Building Extensible Workflow Systems Using an Event-Based Infrastructure

Dimitrios Tombros and Andreas Geppert

Department of Information Technology, University of Zurich
{tombros, geppert}@ifi.unizh.ch

Abstract. This paper describes an approach towards the systematic composition of workflow systems using an event-based approach. The complex nature of workflow systems and the heterogeneity of the application systems which are integrated require powerful general purpose composition mechanisms. Furthermore, it is advantageous if the functionality of the underlying workflow management system can be adapted to the type of workflow applications for which a system is intended. We propose an extensible event-based architectural framework for workflow systems which allows the composition of workflow systems by reuse and customization of reactive components representing workflow system processing entities. We also consider the structure of a build-time repository to support this architectural framework.

1 Introduction

Traditional software development methods do not meet the evolving requirements of heterogeneous and process-oriented systems. A principal open problem remains the systematic development of workflow systems (WFS) based on appropriate abstractions. The current state of WFS technology has not yet reached consensus on the proper component abstractions for the workflow domain. Instead, there is a proliferation of research proposing the benefits of particular workflow management concepts. Some efforts, such as those of the Workflow Management Coalition and the OMG Workflow Workgroup [16], attempt to converge various ideas in the workflow management community. Still, however, WFS development remains a largely ad hoc effort done on a case-by-case basis.

A methodical approach to WFS development utilizes component-based engineering, in which new WFS are composed out of already existing, reusable artifacts. Component-based WFS development is however in a very early stage. This is not only an inherent problem of workflow management technology but can also be explained in the general context of system composition research. Based on general observations concerning software composition [12], we consider the following facts as especially relevant for WFS development:

- It is not clear how domain knowledge should be captured and formalized to support component-oriented development. Especially in the domain of workflow management, there has been little or no research in this issue. The definition of reuse-oriented domain-specific models is still an ongoing process.

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 325–339, 2000.

- There is no synergy between analysis and design. The relation between workflow specification and workflow implementation has not been considered under the perspective of reusability. For example, most workflow specification languages provide reuse mechanisms for workflow artifacts, but do not consider the reuse of processing entity (PE) implementations. The mapping of workflow specifications to WFS implementation is ad hoc.
- There are no generally accepted methods for the design of frameworks supporting component-based WFS development. In the domain of workflow management an architectural perspective and the resulting framework-based development approach is in a very early stage.
- There are no software tools which facilitate component-oriented WFS development. The support provided by workflow management systems (WFMS) in this respect is limited to mere workflow specification.

We follow a composition-based approach for WFS built around an appropriate architectural framework. The approach supports the architecture-centric development and extension of WFS. It consists of the following elements:

- a lightweight but extensible generic event-based workflow execution and application integration platform;
- an approach for the analysis of the WFS architecture and the classification of WFS components. We consider components as self-contained units of abstraction, with defined connection interfaces, and an individual life cycle (see, e.g., [12,19] for more elaborate discussions of the nature of components);
- a domain-specific metamodel for the description of the architecture and functionality of the intended WFS;
- a framework for the composition of WFS out of pre-existing parameterized component templates; and
- support for the reuse-based composition of WFS through an architecture artifact repository and the event-based execution platform. This infrastructure provides a workflow specification execution system.

In this paper, we concentrate on the domain-specific metamodel and the compositional framework for WFS. Due to space considerations, we describe only briefly the underlying analytical approach and the repository-based reuse of WFS components.

The remainder of this paper is organized as follows. The next section introduces the WFS development life cycle. Sections 3 and 4 discuss PE as reusable components. Section 5 presents repository support for WFS construction, section 6 discusses relevant related work, and section 7 concludes the paper.

2 Workflow System Development Life Cycle

In this section, we introduce a WFS development life cycle based on composition of reusable components. A WFS consists of heterogeneous real-world PE each of which contributes to the tasks of the WFS in some way. Typical PE

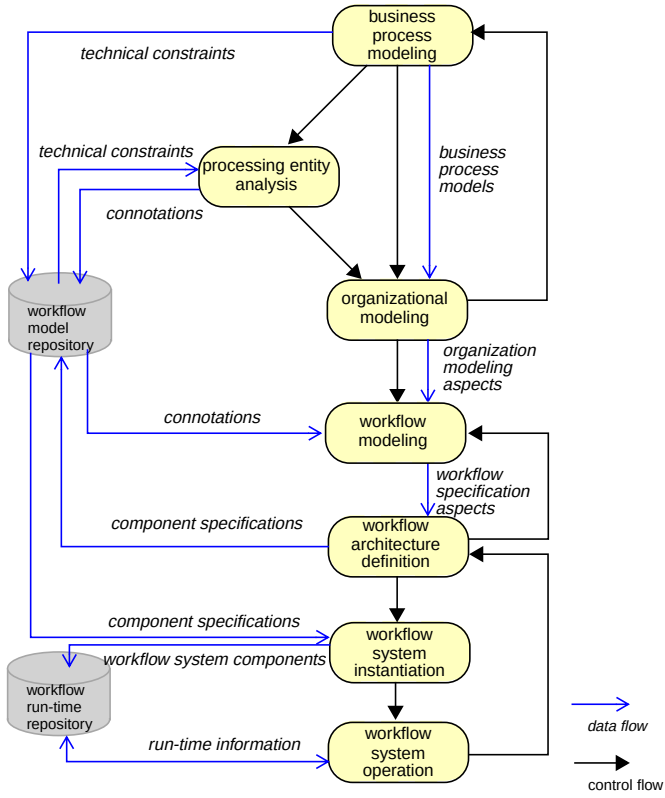

Fig. 1. Workflow system development lifecycle

include application software, WFS interface software (e.g., worklist interfaces, or administration tools such as a workflow execution monitor and visualization tool used as an example in this paper), and human entities. Depending on the specific workflow application, a different set of PE may be required. These are represented by appropriate WFS components as described below.

In a nutshell, the aim of the WFS construction process is to compose reusable representations of the PE together into a coherent system. Any WFS provides minimal functionality for workflow management. This core WFMS comprises an event composition, routing and notification subsystem (described in more detail in [8]), a subsystem for executing event-condition-action rules (ECA rules) [22], buildtime and runtime repositories, as well as a set of component design and instantiation tools. Additional components can be added to the WFS as required, organizational aspects of the PE can be defined, workflow specifications can be designed, and eventually, workflow instances can be created and enacted. The lifecycle consists of the following groups of activities (see Fig. 1).

We assume that a project initiation phase precedes the entire lifecycle. In this phase, the goals with respect to the developed WFS are set. The project management team determines which business processes will be automated by the WFS. An appropriate *business process modeling* (BPM) formalism should be selected at this point. It is important however, that a mapping is provided by the BPM environment to the target workflow specification language.

Processing Entity Analysis and Integration. PE are integrated in a WFS; they either extend the functionality of the core WFS or are the resources that carry out the business-related workflow tasks. For example, in the course of our work we were able to augment the core WFS with workflow monitoring and event auditing components. The analysis and integration of PE includes the composition of a reactive component template which represents the PE based on the analysis of its characteristics. These characteristics determine the properties of the component that represents the PE. For each PE it is defined which workflow tasks (services) it provides to its clients, which are in general other PE.

Organizational Modeling. Once new PE (i.e., their representing component templates) have been added to the WFS, organizational aspects such as specific attributes (e.g. costs per execution time unit) and relationships between PE (e.g. is-supervisor-of relationships among human PE) may have to be modeled to support the definition of task assignment rules refering to organizational aspects.

Workflow Modeling. New workflow specifications can be constructed by referencing and using information available about PE and other workflow specifications. The use of an appropriate workflow specification language which supports the reuse of process definition artifacts is required. For the monitoring component mentioned above, its behavior was specified such that it should be notified of the occurrence of the events which the WFS administrator wanted to monitor.

Workflow Architecture Definition. The next phase involves the development of a homogeneous event-based architecture model of the WFS. The architectural elements are summarized in Table 1, where the correspondence between relevant facts from the real world, their respective aspect in workflow specification, and finally the provided architectural abstraction is depicted.

Workflow System Instantiation and Enactment. New instances of the defined workflow component templates can be created. These create and enact the defined workflow models.

Multiple iterations of these above steps may be required to compose a WFS.

3 Analysis of Workflow System Architecture

Successful composition-based construction of WFS requires appropriate abstractions as well as the maintenance of sufficient information about their functionality and semantics To that end, we represent explicitly the integration-related properties characterizing a PE type with respect to the WFS under construction. These properties are termed *processing entity connotations* (see Fig. 2). Components and connectors [1] are implementation level constructs defining the structure and communication paths of a WFS. PE connotations associate the

Table 1. Workflow system architecture definition elements

Business process model aspect	Workflow modeling aspect	Architectural abstraction
information system components and connectors	workflow system architectural structure	reactive component templates, event type registrations
organizational constraints under which the workflow is executed	organizational model	organizational relations
PE functionality used in business process	workflow tasks	services
activity execution sequence	control flow	ECA-rules
activity data dependencies	data flow	event parameters
business rules	task assignment	suitability, dispatching
human role, application functionality	PE behavior	ECA-rules, rule packages

properties of the PE type with the characteristics of the components and connectors needed to adequately represent PE in the WFS.

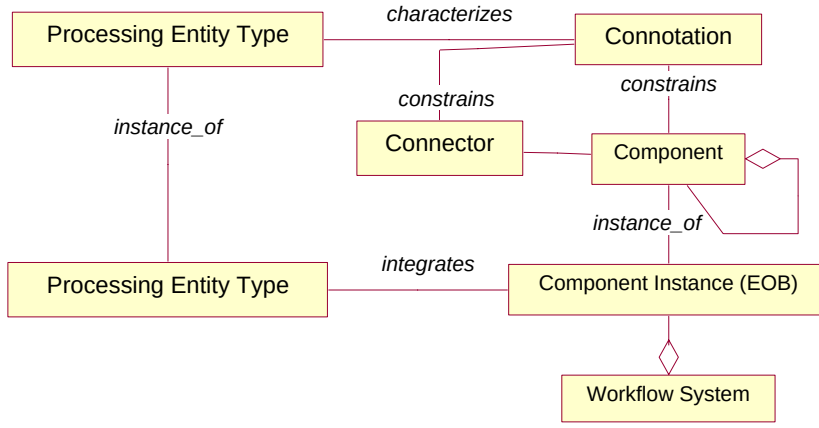

Fig. 2. Relationships between workflow system composition elements (in UML)

The definition of PE connotations is based on a faceted classification scheme [15] and consists of so-called *traits* (properties of PE). Unlike the pure faceted classification, leafs of the classification trait trees may be either *simple terms* or *expressions*. Simple terms are chosen from the standard vocabulary of the domain, while expressions describe a dependency to other elements of the WFS architecture—elements which are not necessarily components. Such elements may be defined outside the scope of the classification scheme (e.g., in the workflow model). The classification serves three main purposes: (i) determination of the required properties of the components used for PE representation, (ii) analysis of the integration issues facing the composer of a WFS, and (iii) architecture-centric based development and reuse [3].

Table 2. Connotation of the EVE monitoring interface component

Participation	Optionality	optional
	Multiplicity	max(num_of(EVE_server))
	Dependency	local_EVE_server, defined(MonitorWF)
	Functionality	user interaction, display, notification
Implementation	Automation	interactive
	State	yes
	Server type	concurrent
	Guarantee	none
	Access point	in, out: implicit-triggering (EVE messages)

Participation traits characterize a PE's role in a particular WFS. They define architectural constraints on the WFS:

- *Optionality* refers to the requirements concerning the existence of the PE. The implications of the trait concern the existential properties of component instances representing the PE in the WFS.
- *Multiplicity* refers to the number of instances of the PE type which may be concurrently active. This affects the required task assignment protocols.
- *Dependency* refers to the assumptions made by the PE for its operation with respect to the existence of other PE and provided services.
- *Functionality* provides an implementation-independent characterization of the PE's role in the WFS.

Implementation traits constrain the components of the WFS which represent/implement the PE. They include its degree of automation, its ability to maintain information about its process execution context and across workflows, server-related properties, execution guarantees and transaction support, and access points, i.e., the mechanisms of interaction that the PE supports (the incoming interface) and the mechanisms of interaction it uses (the outgoing interface).

Additional traits may be required and even within traits which seem complete, i.e., all except those concerning interface and service, additional terms may be required. Such modifications and additions are not excluded by the classification scheme. However, the implications and effects they have on the architectural framework will have to be considered.

An example of a connotation for a monitoring interface component is presented in Table 2. The participation traits define that the component is not required for system operation and may be instantiated at most once for each server in the system. Its operation depends on the operation of the server for which it is instantiated and on the definition of a monitoring workflow type defining monitoring services. The component provides user interaction, display, and notification functionality.

4 Reusable Workflow System Components

Processing entities are represented on the architectural level by instances of composite reactive components called *event occurrence brokers* (EOB). They execute workflow tasks as a reaction to workflow-relevant situations in their environment. The occurrence of these situations is manifested by events. EOB generate and react to WFS events, and map their meaning to concepts present in the vocabulary of the respective PE. In other words, they provide a homogeneous representation of heterogeneous real-world PE and provide the PE functionality to the WFS-internal miniworld.

4.1 Events and Services in Workflow Systems

The most flexible and loose model of integration is asynchronous interaction based on events. An event-based approach to data, control, and process integration in a WFS architecture has many advantages [2]. Hence, in our approach events are the sole component integration and interaction mechanism, i.e, they are the only available type of connectors between EOB in a WFS. Events are used for the following purposes:

- signaling of relevant workflow situations and definition of control flow,
- invocation of functionality provided by PE, and
- exchange of data between PE.

The event typology is a tree subdivided into primitive and composite events. Primitive events belong to the following types: *time events*, *interaction events*, and EOB *internal events*. EOB internal events are relevant only for the implementation of EOB-internal communication.

Time events can be absolute, relative, or periodic. Absolute time events express real-time points and are defined in terms of a time specification expressing a date and time recorded by a local clock site.

The exchange of coordination information between EOB takes place by the signaling and reaction to EOB interaction events. Thus the semantics of EOB interaction events are associated with execution states of workflows. The manifestation of these events are event messages which are forwarded by the underlying communication infrastructure to EOB which have a registered interest in these events, called the event listeners. Interaction events are service requests, confirmations, replies, and exceptions. These events contain system-provided (implicit) and user-provided parameters. Through the use of service request events the execution of services by EOB can be triggered. The initiation of service execution is signalled by request confirmation events. Finally, the results of service execution can be communicated to other EOB by service reply events.

The special meaning of exception event types with respect to the other event types consist in the fact that for each such type defined, a corresponding exception handling component must be defined, i.e., an EOB which has some predefined reaction on the event occurrence.

Composite events are used to express complex workflow situations. As with primitive events, we distinguish between event types and occurrences of these types. Composite event types are defined by applying unary and binary event operators to event types. The event operators include conjunction, sequence, exclusive-or and inclusive-or disjunction, repetition, iteration, and negation. The main extension is the concurrency operator which is meaningful only in the context of distributed systems. Composite events have formal semantics described in detail in [20]. It is important to mention that workflow execution based on events produces an *event history*.

Services are defined based on the function traits of PE. A service is provided by one or more EOB in the WFS called the server(s) in the context of a service execution. It can be requested by other EOB which are the clients in the context of that execution. Services define the PE capabilities assumed for a given WFS, independent of the EOB defined in the system. They represent the interface between workflow specification and WFS architecture and can be leveraged to workflow task types declared in a workflow schema. In other words, workflow specification makes use of the service abstraction in order to define the functionality of desired processing steps.

A service identifies an operational interface with a predefined set of parameters which are provided by the client EOB during service request. It also identifies a set of replies which express the possible outcomes of a service execution. These replies may have various parameters provided by the server EOB.

4.2 Event Occurrence Brokers

An EOB is an instance of a composite component template built from reusable subcomponents which implement parts of the EOB functionality and communicate through an internal message bus. The type of the EOB determines its subcomponents, their properties, and the events they understand. Every EOB includes at least the following subcomponents: an event delivery (EDI) and an event posting (EPI) interface, a persistent state management subcomponent, an ECA rule management subcomponent, and a PE management subcomponent. This structure is depicted in Fig. 3.

The component instances composing a WFS communicate by broadcasting event messages over a reliable shared message bus which provides a single message broadcast operation. Thus, all component instances connected to the message bus are notified of an event message and react accordingly. The set of messages each component understands is fixed for each subcomponent type. The general EOB-internal message structure is the following: *(message_name, message_origin, message_parameters)*. The message parameters are typed using the OMG Interface Definition Language [13]. Messages can be sent either in a synchronous or asynchronous mode; in the first case the message sender expects a response message with the output parameters, in the second case the message sender can continue processing.

An EOB contains a *state manager* (STATEMAN) implemented over some storage system not further specified. The STATEMAN is responsible for the

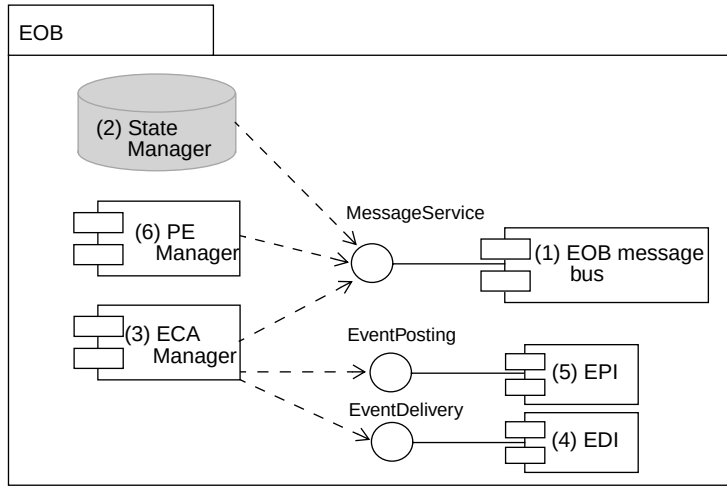

Fig. 3. General structure of an EOB and its subcomponents

persistent storage of the EOB state objects. Persistence is achieved by a system-provided persistent root whose constituents, i.e., composite objects declared in the EOB state persist automatically.

Every EOB contains a *rule management* (ECAMAN) subsystem responsible for management and execution of ECA rules expressing the behavior of EOB. ECA rules are used to implement the following functionality:

Initiation of service execution by PE. Workflow service specification does not always correspond directly to the operations or services provided by PE. For example, a workflow service may be implemented by a series of database queries and the subsequent evaluation of the results. This processing logic is described in ECA rules which bridge the semantic gap between application operations and services provided (by EOB) in the WFS.

Enforcement of task execution ordering and guarding of task execution conditions. The workflow schema expresses various dependencies among workflow tasks. These dependencies are expressed by composite events and by the rule actions which generate new events which eventually trigger the subsequent tasks. ECA rule conditions can be used to express task execution constraints which can be evaluated against task execution results available through event parameters.

Exception and failure handling. ECA rules are used to specify failure handling. They can also express recovery policies.

Rule execution by the ECAMAN is performed as follows:

1. An event arrives at the delivery interface of the ECAMAN.
2. At some point in time, the ECAMAN process examines the oldest event in the delivery queue by sending EDI the message *peek()*, and determines the set of active rules that are fired by this event.

3. All fired rules are inserted into a persistent pending evaluation queue, where the relative order of the rules depends on the defined rule priorities. If no rule priorities are defined, the queueing order depends on the specification order or rule creation timestamp (starting from the oldest rule first).
4. The conditions of the rules in the queue are evaluated sequentially. For each rule whose condition is true, the action will have to be executed.
5. A reference to such rules is inserted into a persistent pending execution queue. Rules whose conditions are evaluated to false are removed from the pending evaluation queue.
6. The event is removed from the EDI by sending it a *consume()* message. Each rule in the pending execution queue is then examined as follows: if the event is a request and the rule action produces a reply event, then a confirmation event is posted to the WFS through the EPI with the message *put()*. This actually guarantees that a reply or exception must be generated by the EOB at some later time. Although only a single rule can generate a reply, multiple rules may be triggered by the same request, without however producing a subsequent reply.
7. The rule actions in the pending execution queue are executed sequentially. After action execution is completed, each rule is removed from the pending execution queue. If an action produces a reply event, this is posted to the WFS through the EPI.
8. When the pending execution queue is empty, the next event can be examined from the delivery interface.

Interaction with PE is implemented by the *PE management* (PEMAN) subsystem. In general, the PEMAN implements the access to the functionality of PE by using some wrapping technique to access the external systems; it may alternatively implement the desired functionality itself in some programming language without aceesing other software. The PEMAN however understands a set of operations which are called in ECA rule actions. It implements the automation and access point traits of the PE connotation (see above).

As already mentioned, the PEMAN may wrap an external system (e.g., a database application), implement system functionality (e.g., store an event log), or provide a user interface (e.g., the monitoring interface). When the PEMAN directly implements desired functionality, the WFS infrastructure plays the role of a white-box PE for a workflow task. The implementation of different wrapping techniques is a problem which has been addressed by a large body of research; in our work we concentrate on the issues relevant to the use of these wrapping techniques within a reactive component-based WFS architecture.

In the case of the monitoring EOB, the component has to be registered as a listener to all events which denote workflow situations to be monitored (e.g. all service request events). It reacts to occurrences of these events by changing its internal state to visualize the execution state in a defined way (e.g. change the color of a graphical screen element).

Table 3. The EOB typology based on the ECAMAN and PEMAN properties

EOB type	ECAMAN rule action execution	PEMAN functionality
caller	single-threaded	execution of envelope
uni-server	single-threaded	implementation of a wrapping method for an external server system
multi-server	multi-threaded	implementation of a wrapping method for an external server system
user	single-threaded	user worklist management and interaction
group	single-threaded	-
uni-extender	single-threaded	implementation
multi-extender	multi-threaded	implementation

Depending on the properties of EOB subcomponents, different types of EOB can be composed. The typology—as determined from the properties of participating ECAMAN and PEMAN—is presented in Table 3.

A complete WFS assumes the existence of a specialized EOB called an event engine (EVE). EVE is a distributed glue system which—among other functions—provides event composition and notification functionality to EOB according to the interest they have registered. EVE is described in more detail in [8] and enforces the semantics described in [20]. EOB in a WFS consume and produce events. The correctness of their behavior and thus the correctness of the operation of PE in the WFS can be examined over the resulting event history. An EOB has behaved correctly iff:

- it has reacted to all requests it is responsible for;
- it has reacted to all situations for which it defines workflow-specific rules and it is responsible for; and
- it has produced a reply or an exception for each request it has confirmed.

Finally, we note at this point that the correctness of the composed WFS depends on the maintenance of certain invariants which can be derived from participation traits of the involved PE. For example, dependency traits describe completeness constraints on the WFS architecture which can be examined at the EOB-implementation level. A detailed discussion of such invariants is omitted due to space considerations.

5 Repository-Based Composition

The abstractions previously described support the composition of WFS. The main abstractions—EOB types and their services—can be most closely compared to software schema abstractions [10]. Software schemas are formal extensions to reusable components which, however, emphasize the reuse of abstract algorithms and data structures. The abstraction specification of the software schema is the

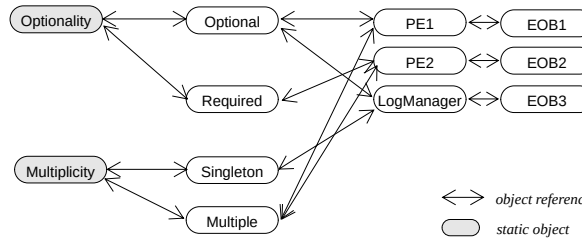

Fig. 4. Example of participation traits and PE in the buildtime repository

formal exposition of the algorithms and data structures, in service definitions, behavior and state of the EOB types. The abstraction realization corresponds to the source code produced when EOB types are instantiated.

The development of effective and usable repositories for software artifacts represent an active field of research (e.g., [5,9]). In our work we have attempted to utilize the experience and results gained from this research within the context of WFS composition. The object-oriented build-time repository schema represents a complete mapping of WFS artifacts as discussed next.

For each participation trait an object of type Participation_Trait exists in the build-time repository. Its attributes are the terms—which can be composite objects in themselves—defined for the trait. A term contains a collection of references to objects of the type Processing_Entity, each of which can be associated with an EOB object. This structure is exemplified for two traits in Fig. 4.

Organizational units defined in the organizational schema are represented by Organizational_Unit objects. Organizational relationship types are represented as OrgRel_Type interface types.

Each EOB is represented by a composite EOB object, whose structure and implementation depends on the EOB type. An appropriate interface type is defined for each EOB type. Subcomponents of an EOB are instances of STATEMAN, PEMAN, and ECAMAN types. These are collections of references to State_Variable_Type objects, Operation_Type objects, and ECA_rule objects. Implementations of PEMAN subcomponents are stored as symbolic links to object files.

Services are represented by Service_Interface objects. For each service definition a corresponding Request and Confirmation event type object is created. For each reply and exception event type a corresponding Reply and Exception event type object is created. Confirmation, reply, and exception event type objects have bidirectional references to their request event type object. Event type parameter objects are created for each parameter based on corresponding interface types.

An interface type ECA_Rule for ECA rules is defined. It contains references to the event type objects in its event clause. Composite event types implicitly defined in the rule event clause are represented by objects of the appropriate interface type. These are created when the rule object is stored in the repository. An ECA_Rule also contains references to conditions. Conditions are sequences of clauses containing object references of the form *({AND|OR|NOT}, name, type,*

logical_operator) where name and type are references to corresponding objects defined in the repository, the logical operator must be defined for the type, and the boolean operators determine the relationship to the previous element. The precedence order of the expression determines the order of sequence elements. Actions are sequences of operation_name clauses where operation_name is a name of an operation type defined for the EOB, or a raise-operation for an event type. The corresponding objects can be retrieved through predefined queries. Additionally, actions of extender rules can contain symbolic links to source and object code files.

During the creation of objects in the build-time repository, some objects can be in the *in-development* state (e.g., because they reference names of as yet undefined objects). This is allowed during the composition phase of a new WFS. However, when the WFS is instantiated, every participating object must be in the *in-production* state. This is ensured by the transactions which create objects in the runtime system and ensure that the WFS invariants are respected. Additional states can be defined if required by the development process.

6 Related Work

The systematic development of workflow management applications has been recently identified as an important research issue. Consequently, some relevant publications have appeared which advance the issue beyond initial attempts focussing primarily on the expressiveness of workflow modeling.

In [14], an object-oriented environment for the development of transaction-based workflow applications is provided whose emphasis lies on the reuse of process objects with predefined scheduling and transactional behavior. The—rather strong—assumption of the existence of a mapping between external operations and native operations is made.

A stronger emphasis on application integration and the definition of a workflow application system architecture is presented in [17] where a type library for wrapping different applications is provided which are subsequently implemented by different execution objects. The provided support is focused on the implementation aspect and does not consider in depth the systematic reuse of software components. Similarly, in [21] a framework for the systematic integration of software tools in process-centered environments by using specialized wrappers is described. Although the authors only consider software development tools, their work is applicable to WFS. Compared to our work however, the emphasis lies on the implementation of the integration mechanisms and not on the definition of a reusable component-based system architecture.

APEL [7] is a platform-independent environment for the development of process-oriented systems. It provides modeling abstractions for various aspects of such systems and maps them to low-level executable formalisms. The initial emphasis has been in covering a large spectrum of design activities; the authors state that support for systematic reuse is part of their intended future work.

Event-based approaches have been advocated by several authors [4,6,11] for workflow execution and cooperative information systems. In contrast to these efforts, events in our approach are conceptual and architectural constructs (instead of execution level constructs). In contrast to CoopWARE [11] (which considers services as the major abstraction), PE and their representation as EOBs are the most important abstractions on which the composition of WFS is based.

Finally, we refer at this point to emerging component standards and especially to Enterprise JavaBeans [18] which proposes a generic component model for server-side components. This model is bound to a particular programming language and considers application-neutral implementation aspects. Our approach, in contrast, adresses domain-specific issues, attempts to provide support for the development of a specific class of applications, and is explicitly kept independent from a particular implementation language.

7 Conclusions and Evaluation

The systematic composition of WFS provides an efficient way to construct the large complex application systems that are typical in the workflow management domain. In this paper we described an approach for the reuse-based development of such applications:

- we implement WFS by extending the functionality of a core event-based integration platform;
- we provide mechanisms for the characterization and classification of PE;
- we provide a component-based architectural framework for WFS; and
- we support a repository-based development process which can benefit from extensive large-grain component reuse.

Despite the promising initial experiences we have made with our system, it still remains to be seen if a reuse-based WFS development approach can become part of an industrial environment. As the main problems during the introduction of the approach we anticipate the initial population of the repository with process entities which actually have a reuse potential and the implementation of PEMAN components for various applications. Especially the second task is complex and requires the use of developer resources without an immediately visible return-on-investment. Furthermore, in our future work we plan to extend the system to support WFS evolution through workflow component versioning and a set of well-defined modification operations on EOB. These operations should respect the invariants defined in the WFS framework.

References

1. R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. on Software Engineering and Methodology*, 6(3), July 1997.
2. D.J. Barrett, L.A. Clarke, P.L. Tarr, and A.E. Wise. A Framework for Event-based Software Integration. *ACM Trans. on Software Engineering and Methodology*, 5(4):378–421, October 1996.

3. B. Boehm and W. Sherlis. Megaprogramming. In *Proc. DARPA Software Technology Conference*, Arlington, VA, April 1992.
4. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Deriving Active Rules for Workflow Management. In *Proc. 7^{th} DEXA*, Zurich, Switzerland, September 1996.
5. P. Constantopoulos, M. Jarke, J. Mylopoulos, and Y. Vassiliou. The Software Information Base: A Server for Reuse. *VLDB Journal*, 4(1):1–43, 1995.
6. G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS. Technical report, CEFRIEL, Politecnico di Milano, 1998.
7. S. Dami, J. Estublier, and M. Amiour. APEL: a Graphical yet Executable Formalism for Process Modeling. In E. Di Nitto and A. Fugetta, editors, *Process Technology*. Kluwer Academic Publishers, 1997.
8. A. Geppert and D. Tombros. Event-Based Distributed Workflow Execution with EVE. In *Proc. IFIP Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, England, September 1998.
9. S. Henninger. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. *ACM Trans. on Software Engineering and Methodology*, 6(2), April 1997.
10. C.W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(2):131–183, 1992.
11. J. Mylopoulos, A. Gal, K. Kontogiannis, and M. Stanley. A Generic Integration Architecture for Cooperative Information Systems. In *Proc. 1^{st} CoopIS*, Brussels, Belgium, June 1996.
12. O. Nierstrasz and L. Dami. Component-Oriented Software Technology. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 3–28. Prentice Hall, London, 1995.
13. The Common Object Request Broker: Architecture and Specification. Revision 2.0. Object Management Group, July 1995.
14. M. Papazoglou, A. Delis, A. Bouguettaya, and M. Haghjoo. Class Library Support for Workflow Environments and Applications. *IEEE Transactions on Computers*, 46(6), June 1997.
15. R. Prieto-Diaz and P. Freeman. Classifying Software for Reusability. *IEEE Software*, 4(1), 1987.
16. M.T. Schmidt. The Evolution of Workflow Standards. *IEEE Concurrency*, June 1999.
17. H. Schuster, S. Jablonski, P. Heinl, and C. Bussler. A General Framework for the Execution of Heterogenous Programs in Workflow Management Systems. In *Proc. 1st CoopIS*, Brussels, Belgium, June 1996.
18. Sun Microsystems. *Enterprise JavaBeans Specification Version 1.0*, March 1998.
19. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1997.
20. D. Tombros, A. Geppert, and K.R. Dittrich. Semantics of Reactive Components in Event-Driven Workflow Execution. In *Proc. CAiSE*, Barcelona, Spain, June 1997.
21. G. Valetto and G. Kaiser. Enveloping Sophisticated Tools into Computer Aided Software Engineering Environments. *Journal of Automated Software Engineering*, 3(3-4), 1996.
22. J. Widom and S. Ceri. *Active Database Systems*. Morgan Kaufmann, 1996.