exception-handling functionality dynamically detects and mitigates exceptions as they occur, using the same repertoire and rule set framework[1].

Now, recent additions to the service provide new methods of interaction that extend its capabilities through the introduction of an *application programming interface* (API) that allows external custom services and applications to interact with it directly, and an *event server* that allows listener services to be notified when a worklet is selected or an exception raised during a process execution. Of particular interest are the ways in which the service can now be more easily introduced into learning situations.

2 The Worklet Gateway Client API

The `WorkletGatewayClient` is a client side class that can be used by custom services [5] to interact with the Worklet Service via an API. It provides two main functionalities:

[1] See [1] for more details on the design and implementation of the Worklet Service

– creating, accessing, updating and evaluating Ripple Down Rules (RDR) directly, circumventing the need to work with the Windows-based Rules Editor application (see Section 3); and
– adding and removing worklet event listeners (see Section 4).

An instance of `WorkletGatewayClient` can be used by custom services and applications to first create a session with the Worklet Service, then use the returned *session handle* in subsequent API method calls. Any service or application registered with the YAWL Engine is trusted by the Worklet Service, and so the same credentials can used to establish the connection (see Listing 1.4 for an example of how to connect and use the API).

3 Working with Ripple Down Rules

Since the primary aim of the Worklet Service is to support processes enacted by the YAWL engine, each rule set is typically uniquely identified for a particular specification by its `YSpecificationID` object. However, to generalise RDR support for other custom services and applications, rule sets can now also be uniquely identified by any unique string value. Thus, for each client method there are variations to support the fact that either a specification identifier or a generalised name may be used to identify rules.

Listing 1.1. Examples of using the Worklet Gateway Client 'get' methods

```
public void example throws IOException {
    YSpecificationID specID = new YSpecificationID(null, "0.1",
            "someSpecification");
    String processName = "someProcess";
    RdrMarshal marshal = new RdrMarshal();
    WorkletGatewayClient client = new WorkletGatewayClient();
    String handle = client.connect("admin", "YAWL");

    // get a Rule Set
    String ruleSetXML = client.getRdrSet(specID, handle);
    if (! successful(ruleSetXML)) throw new IOException(ruleSetXML);
    RdrSet ruleSet = marshal.unmarshalSet(ruleSetXML);

    // get a Rule Tree
    String ruleTreeXML = client.getRdrTree(processName, null,
            RuleType.CasePreconstraint, handle);
    if (! successful(ruleTreeXML)) throw new IOException(ruleTreeXML);
    RdrTree ruleTree1 = marshal.unmarshalTree(ruleTreeXML);
    ruleTreeXML = client.getRdrTree(specID, "Treat", RuleType.ItemSelection,
            handle);
    if (! successful(ruleTreeXML)) throw new IOException(ruleTreeXML);
    RdrTree ruleTree2 = marshal.unmarshalTree(ruleTreeXML);

    // get a Rule Node
    String ruleNodeXML = client.getNode(specID, null,
            RuleType.CasePreconstraint, 1, handle);
    if (! successful(ruleNodeXML)) throw new IOException(ruleNodeXML);
    RdrNode node1 = marshal.unmarshalNode(ruleNodeXML);
    ruleNodeXML = client.getNode(processName, "Archive",
            RuleType.ItemSelection, 4, handle);
    if (! successful(ruleNodeXML)) throw new IOException(ruleNodeXML);
    RdrNode node2 = marshal.unmarshalNode(ruleNodeXML);
}
```

Get Methods Firstly, there are a number of *get* methods, which can be used to retrieve an individual rule node, a rule tree (a set of connected nodes for a particular rule type and task id), or an entire rule set (the set of rule trees for a particular specification or process). Each of the *get* methods, and in fact all of the client methods, return a String value representing a successful result or an appropriate error message. A successful result for a *get* method contains an XML String representation of the object retrieved, which can be unmarshalled into the appropriate object using the `RdrMarshal` class, as shown in the examples in Listing 1.1.

Evaluation Methods The rule sets maintained by the Worklet Service can be evaluated against a given data set via the client API at any time. Like the *get* methods, there are variations that accept a specification identifier or a generalised name, and if evaluating an item-level rule tree, a task id. There are three possible results when a rule set is evaluated:

1. No rule tree exists for the given specification id or process name, and/or task id, and rule type (as required). An error message to that effect is returned.
2. A rule tree exists for the given parameters, but none of its nodes' conditions were satisfied. An error message to that effect is returned.
3. A rule tree exists for the given parameters, and the conditions of at least one its nodes were satisfied. An `RdrConclusion` object is returned.

For a rule set associated with a YAWL specification, an `RdrConclusion` object represents a set of *primitives*, each comprising an *action* and a *target*, and provides methods for iterating through the primitives set. Of course, if the Worklet Service is being used merely as an RDR store/evaluator, and thus there is no intention to have the rules evaluated by the Worklet Service in relation to a YAWL process instance, then the conclusion of a rule node may contain any data of relevance to an external service or application. For example, the conclusion of an *ItemSelection* rule node contains one primitive, with an action of 'select' and a target naming the worklet to select. The conclusion of an exception rule type may contain any number of primitives (see the Worklet Service chapter of the YAWL User Manual for more information). An example of an *evaluate* method call is shown in Listing 1.2.

Listing 1.2. Example using the Worklet Gateway Client 'evaluate' method

```
public void example throws IOException {
    YSpecificationID specID = new YSpecificationID(null, "0.1",
            "someSpecification");
    RdrMarshal marshal = new RdrMarshal();
    WorkletGatewayClient client = new WorkletGatewayClient();
    String handle = client.connect("admin", "YAWL");

    // evaluate a Rule Type against a data set
    String dataStr = "<data><Age>40</Age><Fracture>true</Fracture></data>";
    Element data = JDOMUtil.stringToElement(dataStr);
    String conclusionXML = client.evaluate(specID, "Treat", data,
            RuleType.ItemSelection, handle);
    if (! successful(conclusionXML)) throw new IOException(conclusionXML);
    RdrConclusion conclusion = marshal.unmarshalConclusion(conclusionXML);
}
```

Adding a Rule Node RDR are structured in such a way that rule nodes can be added at any time, but never deleted, since deleting a node would 'break' the tree's internal structure. The client API provides a method to add a rule node to a rule set for a specification or process, with similar variations to those mentioned above for the different rule types. An example of adding a node is shown in Listing 1.3.

Listing 1.3. Example using the Worklet Gateway Client to add a new Rule Node

```
 1  public void example throws IOException {
 2      YSpecificationID specID = new YSpecificationID(null, "0.1",
 3              "someSpecification");
 4      RdrMarshal marshal = new RdrMarshal();
 5      WorkletGatewayClient client = new WorkletGatewayClient();
 6      String handle = client.connect("admin", "YAWL");
 7
 8      // add a Rule Node
 9      String cornerStr = "<data><Age>40</Age><Fracture>true</Fracture></data>";
10      Element eCornerstone = JDOMUtil.stringToElement(cornerStr);
11      RdrConclusion conclusion = new RdrConclusion();
12      conclusion.addPrimitive(ExletAction.Suspend, ExletTarget.Case);
13      conclusion.addCompensationPrimitive("myWorklet");
14      conclusion.addPrimitive(ExletAction.Continue, ExletTarget.Case);
15      RdrNode node = new RdrNode("Age=>75", conclusion, eCornerstone);
16      String newNodeXML = client.addNode(specID, "Treat",
17              RuleType.ItemPreConstraint, node, handle);
18      if (! successful(newNodeXML)) throw new IOException(newNodeXML);
19      node = marshal.unmarshalNode(newNodeXML);
20  }
```

When adding a rule node, values for *condition*, *conclusion* and *cornerstone* are all that is required, as shown in line 15 of Listing 1.3. The Worklet Service will determine the appropriate location in its rule tree for the new node[2]. The *cornerstone* data is particularly important when adding a node to an existing tree, as it is used to determine the proper location of the node. If no rule set currently exists for the specification id or process name given, then one is created. Further, if no rule tree currently exists for the specification id or process name, and rule type (and task id if required) given, then one is created, and the node is added as the first node in the tree (after the root node).

User-defined Functions The condition defined for each rule node is an expression that must evaluate to a final boolean (true/false) value. Expressions can be formed using the usual numeric, comparison and logical operators, and operands consisting of literal values, and/or the values referenced by the name of case and/or workitem data variables. Alternately, an XQuery expression may be used, which is useful if you need to query case or workitem data stored as a complex data type.

In addition, conditional expressions support the use of *user-defined functions* that are defined to evaluate the available data in a particular way, and in turn may be inserted into complex, conjunctive expressions. These functions can pass arguments consisting of literal values, and case or workitem data variable names,

[2] Interested readers can find details of how the Service determines the location of a new rule within the tree on pages 59–60 of [1]

which will be substituted with their actual runtime values before being passed to the function[3].

For work item level rules, a special data structure, called `this`, can be passed as an argument to a user-defined function (e.g. `myFunc(this)`). The structure is assigned an XML string representing the complete work item record, so that specification and task descriptors, and state and timing values may be accessed within the user-defined function.

4 Worklet Event Listeners

Each time the Worklet Service selects a worklet for a work item, or raises an exception for a work item or case, details of the event are announced to registered listener classes. Listeners can use these event announcements to perform additional processing as required, and make it possible to create *exception handling service chains*.

An abstract class called `WorkletEventListener` is the base class for all worklet listeners. This class is a `HttpServlet` that handles all of the mechanics involved in receiving notifications from the Worklet Service and transforming them into five abstract method calls that are to be implemented by extending classes:

- **caseLevelExceptionEvent** is called each time a case level exception is raised, and passes descriptors for case-level data, the type of exception raised, and the `RdrNode` that evaluated to true and so triggered the raising of the exception.
- **itemLevelExceptionEvent** is called each time a item level exception is raised, and passes descriptors as above in addition to the work item record that was the 'subject' of the exception being raised.
- **selectionEvent** is called each time a worklet selection occurs, i.e. when a worklet is substituted for a work item, passing descriptors for case-level data, the work item that the worklet was selected for, and the rule node that evaluated to true and so triggered the selection.
- **constraintSuccessEvent** is called each time a case or work item that has pre- or post-constraint rules defined has had those rules evaluated and the case or work item passes the constraints (i.e. no rules were satisfied and so no exception was raised).
- **shutdown** is called when the Worklet Service is shutting down, and allows implementers to take any necessary action.

Once the listener class implementation is complete, the extra servlet definition section needs to be added to the web.xml of the implementing (listener) service, and then at runtime the listener is to be registered with the Worklet Service, as shown in Listing 1.4.

[3] See the YAWL User and Technical Manuals for details on the definition of user-defined functions

Listing 1.4. Registering a listener with the Worklet Service

```
public boolean registerWorkletListener() {
    WorkletGatewayClient client = new WorkletGatewayClient();
    try {
        String url = "http://localhost:8080/myService/workletlistener";
        String handle = client.connect("admin", "YAWL");
        return successful(client.addListener(url, handle));
    }
    catch (IOException ioe) {
        log.error("Failed␣to␣␣register␣listener:␣" + ioe.getMessage());
    }
    return false;
}
```

5 Conclusion

The additions to the Worklet Service outlined above allow it to be more easily integrated into new and existing projects. Since rules can now be easily added and evaluated directly via the API, there is no longer the requirement to use the Windows-based Rules Editor for that purpose, and therefore it is no longer necessary to undertake the learning curve associated with using the Rules Editor before the benefits provided by the Service can be realised. In addition, the ability to define and evaluate rules sets for environments external to YAWL process executions broadens the application of the Service's functionality to much wider domains. Finally, the ability for external applications and services to be notified when an exception or worklet selection has occurred during a process execution, and when a case or work item has passed a constraint evaluation, opens the Service up to new areas of functional support, particularly to the facilitation of exception handling service chains.

An example of a novel use of these extensions can be seen in the development of *blended workflow*, which integrates two specifications for the same workflow process, one structured (YAWL) and one ad-hoc (goal-based), allowing users to switch between two views of the same process instance [4].

References

1. M. Adams. *Facilitating Dynamic Flexibility and Exception Handling for Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2007.
2. M. Adams, A.H.M. ter Hofstede, W.M.P. van der Aalst, and D. Edmond. Dynamic, extensible and context-aware exception handling for workflows. In R. Meersman and Z. Tari, editors, *Proceedings of the 15th International Conference on Cooperative Information Systems (CoopIS'07)*, volume 4803 of *Lecture Notes in Computer Science*, pages 95–112, Vilamoura, Portugal, November 2007. Springer.
3. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In R. Meersman and Z. Tari et al., editors, *Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS'06)*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308, Montpellier, France, November 2006. Springer.

4. D. Passinhas, M. Adams, B. O. Pinto, R. Costa, A. Rito Silva, and A.H.M. ter Hofstedet. Supporting blended workflows. In N. Lohmann and S. Moser, editors, *Proceedings of the Demonstration Track of the 10th International Conference on Business Process Management (BPM 2012)*, volume 940 of *CEUR Workshop Proceedings*, pages 23–28, Tallinn, Estonia, September 2012. CEUR-WS.org.
5. A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment.* Springer, 2010.