

Reconciling Object-oriented and Process-oriented Approaches to Information Systems Engineering

Guy Redding¹, Marlon Dumas¹, Arthur H.M. ter Hofstede¹ and Adrian Iordachescu²

¹ Queensland University of Technology, Brisbane, Australia
{g.redding, m.dumas, a.terhofstede}@qut.edu.au

² Shared Web Services Pty Ltd, Sydney, Australia
adrian@sws.com.au

Abstract. Object-oriented modelling is an established approach to document the structure and behaviour of information systems. In an object model, a system is captured in terms of object types and associations, state machines, collaboration diagrams, etc. Process modeling on the other hand, provides a different approach whereby behaviour is captured in terms of tasks, flow dependencies, resources, etc. These two approaches have their relative strengths and weaknesses. In object models, behaviour is split across object types, whereas in process models, behaviour is seen holistically along chains of logically related tasks. Also, object models and process models lend themselves to different styles of implementation. There is an opportunity to leverage the relative advantages of object models and process models by creating integrated meta-models and transformations, so that modellers can switch between these views. In this paper we define a transformation from a meta-model for object behavior modeling to a meta-model for process modeling. The transformation relies on the identification of elementary causal relations in the object model. These relations are encoded in a heuristics net from which a process model is derived.

Key words: Process model, object model, model transformation

1 Introduction

The behaviour of an Information System can be modelled using a variety of different design approaches. Two well-known modelling approaches are the object-oriented (OO) and process-oriented approaches [1]. Each approach is a different perspective that has its own way of thinking. For example, modelling an Information System in terms of the objects involved in the system results in a series of object-oriented models often prepared using the UML notation.

Object-orientation is a programming paradigm that can be used in domains other than programming, such as the design of Information Systems. The UML

modelling notation [2] captures the behaviour of *objects* in an Information System. OO emphasises the definition of classes in a system, the data structures and operations of those classes and the identification of message exchanges between classes. The OO approach groups related data and operations into the same class, allowing modularisation and encapsulation. Purported advantages of this paradigm include improved reuse and maintainability.

A process model captures the *control flow behaviour* of a process. The practise of process modelling provides benefits such as task structuring in the form of workflow and the ability to simulate processes prior to their execution. BPMN [3], BPEL [4] and YAWL [5] are all examples of notations that model the behaviour of a process-oriented application. The advantages of process modelling in Information Systems are well known which has given rise to a class of Information System known as Process Aware Information System (PAIS) [6].

There is a need to reconcile object-oriented and process-oriented development approaches to Information Systems engineering to gain the advantages from either viewpoint. Either modelling approach captures information in profoundly different models, thus information captured in one approach may be missing from the alternative approach. For example, a class in an OO model can be modelled to contain references to tasks (i.e. in a sequence chart), but objects are predominately state-based and do not explicitly define tasks or the control flow relations between them. Likewise, a process-oriented model contains references to states (i.e. the *work item lifecycle* [7] is a state machine), but a process-centric model is predominately task-based with no references to classes. Because the foundations of these two approaches are different, neither approach directly contains the information required to reconcile with the other. In this paper we investigate the possibility of performing reconciliation between the deliverables of the design phase of the object-oriented development approach (OODA) and process-oriented development approach (PODA) as shown in Figure 1.

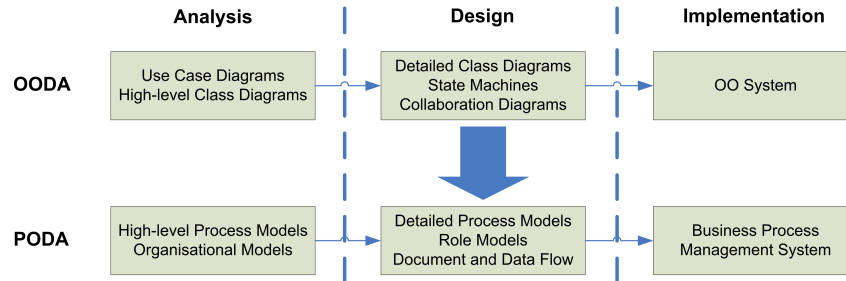


Fig. 1. Reconciling Object-oriented and Process-oriented Development Approaches

This paper is organized as follows. Section 2 introduces an example to illustrate the problem. In Section 3 we define a meta-model for a process aware object-oriented information system. Section 4 introduces an algorithm to reconcile an object-oriented and process-oriented design and apply it to our example.

Section 5 presents some related literature and Section 6 contains the conclusions from this work.

2 An Illustrative Example

In this section we present an example of a process that we have used as a test scenario for our approach and implementation. Using object-oriented modelling analysis the problem domain is presented as a class diagram to capture the classes and their high-level relationships as shown in Figure 2. Each class may have one or more state machines, but for space reasons we have only presented the classes. Our example is a process for inspection and maintenance of heavy equipment such as open mine excavators and shipping container cranes. Such equipment is subjected to inspections at regular intervals when a number of issues requiring maintenance may be raised with the equipment. Depending on the severity of an issue and the criticality of the equipment some issues will be determined to be resolved with more urgency than others.

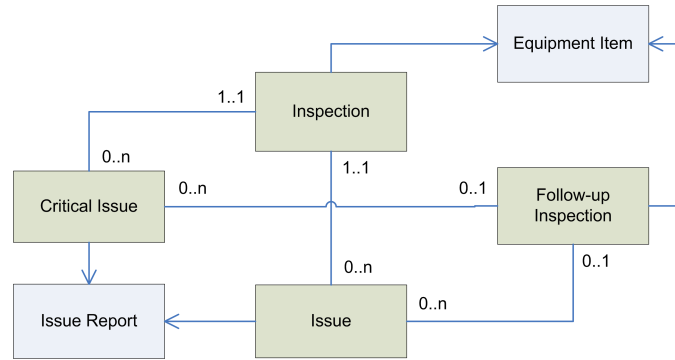


Fig. 2. Illustrative Example - Maintenance Process

An *inspection* for a particular piece of equipment may uncover zero or more *issues* to be resolved. These are added to the set of existing *issues* for that piece of equipment so that the main *inspection* can only be completed after all *issues* have been resolved and completed. A *critical issue* may also be detected during an *inspection* or *follow-up inspection*. These *critical issues* are identified separately due to the elevated need to have them resolved. An *issue* may raise a number of *follow-up inspections*, which in turn may lead to new (critical) *issues* being raised, and so on. The multiplicity between classes such as 1:1 and 0:n specifies the number of instances that a class interacts with runtime. This identifies *multiple instance relationships* between classes. Resources that are affected by a class such as an *equipment item* and an *issue report* are included in the class diagram although from here onwards we will focus only on the control flow perspective.

diagrams with concepts from UML sequence diagrams, allowing us to capture both intra-object and inter-object behavior in the same model. Secondly, the FlowConnect-based meta-model is a representative of other object-oriented meta-models (e.g. Proclefs [10], Merode [11], OCoN [12]), thus it is possible to generalise the results presented here to other meta-models.

At the highest level an **object model** is a container for all classes in an object-oriented PAIS. We define a class as a “cluster” of related states that share some common context. An example of shared context are states that belong to an *inspection*. Grouping these states together allows control flow associations to be made between them and an *inspection* begins to take shape. The object model contains one or more **classes** that contain one or more **state machines**. A state machine contains one or more **states**. A state models a moment in a process lifecycle where the context of the process can be distinguished and named, e.g. *Distribute Report*, *Load Data* or *Test Structural Strength*.

A **link** connects the output of a state to the input of another state. There are two types of link that can be used, which are a **transition** or a **signal**. Both of these links have a single source and target state. A transition connects two states in the same state machine and may have an Event-Condition-Action (ECA) rule. The occurrence of an **event** will cause the transition labeled by that event to be taken. A transition may also have one or more **conditions** attached to it that must be satisfied before the transition can occur. When a transition is taken it may also execute an **action**. We do not specify the exact details of an ECA rule language since this is out of scope of this work. On the other hand, a signal relates states belonging to different state machines as explained below.

Each state contains three sub-states; a **pre-processing gateway**, **main processing sub-state** and **post-processing gateway**. The main processing sub-state contains zero or more atomic **tasks** that needs to be completed. The pre-processing and post-processing sub-states are known as gateways because they are the entry and exit points for the main processing sub-state. A pre-processing gateway may be entered through an incoming transition. This gateway may send signals to other states and receive (wait for) signals from other states. Thus a gateway may have incoming and/or outgoing signals. The post-processing gateway is similar, but is responsible for control flow synchronisation after the main processing sub-state has terminated.

A **signal** enables a connection between the post-processing or pre-processing gateways of two states. In contrast to a transition, a signal does not have an ECA rule. There are three allowable types of signal: **spawn** (i.e. creates a new state machine) indicated by ‘S’, **finish** (i.e. terminating) indicated by ‘F’ and **message** (i.e. non-terminating) indicated by ‘M’.

Some signals occur in response to, or following another signal. For example, a terminating signal S2 may occur following a spawn signal S1. Accordingly, the meta-model allows signals to be paired then grouped into **signal relationships** that consist of a spawn signal and a non-spawn signal. creates a new instance of a child class. Messages exchanged between classes in a relationship are marked on a link with a signal name (e.g. “Sig01”) followed by ‘M’. In this example

each instance of the child indicates its completion to its parent by sending a *finish* signal as the return link marked with a signal name followed by ‘F’. The multiplicity of each relationship is indicated as a 1:1 or 0:n to indicate the lower and upper bound for the number of times that the relationship may be invoked by a parent class.

4 Reconciling the Approaches

In this section we introduce a proposal to map an object-oriented design to a process-oriented design model. The method aims to enable the semi-automated development of a process-oriented model from an object-oriented design. We have used our maintenance process as a test scenario in an implementation of the object-oriented meta-model and conversion algorithm of this proposal.

From Object Models to Process Models

To reconcile the object-oriented and process-oriented approach, we have followed the procedure shown in Figure 4.

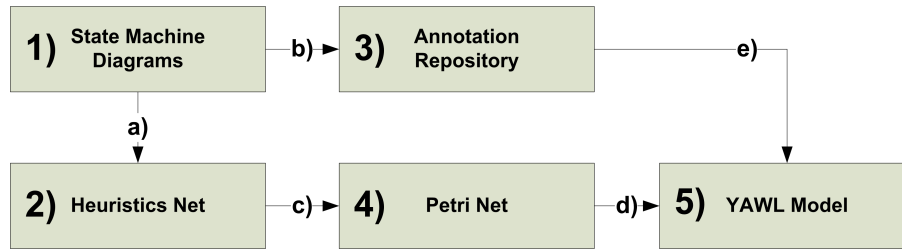


Fig. 4. Overview of Reconciliation Procedure

The required steps in the procedure to convert between the deliverables shown in Figure 4 are:

- a) Heuristics Net Algorithm (Algorithm I).
- b) Record Information in Annotation Repository (Algorithm I).
- c) Petri Net from Heuristics Net (Algorithm II).
- d) Convert Petri Net to YAWL Model.
- e) Apply Annotations to YAWL Model.

As defined in [13], a heuristics net is a representation of the control flow behaviour of a process specification. In the Process Mining (ProM) framework³ heuristics nets are used to convert between different representations of a process, specifically from event logs to Petri Nets. In our case a heuristics net enables conversion between object-oriented and process-oriented representations of a process as it provides a “middle-ground” between either model, shown in Figure 5.

³ ProM is available at <http://is.tm.tue.nl/~cgunther/dev/prom/>

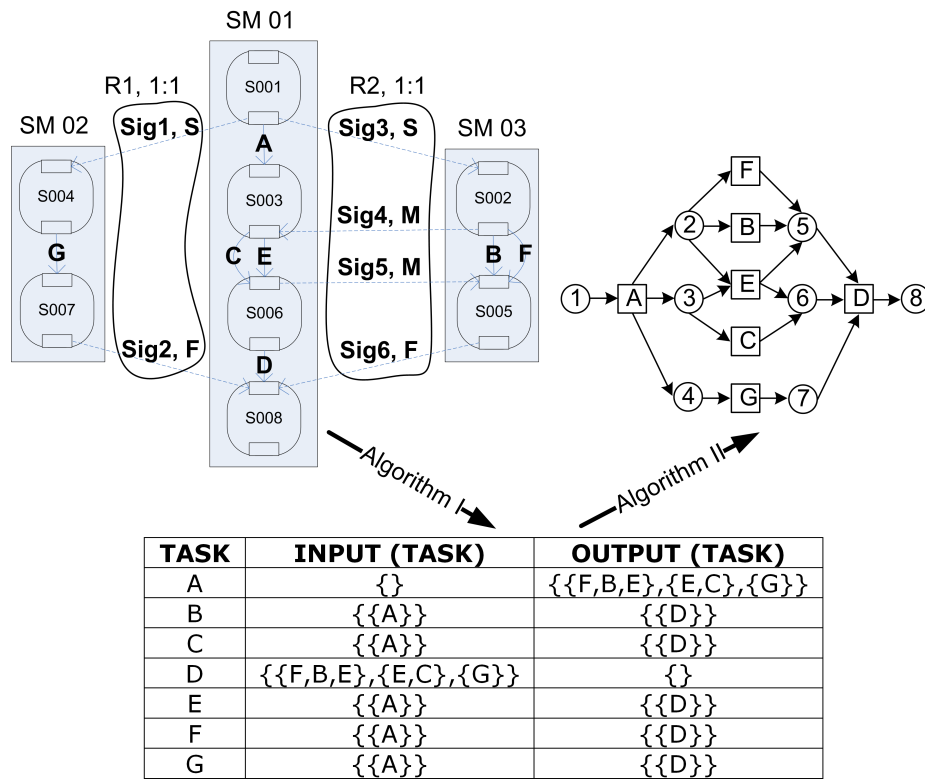


Fig. 5. Reconciliation Using a Heuristics Net

Step a) involves the application of *Algorithm I* (see Figure 6) to an object model. This algorithm provides a complete heuristics net as output.

Step b) involves capturing missing information in the form of annotations that are stored in a repository. This is done as part of *Algorithm I*. Heuristics nets focus on process control flow but there is low-level information expressible in an object-oriented model that can not be expressed in a heuristics net. The result is that a self-standing heuristics net does not give a complete view of process control flow because the information listed below can not be captured in a heuristics net. These annotations are looked up when converting the heuristics net into a dedicated process-oriented modelling notation such as YAWL or BPMN so that the information not expressible by the heuristics net is added to the process model. Annotations are added to identify these elements:

1. Multiple instance tasks.
2. Structured loops.
3. Event, condition, action expressions (ECA Rules).

For the object model.
 For each class in the object model.
 For each state machine in the class.
 For each state in the state machine.

1. **XOR-input check:** Find input disjuncts for each incoming transition and incoming signal. Add disjuncts to input set.
2. **AND-input check:** Find input conjuncts for each incoming transition and incoming signal. Add conjuncts to input set.
3. **XOR-output check:** Find output disjuncts for each outgoing transition and outgoing signal. Add disjuncts to output set.
4. **AND-output check:** Find output conjuncts for each outgoing transition and outgoing signal. Add conjuncts to output set.
5. **Tau transition check:** Add a Tau transition to the input set if a spawn signal is sent from the pre-processing gateway. Else add a Tau transition to the output set if a finish signal is received by the post-processing gateway.
6. **Annotation check:**
 - * Add "Multiple Instance container task" annotation if a spawn signal is sent from the input or output gateway.
 - * Add "Multiple Instance task decomposition" annotation if the state belongs to a child class in a relationship.
 - * Add "Structured loop entry point" annotation if looping script exists on an input or output gateway.
 - * Add "Structured loop exit point" annotation if the target state of one of the exiting transitions is a "structured loop entry point".
 - * Add "ECA rule" annotation if an ECA rule exists on transition.
7. **Insert input and output set:** The complete input set is inserted as a row and the complete output set is inserted as a column in the heuristics net.

Fig. 6. Algorithm I

In **Step c)** the heuristics net is sent to the Heuristics Net tool⁴ in ProM to obtain a Petri Net. This is *Algorithm II* in Figure 5. This Petri Net gives us insights into a process-oriented representation of what the control flow in an *inspection* looks like. **Step d)** is now performed to create a skeleton structure of a process-centric model that represents the "pure" control flow in terms of tasks and their control flow relations. Here we can make use of existing conversion plugins, such as the conversion plugins in ProM [14].

Step e) completes the process-centric model. The process-centric model is parsed task-by-task. If an annotation is found for that task in the annotation repository, the details in the repository are added to the process-centric model. The application of this procedure to our example in Section 2 provides the executable end-to-end process-centric model in Figure 7 of an *inspection* using the YAWL modelling notation. In a process-oriented form we can now see the end-

⁴ The org.processmining.convertng.HNetToPetriNetConverter class

schema as output at the conclusion of the lifecycle. Our proposed approach would be an extension to these design methodologies and allow process analysts and designers to produce a completely process-oriented view of an object-oriented model. FlowConnect [8], the Business Modelling Language (BML) [17] and Proclerts [10] are examples of object-oriented systems and modelling notations. This influences these kinds of systems and languages in a way that allows them to map processes to an implementation in an object-oriented programming language.

Reijers et al. proposed a methodology for Product-Based Workflow Design (PBWD) that presented an analytical clean-sheet approach for process design specified by the bill-of-material for products that are affected by the process [18]. The focus on the bill-of-material for designing product manufacturing workflows means that this modelling approach can be considered to be object-oriented. A limitation of PBWD is there is no notion of object life-cycles, which is an area covered by artefact-centric process modelling [19]. This approach unifies data and process in an “Operational Specification” but does not consider the possibility of converting this specification to alternative modelling representations.

There are many example of process-oriented systems, modelling languages and standardisation efforts. Some examples include ADEPT [20], YAWL [5], BPMN [3] and XPDL [21]. Due to the process-oriented way of thinking in each of these examples, object-oriented design techniques are generally not available to these approaches, making these approaches unsuitable for modelling object-oriented applications.

An architecture for mapping between object-oriented and activity-oriented process modelling approaches has been proposed by Snoeck et al. and implemented as the MERODE project [11]. The developers of this architecture recognised the advantages of mapping object-oriented development to process modelling. Similar to our approach, an object-relationship diagram models object associations and business rules and an object-event table models the behaviour of domain objects. Consideration of the various kinds of control flow splits and joins between objects appears to be missing from this architecture which we consider is necessary for object-oriented design.

Bontemps et al. have presented an alternative approach to reconciling modelling approaches [22]. Their approach focuses upon reconciling sequence charts and state machines whereas we consider that process-oriented models such as activity diagrams are the most relevant to kind of models for Information Systems. MENTOR [23] uses a design approach that combines state charts and activity diagrams to develop a process specification. The presented method of partitioning these diagrams and assigning activities to states is related to our work but our approach attempts to reconcile modelling approaches rather than combine them in a single model.

6 Conclusions and Future Work

Mainstream approaches to implement software systems in general, and Information Systems in particular, are generally based on object-oriented technology.

This predominance distills into mainstream analysis and design practices (e.g. those based on UML), which are based on concepts of objects whose structure is captured as classes and whose behavior and interactions are captured as state machines, sequence diagrams and similar notations. On the other hand, recent trends have seen an uptake of approaches to Information Systems engineering that treat processes as a central concept throughout the development lifecycle. These approaches, which naturally lead to “process-aware” systems rely on the progressive refinement of high-level process models down to detailed executable process definitions that can be deployed into process execution engines.

Unavoidably, the co-existence of these two approaches to Information System engineering leads to situations where a project starts with a model corresponding to one approach (e.g. an object-oriented design model) and needs to switch to a model corresponding to the other approach (e.g. a process-oriented design model). A number of issues arise during such handovers, which are natural to expect given that the object-oriented and process-oriented approaches correspond to different ways of decomposing the business logic of a system. In this paper, we have proposed an approach to bridge these differences in terms of the control flow logic and discussed how the conversion technique has been implemented.

Future work will continue on the topic of transforming object-oriented models to process-oriented models and vice-versa. There is a need to cover not only the control-flow aspects as outlined in this paper, but also data flow and resource allocation. We also note that a similar problem arises in the opposite direction, i.e. moving from a process-oriented to object-oriented design for the purpose of implementing process-oriented design models using object-oriented technology. Therefore another future challenge will be a proposal of a reverse transformation from process-oriented to object-oriented models.

References

1. Kueng, P., Bichler, P., Kawalek, P., Schrefl, M.: How to compose an object-oriented business process model? In: Proceedings of the IFIP TC8, WG8.1/8.2 working conference on Method engineering : principles of method construction and tool support, London, UK, Chapman & Hall, Ltd. (1996) 94–110
2. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley Professional (1998)
3. Object Management Group: Business Process Modelling Notation (BPMN), Version 1.0. (2006)
4. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. <http://dev2dev.bea.com/webservices/BPEL4WS.html> (2003)
5. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. *Information Systems* **30** (2005) 245–275
6. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., eds.: Process-Aware Information Systems. John Wiley & Sons, New Jersey (2005)

7. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: *Advanced Information Systems Engineering, 17th International Conference (CAiSE)*, Porto, Portugal (2005) 216–232
8. Shared Web Services Pty. Ltd.: FlowConnect Model. Background notes prepared for meeting at QUT (August 5, 2003)
9. Halpin, T.: *Information modeling and relational databases: from conceptual analysis to logical design*. Morgan Kaufmann Publishers Inc. (2001)
10. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclats: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems* **10** (2001) 443–481
11. Snoeck, M., Poelmans, S., Dedene, G.: An architecture for bridging OO and business process modelling. In: *33rd International Conference on Technology of Object-Oriented Languages (TOOLS)*, Mont-Saint-Michel, France (2000) 132–143
12. Wirtz, G., Weske, M., Giese, H.: The OCoN Approach to Workflow Modeling in Object-Oriented Systems. *Information Systems Frontiers* **3** (2001) 357–376
13. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic Process Mining. In: *Applications and Theory of Petri Nets, 26th International Conference, ICATPN 2005*, Miami, USA (2005) 48–69
14. Verbeek, H., van Dongen, B., Mendling, J., van der Aalst, W.: Interoperability in the ProM Framework. In Latour, T., Petit, M., eds.: *CAiSE 2006 Workshop Proceedings - Open INTEROP Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP 2006)*. (2006) 619–630
15. Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Corporation (2004)
16. Kim, J., Carlson, C.R.: A Design Methodology for Workflow System Development. In: *Databases in Networked Information Systems (DNIS)*, Second International Workshop, Aizu, Japan (2002) 15–28
17. Wohed, P., Perjons, E., Dumas, M., ter Hofstede, A.H.M.: Pattern Based Analysis of EAI Languages - The Case of the Business Modeling Language. In: *Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS)*, Angers, France (2003) 174–184
18. Reijers, H.A., Limam, S., van der Aalst, W.M.P.: Product-Based Workflow Design. *Journal of Management Information Systems* **20** (2003) 229–262
19. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Systems Journal* **42** (2003) 428–445
20. Reichert, M., Rinderle, S., Dadam, P.: ADEPT Workflow Management System. In: *Business Process Management*, Eindhoven, The Netherlands (2003) 370–379
21. Workflow Management Coalition: Workflow Process Definition Interface - XML Process Definition Language, Version 2.0. http://www.wfmc.org/standards/docs/TC-1025_xpdl.2-2005-10-03.pdf (2005)
22. Bontemps, Y., Heymans, P., Schobbens, P.Y.: From Live Sequence Charts to State Machines and Back: A Guided Tour. *IEEE Transactions on Software Engineering (TSE)* **31** (2005) 999–1014
23. Muth, P., Wodtke, D., Weißenfels, J., Dittrich, A.K., Weikum, G.: From Centralized Workflow Specification to Distributed Workflow Execution. *Journal of Intelligent Information Systems (JIIS)* **10** (1998) 159–184