Workflow Management Facility

Convenience Document combining

bom/98-06-07 and bom/98-07-15 (Errata)

March 9, 1999

Joint Submission by:

- ***CoCreate Software***
- ***Concentus***
- ***CSE Systems***
- ***Data Access Technologies***
- ***Digital Equipment Corporation***
- ***DSTC***
- ***EDS***
- ***FileNet Corporation***
- ***Fujitsu Limited***
- ***Hitachi Ltd.***
- ***Genesis Development Corporation***
- ***IBM Corporation***
- ***ICL Enterprises***
- ***NIIIP Consortium***
- ***Oracle***
- ***Plexus - Division of BankTec***
- ***Siemens Nixdorf Informationssysteme***
- ***SSA***
- ***Xerox***

OMG Document Number: bom/99-03-01

Preface

This submission is based on standards defined by the Workflow Management Coalition (WfMC). Founded in 1993, the WfMC is a non-profit, international organization of workflow vendors, customers and users whose mission is to promote the use of workflow through the establishment of standards for software terminology, interoperability and connectivity between workflow products. With more than 200 members in 25 countries, the Coalition has quickly become established as the primary standards body for this rapidly expanding software market.

The technology submitted in this proposal is directly based upon the WFMC standards for workflow interfaces, which have been available in the public domain for a number of years (see [3], [4], [6]) and provide a stable base for the introduction of workflow technology into the OMG architecture. The WfMC has approved the use of these standards by the submitters of this proposal. As an industry consortium the WfMC is not able formally to act as a submitter but fully endorses this submission as a supporter.

In 1996 the WfMC released an OMG IDL binding for its Client Application Programming Interface and have been developing a similar binding for the Interoperability interface (see [3], [7]). This submission is based directly on those bindings.

It is the intention of the submitters of this proposal that there be one Workflow Management Facility IDL specification endorsed by both the WfMC and the OMG. In addition to the standards incorporated in this submission the WfMC is actively engaged in the development of standards in related areas of workflow management, e.g., specifications for process definition and organization models. The submitters intend to make such specifications available to OMG as and when appropriate.

Supporters of this submission include:

- Workflow Management Coalition (WfMC)
- Abbott McCarthy
- Action Technologies
- Baan Company NV
- Computron Software, Inc.
- COSA SOLUTIONS Standardsoftware GmbH
- Eastman Software, Inc.
- Fuego Technology Corp.
- GLS Conseil
- Hatton Blue Ltd.
- Hewlett Packard
- IABG
- IDS Prof. Scheer GmbH
- IMA
- InConcert Inc.
- Meta Software
- Netscape
- Optika Imaging Systems, Inc.
- Sema Group Sae
- SNS Shared Network Systems Inc.
- TDI
- US DoD Defense Information Systems Agency
- Xedoc Software Development, Inc.

Contact Information

Submitters:

CoCreate Software: Dan Matheson, danm@fc.hp.com
Concentus: Robert Casto, casto@concentus-tech.com
CSE Systems: Emma Preininger, empr@csesystems.com
Data Access Technologies: Cory Casanave, cory-c@dataaccess.com
Digital Equipment Corporation: Manfred Koethe, Koethe@kar.dec.com
DSTC: Zoran Milosevic, zoran@dstc.edu.au
EDS: Fred Cummins, cummins@eng.eds.com
FileNet Corporation: Mike Marin, mmarin@filenet.com
Fujitsu Limited: Adel Ghoneimy, adelg@fsc.fujitsu.com
Hitachi Ltd.: Hiroshi Majima, majimahi@soft.hitachi.co.jp
Genesis Development Corporation: Robert Kosakowski, robkoz@gendev.com
IBM Corporation: Marc-Thomas Schmidt, mts@us.ibm.com
ICL Enterprises: Mike Anderson, mja@process.icl.co.uk
NIIIP Consortium: David Zenie, zenie@us.ibm.com
Oracle Corporation: Jim Trezzo, jtrezzo@us.oracle.com
Plexus - Division of BankTec: Richard Hemmerling, hemmerling@plx.com
Siemens Nixdorf Informationssysteme: Franz Klute, klute.pad@sni.de
SSA: Gene Glaudell, gglaudel@ssax.com
Xerox: Van-Si Nguyen, Vansi_Nguyen@wb.xerox.com

Supporters:

Abbott McCarthy: Dennis McCarthy, mccarthy@acm.org
Action Technologies: Raul Medina-Mora, Raul_Medina-Mora@actiontech.com
Baan Company NV: Sjaak Brinkkemper, sbrinkkemper@baan.nl
Computron Software, Inc.: Steve Dworkin, sdworkin@ctronsoft.com
COSA SOLUTONS Standardsoftware GmbH: Thomas Basting, tb@cosa.de
Eastman Software, Inc.: Mordechai Beizer, mordechai.beizer@eastmansoftware.com
Fuego Technology Corp.: Bruno Depascale, bruno@fuegotech.com
GLS Conseil: Gerard SACCONE, gsaccone@gls-conseil.fr
Hatton Blue Ltd.: Jon Iles, jon@hattonblue.com
Hewlett Packard: Ming Chien Shan, shan@hpl.hp.com
IABG: Barbal Wolters, bwolters@iabg.de
*I****DS Prof. Scheer GmbH***: Andreas Kronz, a.kronz@ids-scheer.de
IMA: Kian Saneii, saneiik@ima-inc.com
InConcert Inc.: Sunil Sarin, sarin@inconcert.com
Meta Software: Robert Shapiro, MetaRobert@aol.com
Netscape: Keith Swenson, kswenson@netscape.com
Optika Imaging Systems, Inc.: Robert Latham, rlatham@optika.com
Sema Group Sae: Agustin Gonzales-Quel, agonzales@sema.es
SNS Shared Network Systems Inc.: Rob Allen, rob.allen@uk.sns.ca
TDI: Nan Xiong, nan@tdiinc.com
US DoD Defense Information Systems Agency: Dan Wu, wud@ncr.disa.mil
Workflow Management Coalition: wfmc@wfmc.org
Xedoc Software Development, Inc.: J. Matthew Pryor, jmp@xedoc.com

Table of Contents

Introduction and Overview 8
- Guide to the Submission 8
- Scope 8
- Design Rationale 8
  - Workflow 9
  - Workflow Management Systems 9
  - The Workflow Reference Model 12
- Proof of Concept 13
- Resolution of Mandatory RFP Requirements 14
  - Workflow Metamodel 14
  - Interfaces for Workflow Enactment 14
  - Workflow Monitoring 15
  - Workflow Audit Trail 15
  - Nesting of Workflows 16
- Resolution of Optional Requirements 17
  - Definition of Workflow Schemas 17
  - Nesting of Workflow Schemas 17
  - Support for Ad-hoc Workflows 17
  - Level-of-service Parameters 18
- Discussion Issues 18
  - Transactions 18
  - Archiving of Workflow Instances 19
  - Resource Selection 19
  - Computing Resource Management 19
  - Workflow Integrity 20
  - Security of the Workflow Management Facility 20
  - Relationship to Envisioned OMG Technology 21
  - Virtual Enterprise 23
- Relation to Existing Standards 23
  - Workflow Management Coalition 23
  - WfMC Application Client specification 23
  - WfMC Audit Data specification 24
  - WfMC Interoperability specification 24
- References 24

Specification 26
- Overview 26
  - Workflow Interfaces 26
  - Process Enactment 27
  - Process Monitoring 28
- WorkflowModel module 29

Data Structures 29
Exceptions 30
Patterns 33
WfRequester 32
IDL 33
Relationships 33
Operations 34
WfExecutionObject 34
IDL 34
Attributes 36
States 38
Relationships 41
Operations 41
WfProcessMgr 43
IDL 43
Attributes 45
Relationships 46
States 47
Operations 47
WfProcess 48
IDL 49
Attributes 50
Relationships 51
Operations 52
WfProcessIterator 52
WfActivity 52
IDL 54
Attributes 55
Relationships 56
Operations 56
WfActivityIterator 57
WfAssignment 57
IDL 57
Relationships 58
States 59
WfAssignmentIterator 59
WfResource 59
IDL 59
Attributes 60
Relationships 61
Operations 61
WfEventAudit 62

IDL 62
Attributes 63
Relationships 65
WfEventAuditIterator 65
Publication via Notification Service 65
WfCreateProcessEventAudit 66
IDL 66
Attributes 67
Publication via Notification Service 67
WfStateEventAudit 68
IDL 68
Attributes 68
Publication via Notification Service 68
WfDataEventAudit 69
IDL 69
Attributes 69
Publication via Notification Service 70
WfAssignmentEventAudit 70
IDL 70
Attributes 71
Publication via Notification Service 71
The WfBase Module 72
Data Types 73
Exceptions 74
Base Business Object Interfaces 75
BaseBusinessObject 75
BaseIterator 75
IDL 75
Attributes 76
Operations 76
Interface Usage Example 78
Additional Information 80
Summary of Optional versus Mandatory Interfaces 80
Proposed Compliance Points 80
Changes or Extensions to Adopted OMG Specifications 80
Complete IDL Definitions 81
Consolidated IDL 81
CDL 90

Introduction and Overview 1

This document specifies interfaces for interaction with the execution environment of a Workflow Management Facility. The specified interfaces allow for a broad spectrum of implementations, including implementations based on existing workflow engines.

1.1 Guide to the Submission

The submission is structured as follows:

Section 1 provides an overview of the specification. It includes a brief introduction to Workflow in general and Workflow Management in particular and describes the scope of the submission. It also addresses RFP requirements and discussion issues.

Section 2 provides a detailed description of the proposed specification.

Section 3 provides additional information including conformance criteria and a complete list of the IDL for the specification.

1.2 Scope

This submission addresses the requirements for interoperability between different workflow object implementations in a CORBA environment. The interfaces specified are intended to be sufficiently general to support a wide range of application domains. These domains include healthcare, electronic commerce, manufacturing, insurance, finance, transportation, printing and publishing.

1.3 Design Rationale

Workflow Management (WfM) is a fast evolving technology which is increasingly being exploited by businesses in a variety of industries. Its primary characteristic is the automation of processes involving combinations of human and machine-based activities, particularly those involving interaction with information technology (IT) applications and tools. Although its most prevalent use is within the office environment in staff intensive operations such as insurance, banking, legal and general administration, it is also applicable to complex, dynamic environments such as design, engineering, and manufacturing.

Many software vendors have WfM products available today with WfM technology and there is a continuous introduction of more products into the market. The availability of a wide range of products within the market has allowed individual product vendors to focus on particular functional capabilities and users have adopted particular products to

meet specific application needs. However, there are no object-oriented frameworks to enable different WfM products and workflow aware applications to work together, which is resulting in incompatible “islands” of process automation.

This Workflow Management Facility specification addresses this problem, by introducing a workflow framework and interfaces that have been developed by a large group of workflow vendors and users under the umbrella of the Workflow Management Coalition (WfMC). This specification is based on the WfMC reference model and architecture.

It has been recognized that all WfM products have some common characteristics, enabling them potentially to achieve a level of interoperability through the use of common standards for various functions. The WfMC has been established to identify these functional areas and develop appropriate specifications for implementation in workflow products. It is intended that such specifications will enable interoperability between heterogeneous workflow products and improved integration of workflow applications with other IT services such as electronic mail and document management, thereby improving the opportunities for the effective use of workflow technology within the IT market, to the benefit of both vendors and users of such technology.

This section describes the basic functionality of the Workflow Facility. Most of this information is extracted from the WfMC Workflow Reference Model document [1].

1.3.1 Workflow

Workflow is concerned with the automation of procedures where information and tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal. Whilst workflow may be manually organized, in practice most workflow is normally organized within the context of an IT system to provide computerized support for the procedural automation.

Workflow is often associated with Business Process Re-engineering, which is concerned with the assessment, analysis, modeling, definition and subsequent operational implementation of the core business processes of an organization (or other business entity). Although not all BPR activities result in workflow implementations, workflow technology is often an appropriate solution as it provides separation of the business procedure logic and its IT operational support, enabling subsequent changes to be incorporated into the procedural rules defining the business process. Conversely, not all workflow implementations necessarily form part of a BPR exercise, for example implementations to automate an existing business procedure.

1.3.2 Workflow Management Systems

A Workflow Management System provides procedural automation of a business process by management of the sequence of work activities and the invocation of appropriate human and/or IT resources associated with the various activity steps.

An individual business process may have a life time ranging from minutes to days (or even months and years), depending upon its complexity and the duration of the various constituent activities.

At the highest level, all WfM systems may be characterized as providing support in three functional areas:

- the Build-time functions, concerned with defining, and possibly modeling, the workflow process and its constituent activities
- the Run-time control functions concerned with managing the workflow processes in an operational environment and sequencing the various activities to be handled as part of each process
- the Run-time interactions with human users and IT application tools for processing the various activity steps

The diagram below illustrates the basic characteristics of WfM systems and the relationships between these main functions

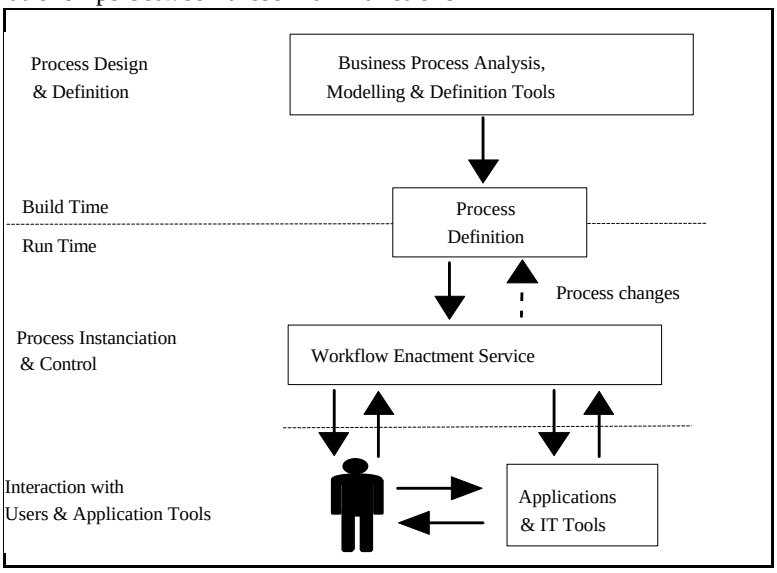

Figure 1-1 Workflow System Characteristics

Build-time Functions

The Build-time functions are those which result in a computerized definition of a business process. During this phase, a business process is translated from the real world into a formal, computer prosecutable definition by the use of one or more analysis, modeling and system definition techniques. The resulting definition is sometimes called a process model, a process template, process meta data, or a process definition. For purposes of this document, the term 'process definition' will be used.

A process definition comprises a number of discrete activity steps, with associated computer and/or human operations and rules governing the progression of the process through the various activity steps. The process definition may be expressed in textual or graphical form or in a formal language notation. Some workflow systems may allow dynamic alterations to process definitions from the run-time operational environment, as indicated by the feed-back arrow in the above diagram.

This Workflow Management Facility specification does not addressed the build-time functions. The specification concentrates on the run-time aspects of WfM.

Run-time Process Control Functions

At run-time the process definition is interpreted by software which is responsible for creating and controlling operational instances of the process, scheduling the various activities steps within the process and invoking the appropriate human and IT application resources, etc. These run-time process control functions act as the linkage between the process as modeled within the process definition and the process as it is seen in the real world, reflected in the runtime interactions of users and IT application tools. The core component is the basic workflow management control software, responsible for process creation & deletion, control of the activity scheduling within an operational process and interaction with application tools or human resources. This WfM software is often distributed across a number of computer platforms to cope with processes which operate over a wide geographic basis.

Run-time Activity Interactions

Individual activities within a workflow process are typically concerned with human operations, often realized in conjunction with the use of a particular IT tool (for example, form filling), or with information processing operations requiring a particular application program to operate on some defined information (for example, updating an orders database with a new record). Interaction with the process control software is necessary to transfer control between activities, to ascertain the operational status of processes, to invoke application tools and pass the appropriate data, etc. There are several benefits in having a standardized framework for supporting this type of interaction, including the use of a consistent interface to multiple workflow systems and the ability to develop common application tools to work with different workflow products.

Distribution & System Interfaces

The ability to distribute tasks and information between participants is a major distinguishing feature of workflow runtime infrastructure. The distribution function may operate at a variety of levels (workgroup to inter-organization) depending upon the scope of the workflows; it may use a variety of underlying communications mechanisms (electronic mail, messaging passing, distributed object technology, etc.). An alternative top-level view of workflow architecture which emphasizes this distribution aspect is shown in the diagram below.

The workflow enactment service is shown as the core infrastructure function with interfaces to users and applications distributed across the workflow domain. Each of these interfaces is a potential point of integration between the workflow enactment service and other infrastructure or application components.

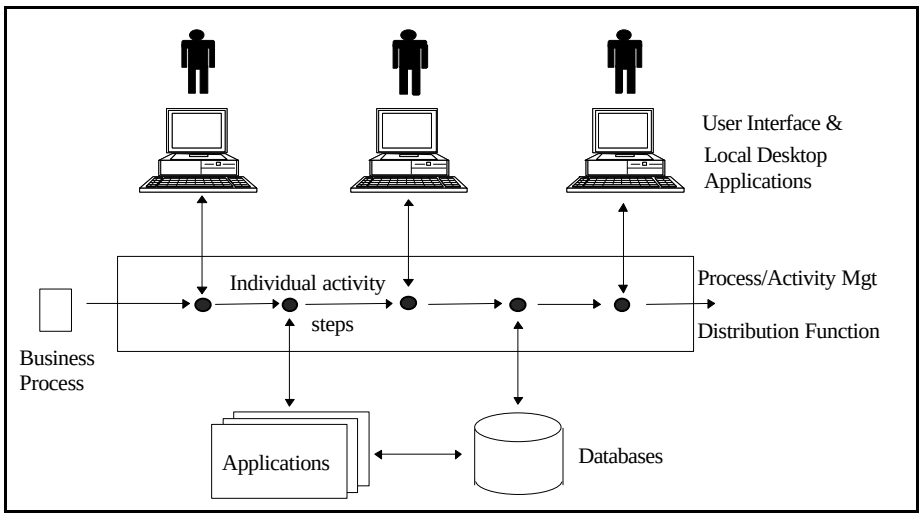

Figure 1-2 Distribution within the WfM Service

The flow of work may involve the transfer of tasks between different vendors implementations of the workflow management facility to enable different parts of the business process to be enacted on different platforms or sub-networks using particular products suited to that stage of the process. In this scenario the flow within the central box passes between two or more workflow products - for example activities 1,2 and 5 may be executed by one workflow system and activities 3 and 4 by a different system, with control passed between them at appropriate points within the overall workflow. Standards to support this transfer of workflow control enable the development of composite workflow applications using several different implementations of the WfM Facility operating together as a single logical entity.

1.3.3 The Workflow Reference Model

The Reference Model identifies the functional areas addressed by the Workflow Management Facility and typical usage scenarios:

- **Process Definition**: specifications for process definition data and its interchange with the Workflow Execution environment.
- **Workflow Interoperability**: interfaces to support interoperability between different workflow systems
- **Invoked Applications**: interfaces to support interaction with a variety of IT application types
- **Workflow Client Applications**: interfaces to support interaction with user interface desktop functions

- **Administration and Monitoring**: interfaces to provide system monitoring and metric functions to facilitate the management of composite workflow application environments.

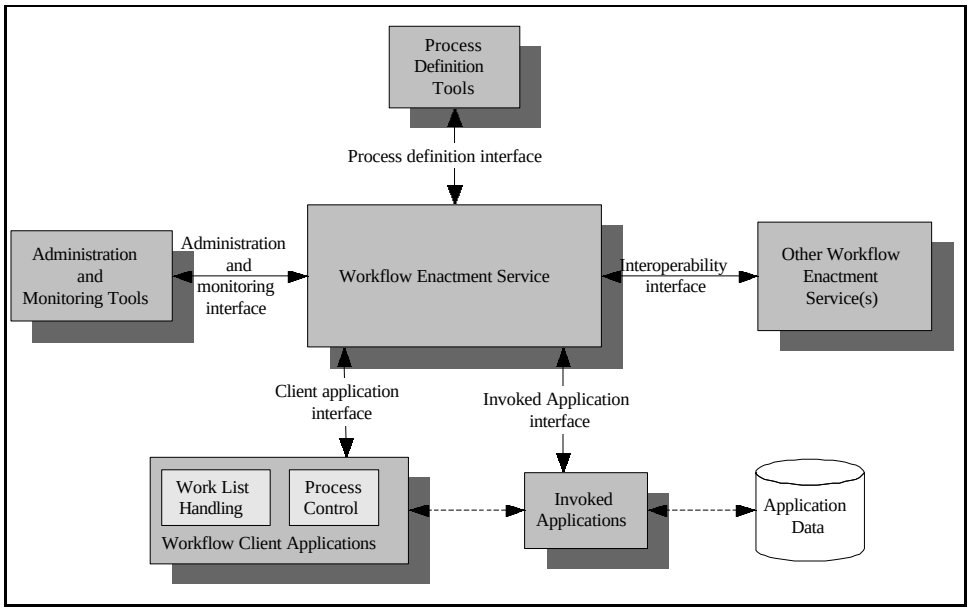

Figure 1-3 Workflow Reference Model

Scope of Specification

The current specification defines the interfaces that support Workflow Client Applications, Interoperability and Process Monitoring as described in the Reference Model. Specification of Process Definition interfaces as well as Administration interfaces should be addressed by a future RFP.

1.4 Proof of Concept

The specification is based on the standards defined by the Workflow Management Coalition (WfMC). These standards have been implemented in a number of WfM products, such as IBM FlowMark, Hitachi Groupmaxup, Concentus KI Shell, CSE/WorkFlow, Fujitsu TeamWARE Flow, FileNet Workflow Connect (prototypes also available for Ensemble and Visual WorkFlo), CoCreate WorkManager, DIGITAL ProcessManager and LinkWorks, TDI WebDeploy Workflow, ICL RoleModel (prototype), Action Technologies ActionWorkflow (prototype), COSA SOLUTIONS COSA (under development), Baan Company NV Enterprise Workflow (under development), COI VisualFloware, Optika Imaging Systems PowerFlow.

WfMC has been defining OMG IDL versions of their specifications since 1995. Early versions of the specification have been prototyped by the NIIIP consortium using existing workflow products such as IBM FlowMark and Concentus KI Shell, by DIGITAL ProcessManager, SNS LiveWorkFlow (under development), by CSE/WorkFlow, and by Fuego Technology Corp. Fuego Engine.

EDS has developed several workflow systems and has worked with a number of workflow management products. As a result of this experience, EDS has a diversity of experience and understanding of workflow management requirements and the potential that this facility offers to users of the technology to achieve a new level of enterprise automation. In addition, EDS has developed a Business Objects Facility and is in the process of implementing large-scale, commercial applications of this technology.

1.5 Resolution of Mandatory RFP Requirements

1.5.1 Workflow Metamodel

Requirement: Submissions shall provide a complete semantic definition of their workflow metamodel (expressed in some well known notation) that includes a description of the relevant concepts that lead to the submitted interfaces, their relationships with each other, and the life cycle of the instances of these interfaces.

Section 2 of this document describes the interfaces of the WfM Facility with a UML diagram of the model, IDL interfaces and description of the semantics of these interfaces.

1.5.2 Interfaces for Workflow Enactment

Requirement: Submissions shall specify a set of interfaces for workflow enactment, i.e., the execution of workflow instances.

This submission defines interfaces for associating resources with workflow objects (*Assignment* and *Resource*), requesting execution, state transitions, and associating process data (*Requester*, *Process*, and *Activity*), and receiving and maintaining a history of events (*EventAudit* and its subtypes). See Section 2 for details.

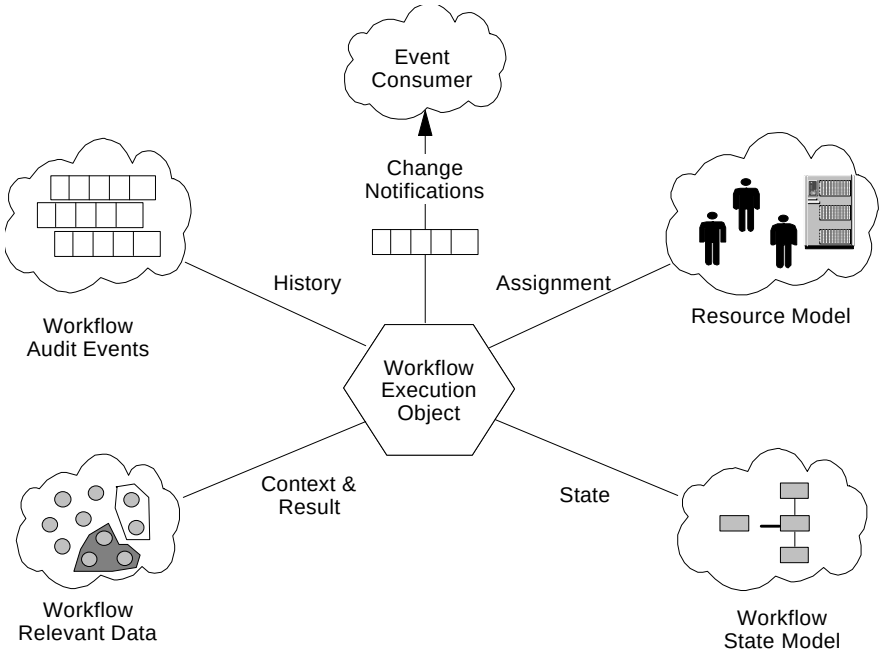

Figure 1-4 Workflow Execution Object

1.5.3 Workflow Monitoring

Requirement: Submissions shall provide interfaces for retrieving information about the status of workflows.

Support is provided with operations for navigation of relationships between workflow objects, access to their runtime information, and with notification of changes. This support can be used for monitoring by both push and pull modalities. *Process* provides information on the overall status of a workflow process and *Activity* provides information of the status of particular steps in a workflow process.

1.5.4 Workflow Audit Trail

Requirement: Submissions shall provide interfaces for retrieving the history of workflow execution. This history could include identification of the parties triggering work and the resources performing the work as well as the work done.

A history is kept of workflow execution events. Each history item records the time, resource, type, and details of the event. Operations are defined for retrieving the history of status changes of *Processes* and *Activities*.

1.5.5 Nesting of Workflows

Requirement: Submissions shall provide interfaces that support accessing nested workflows for purposes of the above functionality (deemed to be workflow enactment, workflow monitoring, extraction of workflow event audit data).

This submission supports nesting of *Processes*. When a *Process* is created, a *Requester* is associated with it; the *Requester* provides a means for the *Process* to signal status changes, most notably its completion, back to the originator of the *Process*. This can be used to realize various workflow nesting scenarios. In all scenarios a sub-*Process* is created as part of the enactment of a main *Process* (using an appropriate process manager factory); an appropriate *Requester* is registered with the sub-process to enable synchronization of the two processes if required. The following describes some examples.

Nested sub-process

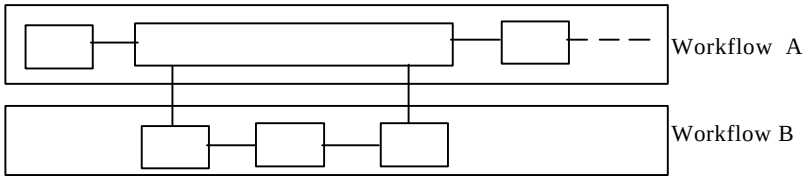

Figure 1-5 Nested workflow structure

In nested sub-process workflow structures, one workflow may invoke another as the performer of an *Activity* and the activity waits for the sub-process to complete. Note that an *Activity* is a *Requester,* and the Activity that is realized by the sub-processes can serve as the synchronization point for interaction of the two workflows.

It is also possible to create the sub-process at a particular process step (i.e., *Activity A*) and to register another *Activity (B)* as the synchronization point (i.e., *Requester*). The Workflow A would proceed until it reached *Activity B,* then Workflow A would wait until Workflow B completed. Since *Activity B* is the *Requester* for Workflow B it would receive the completion event.

Chained processes

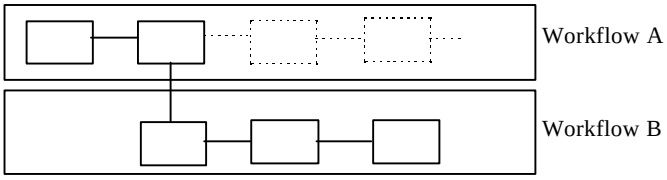

Figure 1-6 Chained workflow structure

In chained workflow structures, one *Process* may invoke another, then carry on with its own flow logic. The *Processes* terminate independently of each other; in this case the *Requester* registered with the sub-process would be another entity that is interested in the results of the sub-process; it might actually be the same *Requester* that was registered with the main *Process*.

1.6 Resolution of Optional Requirements

Optional Requirement: The secondary focus of this RFP is for workflow schema definitions providing access to metadata about workflow instances.

Process Definition will be addressed when sufficient consensus is reached on a standard format for exchanging workflow definitions. This specification concentrates on the run-time aspects of the Workflow Management Facility.

1.6.1 Definition of Workflow Schemas

Optional Requirement: Submissions may describe interfaces that enable creation, retrieval and maintenance of workflow schemas.

The WfMC document Process Definition Interchange (see [5]), addresses workflow schema definitions. The goal of the Process Definition Interchange interface is to allow for the interchange of process definitions between different workflow management systems. The WfMC is in the process of completing that interface.

1.6.2 Nesting of Workflow Schemas

Optional Requirement: Submissions may describe interfaces that enable nesting of workflow schemas within other workflow schemas either by inclusion or reference.

The WfMC proposal for Process Definition Interchange (see [5]) propose that workflow schemas may reference each other as a means of effecting nested schemas.

1.6.3 Support for Ad-hoc Workflows

Optional Requirement: Submissions may outline potential extensions to support ad-hoc workflows, e.g., the ability to add or modify workflow schemas associated with existing workflow instances.

At run-time the process definition is interpreted by software, scheduling the various steps within the process and invoking the appropriate human and IT resources. There exist situations however, where it is either not possible or desirable to obtain a process definition which is detailed enough to be formalized and used as a suitable process definition. There can be several reasons for the lack of a proper process definition, e.g. the lack of detailed knowledge about the process or the business problem, unavailability of the required resources to perform proper analysis and design of a process, lack of tools or other resources, time or economical constraints, etc. In other

cases the problem of combinatorial explosion might prevent that a process definition can be created which is precise enough to be used for a workflow system, e.g. when all potential exceptions within a business process have to be considered and implemented.

In these or similar cases a workflow system shall allow, and optional and future extensions to the suggested standards shall support, the creation of activities on an ad-hoc basis, i.e. based on spontaneous interaction of a workflow participant. Activities may thus be created ad-hoc, i.e. without pre-definition in a process definition.

Security and authorization features of the Workflow Management System may control or influence, when, how and to which degree ad-hoc creation of activities may occur. Ad-hoc activities may result in simple operations or imply the execution of a sub-process, based on any process definition available to the system.

This specification does not prevent this type of ad-hoc behavior. *Activities* can be added to or removed from active *Processes*; however, this specification does not describe interfaces to achieve modifications of workflow process models. Reassignment of resources is supported.

1.6.4 Level-of-service Parameters

Optional Requirement: Submissions may allow the specification of level-of-services parameters that influence workflow enactment, for example: guaranteed completion of a process instance, time-box execution, optimized resource / cost usage.

Level of service parameters are set in the definition of a workflow process model. This specification does not address workflow process definition.

Level of service parameters can be included in the process context, and passed to a Process when it is initialized; we do not, however, define specific level of service parameters at this time.

1.7 Discussion Issues

1.7.1 Transactions

Discussion Issue: It should be discussed how the submission uses the concept of transactions, the underlying transaction-model and its dependency on the OMG Transaction Service.

Operations on workflow objects specified in this submission are performed within transactions. These operations can trigger interactions with other workflow objects. For example, when a *Process* is started, some (or all) of its *Activities* will be created; when a sub-process signals completion to its *Activity* this will trigger evaluation of business rules in the main process to determine the next set of *Activities* to create.

Implementations of this facility will ensure the scope of transactions is limited, complex status changes (as described above) will be split into a sequence of short transactions and the WfM implementation will queue consequential transactions in a recoverable manner. We also expect that an out-of-domain *Requestor* could use a time-out capability to resume if a result never comes back.

1.7.2 Archiving of Workflow Instances

Discussion Issue: Submissions should describe adequate archiving policies and mechanisms for reconstruction of the workflow execution sequence.

The execution history of workflow instances is recorded using *EventAudit* data. Archiving of workflow instances can be realized using this information; the specification of an archiving facility is beyond the scope of this submission.

1.7.3 Resource Selection

Discussion Issue: Submissions should explain how workflows are bound to organizations, people and other resources during their enactment. Modeling of organizations, people and other resources is not within the scope of this RFP; it is being addressed by other OMG specifications and RFPs.

Resource selection is a mechanism for connecting workflows to needed resources, including people and organizations. An *Assignment*, for example, describes the association of a *Resource* to a particular work activity.

Workflow entities are associated with system, information and people resources via operations on these entities. Associations may be abstract and their resolution may occur at any point in the process. For example, people may be specified abstractly as roles which are then resolved to actual people when an activity executes.

A detailed specification of resource assignment is beyond the scope of this submission. However, realization of resource assignment with a specialization of the *Assignment* interface, may be possible. The following sketches the basic concepts:

Certain activities in a process will request the assignment of needed resources. The requests would be made to resource management processes. The resource management work process would provide a generalized resource allocation mechanism. Resources may be human participants, reusable facilities such as machines or consumable goods such as parts and fluids. The resource manager may place work requests in work lists, or assign resources with specialized algorithms. The specific mechanisms and selection options will be left to the implementation.

1.7.4 Computing Resource Management

Discussion Issue: Submissions should address scalability. However, specification on how to manage computing resources is not within the scope of this RFP.

The model proposed is designed to facilitate scalable implementations.

1.7.5 Workflow Integrity

Discussion Issue: Submissions should discuss how the integrity of workflows is maintained. This includes failure recovery and rollback after system failures

The granularity of the workflow entities defined in this submission is such that they may take advantage of an underlying persistence service. The states and their transitions are defined at a low level of granularity simplifying the recoverability of the workflow entities' persistent state. The history capabilities in this submission provide information to facilitate the recovery and rollback after system failure.

1.7.6 Security of the Workflow Management Facility

Discussion Issue: Submissions should discuss how the Workflow Management Facility can be made secure including controlling access to workflows, workflow information (e.g. the Audit Trail) and workflow schemas. Submissions should describe their use of the OMG Security Service.

All objects are security sensitive and all operations are subject to security policy control. The objects in this model are security aware and will pass on security credentials when issuing requests. Default policies for security sensitive objects are not defined and there are no special security considerations introduced.

From CORBA Services Security Section 15.3.2:

"An active entity must establish its rights to access objects in the system. It must either be a principle, or a client acting on behalf of a principle. A principle is a human user or system entity that is registered in and authentic to the system."

To meet this security requirement the process requester must be a principle or a client acting on behalf of a principle to form the basis for security delegation considerations and to achieve the access control needed.

Relative to CORBA Services Security Section 15.3.4, "Access Control Model", there are two important parts to the Access Control Model, the object invocation access policy and the application access policy.

The object invocation access policy is outside the scope of this specification. However, for the object invocation access policy to work there must to control attributes on the object. One extreme is to allow all access. For most situations some minimum level of granularity of object operations should be supported for interoperability.

The application access policy is outside the scope of this specification, and no policy is specified or required. The various implementations of this standard should provide whatever application access policy is needed for their customers and market.

Relative to CORBA Services Security Section 15.3.5, this specification does not list any required application security relevant events. The various implementations of this standard should provide whatever security relevant events are needed for their customers and market.

Relative to CORBA Services Security Section 15.3.6, “Delegation”, the Workflow objects will often be “intermediate” objects in the Security Specification vocabulary. As intermediate objects the Workflow objects should be able to support at least “combined privilege delegation” and in higher accountability situations the Workflow objects should support “traced delegation”.

Relative to CORBA Services Security Section 15.3.7, “Non-repudiation”, as long as a Workflow process executes with in a single security domain, the need for non-repudiation is usually low. When a Workflow process executes across security domains (which is also across business domains in most cases), then non-repudiation services could be required. The Security Service Specification states that “The non-repudiation services specified here are under control of the applications”. The need for non-repudiation services is defined by the specific application situation, therefore this specification does not mandate the use of any non-repudiation services.

Relative to CORBA Services Security Section 15.3.8, “Domains”, workflow processes may have several hierarchical levels. For practical security purposes the process instance and its immediate activities should be within the same security policy domain. Having the process definition, the requester and responsible participants in the same security policy domain as the process instance and activity instances will reduce the security overhead. Sometimes Workflows will exist across security domains. This specification does not mandate that all the objects involved within a Workflow exist within the same security policy domain.

1.7.7 Relationship to Envisioned OMG Technology

Submissions should discuss how they relate to the following envisioned OMG technologies:

Business Object Facility

Discussion Issue: OMG Business Object Facility (BOF): Business Objects are the building blocks of future applications. As it is the major goal of today’s workflow management systems to interconnect applications, the Workflow Management Facility will have to interconnect business objects in the future. As a result, the BOF and the Workflow Management Facility may have to work together in a very tight manner.

The specification includes a description of the workflow model in Boca CDL. It may be implemented with Business Object Facilities that support the Boca architecture. Workflow management systems should be able to directly integrate business objects within a workflow process and adapt related organization and resource components used in the performance of workflow activities.

At the same time, these specifications are designed to incorporate existing workflow systems such that these systems can interoperate with each other, outside of a BOF environment.

Rule Management Facility

Discussion Issue: OMG Rule Management Facility - may be used to support the behavioral aspect by implementing global constraints that have to be verified during workflow execution. Workflows may be started as the implementation of the “Action” Part of ECA-Rules (Event-Condition-Action).

This is not considered relevant until consensus is reached on process definition in the workflow community and until there is an OMG Rule Management Facility.

Meta-Object Facility

Discussion Issue: OMG Meta-Object Facility (MOF) – supports the definition of semantically rich metadata. Since the Workflow Management Facility needs to manage workflow schemas, it may use MOF interfaces to create them. In addition, the workflow metamodel of the Workflow Management Facility itself should be representable in the MOF.

Exploitation of the MOF is considered to be more important in the area of workflow process definition which is not addressed by this submission.

Change Management Service

Discussion Issue: OMG Change Management Service - might be used to provide for the evolution of workflow schemas and related workflow instances.

The change management service could be used to manage workflow definition changes and to handle evaluation of work processes in ad-hoc scenarios. We do not address process definition and ad-hoc workflow in this specification.

Product Data Management Enablers

Discussion Issue: OMG Product Data Management Enablers: Submissions should describe how their interfaces can be used by PDM enablers.

The interfaces for requesting, managing, and querying workflow objects will provide support for PDM Enabler engineering change models. The *Resource* is essentially the same as the PDM *Actor*. This submission works well with the interfaces defined in the PdmChangeManagement module.

Objects by Value

Discussion Issue: OMG Objects by Value (OBV) will be one mechanism by which state information can be moved from one Workflow Management Facility to another or to other facilities.

The specification does not depend on the ObV specification.

1.7.8 Virtual Enterprise

Discussion Issue: Submissions should discuss how they support processes across virtual enterprises.

This specification particularly addresses the requirement for interoperability of workflow systems in virtual enterprises. It allows companies involved in a virtual enterprise to maintain the security and confidentiality of their processes within the macroscopic workflow of the virtual enterprise. The specification supports the dynamics of a virtual enterprise.

1.8 Relation to Existing Standards

1.8.1 Workflow Management Coalition

The Workflow Management Coalition (WfMC) is the standards consortium in the area of Workflow Management Systems. The Coalition has defined a Workflow Reference model and has published standards for the Workflow Application Client, the Workflow Interoperability interfaces and Audit Data formats. The specification described in this document is based on the existing WfMC standards.

OMG WfM Term	**WfMC Term**
WfRequester	Participant or Workflow Enactment Service
WfProcess	Process Instance
WfProcessMgr	WAPI Process Definition
WfActivity	Activity Instance
WfAssignment	Work List
WfResource	Participant
WfEventAudit	Audit Data

Table 1-1 Mapping to WfMC Concepts

1.8.2 WfMC Application Client specification

The WfMC Application Client specification defines operations that allow a Workflow Application Client to obtain lists of workflow runtime entities and to get and set their attributes and state.

The interfaces for *Process, Activity, Assignment* and *Resource* represent the entities defined in the WfMC specification and support the operations defined in the WfMC specification.

1.8.3 WfMC Audit Data specification

The WfMC Audit Data specification defines the status changes to be reported by a WfM system and the format of the Audit records to be produced when such a status change happens.

The format of the *EventAudit* and its subtypes reflects the format of Audit Data defined in the WfMC specification.

1.8.4 WfMC Interoperability specification

The WfMC interoperability specification defines interfaces that enable the collaboration of two WfM systems in the enactment of a complex workflow process. WfMS A may delegate enactment of a sub process to WfMS B; B may inform A about important status changes of the subprocess (including completion of the process) and A controls execution of the subprocess.

Interaction between a main workflow process and subprocesses is supported by the *Requester* interface inherited by *Activity* that allows an *Activity* in a (main) *Process* to interact with another (sub) *Process* that implements the *Activity*. The requester is registered with the sub-*Process* when it is created and the sub-*Process* will signal status changes (most notably its completion) back to the *Requester*.

1.9 References

Here is a list of the OMG and WfMC specifications used in this specification; the WfMC documents can be found at http://www.wfmc.org/wfmc/.

[1] The Workflow Reference Model, Version 1.1, November 1994, WfMC-TC-1003

[2] Terminology & Glossary, Version 2.0, June 1996, WfMC-TC-1011

[3] Workflow Client Application Programming Interface (WAPI) Specification, Version 1.2, October 1996, WfMC-TC-1009

[4] Workflow Interoperability - Abstract Specification, Version 1.0, October 1996, WFMC-TC-1012

[5] Process Definition Interchange, WfMC TC-1016

[6] Audit Data Specification, WfMC TC-1015

[7] Workflow Facility Specification, Draft WfMC TC-2101

[8] OMG Event Service, in CORBAservices: Common Object Services Specification, chapter 4

[9] OMG Life Cycle Service, in CORBAservices: Common Object Services Specification, chapter 6

[10] OMG Naming Service, in CORBAservices: Common Object Services Specification, chapter 3

[11] OMG Property Service, in CORBAservices: Common Object Services Specification, chapter 13

[12] OMG Security Service, in CORBAservices: Common Object Services Specification, chapter 15

[13] OMG Time Service, in CORBAservices: Common Object Services Specification, chapter 14

[14] OMG Trading Object Service, in CORBAservices: Common Object Services Specification, chapter 16

Specification 2

This submission specifies interfaces for workflow execution control, monitoring, and interoperability between workflows defined and managed independently of each other. The interfaces are based on a model of workflow objects which includes their relationships and dependencies with requesters, assignments, and resources. The core workflow interfaces are defined in the WorkflowModel module.

The model is graphically represented in UML class and object interaction diagrams, and specified by IDL interfaces. For each interface, its attributes, relationships, state set and its operations are described. Standard patterns are used for operations realizing relationships and access to attributes and object state.

In addition to the core workflow interfaces, we also include a simple 'framework' that is meant to be a placeholder for a 'Business Component Framework'; we expect a revision of this specification to align this placeholder with the result of the OMG efforts to define such a framework in the context of the BOF and the Component Model RFP. The framework is described in the WfBase module.

2.1 Overview

This section provides a brief overview on the core interfaces defined in this specification; it also discusses typical usage scenarios. The following sections describe the interfaces in detail. Section 2.18 contains more detailed usage scenarios.

2.1.1 Workflow Interfaces

The core workflow interfaces are:

- *WfRequester*, links the immediate owner of a request for a *WfProcess*, i.e., it receives significant events such as 'complete'
- *WfProcessMgr*, provides factory and location support for *WfProcess*
- *WfProcess*, is the performer of a workflow request issued by a user or automated actor such as *WfActivity* as a *WfRequester*
- *WfActivity*, is a step in a *WfProcess* and may also be a *WfRequester*
- *WfExecutionObject*, is an abstract base class for *WfProcess* and *WfActivity*
- *WfAssignment*, links activities to potential/actual *WfResources*
- *WfResource*, a person or thing that can do and accept a *WfActivity*.

- *WfEventAudit*, is a common interface for recording workflow events. Several subtypes of this interface are defined to record change of the state of a workflow object, process data associated with it, and change in the assignment of resources to *WfActivities*.

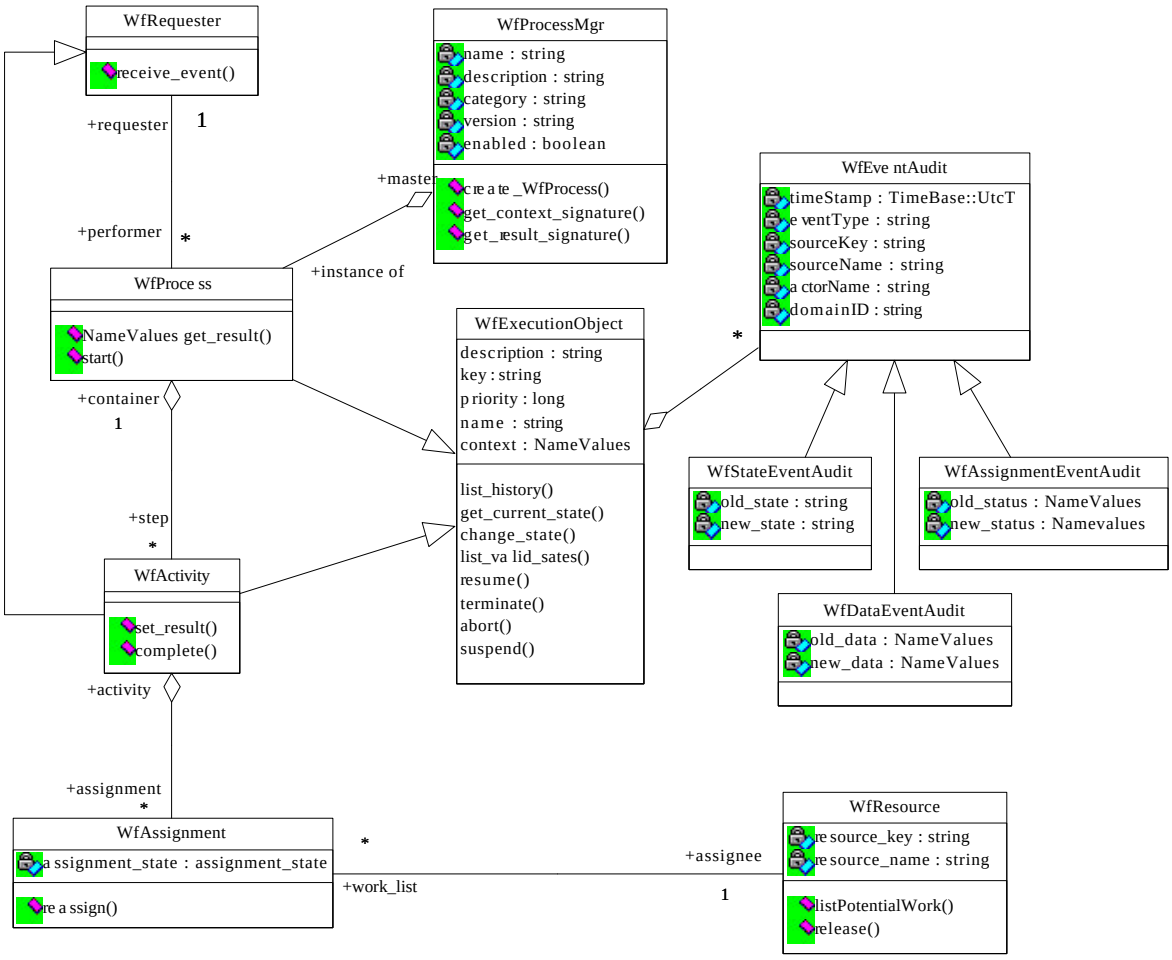

Figure 2-1 Joint Workflow Management Facility Model

2.1.2 Process Enactment

To initiate enactment of a particular workflow process, a Requester that is responsible for that Process would be identified; an existing Requester can be reused or a specific one that observes this Process can be created.

An appropriate Process Manager is identified and the Process is created using the create_process operation of that manager. The Requester is associated with the Process when it is created and will receive status change notifications from the Process. When the Process is instantiated it might create a set of Activities representing process steps in the Process.

The Process is initialized by setting its context data; context data may be used to parametrize a generic workflow process, identify resources to be used by the process, etc.

Enactment of the Process is initiated by invoking its start operation; the process implementation will use context data and built-in logic to determine which Activities are to be activated. It may also initiate other (sub-) Processes.

When an Activity is activated, its context is set and resources may be assigned to it by creating Assignments linking it to Resources; the resource selection mechanism is not defined here, but an implementation might, e.g., use another Process to determine which resources to assign to a particular Activity, using the Activity's context information and other process parameters.

An Activity might be implemented by another (sub-) Process, i.e., it can be registered as the Requester of that Process; the sub-process can be initiated when the Activity is activated. In this case, the Activity is completed when the sub-Process completes and the result of the Activity is obtained from the result of the Process.

An Activity can also be realized by an application that uses the Activity's set_result and complete operations to return results and signal completion of the Activity.

When an Activity is completed, its results will be used by the workflow logic to determine follow-on Activities; the results can also be used to determine the overall result of the Process it is contained in.

A Process is completed when there are no more Activities to be activated; it will signal its completion to the associated Requester. At this time, the results of the process are made available, intermediate results may be accessible while the Process is running.

2.1.3 Process Monitoring

The overall status of a *Process* can be queried using the state, get_context and get_result operations. The *Requester* associated with a *Process* also receives notifications about status changes of the *Process*.

More detailed information on the status of the process steps can be obtained be navigating the *step* relationship between *Process* and its *Activities* and using the status inquiries provided by the *Activity* interface. Navigation of nested workflows is supported by the *process* relationship between an *Activity* (which is a *Requester*) and potential sub-*Processes*.

Whenever an Execution Object (Process or Activity) performs a (workflow relevant) status change, an EventAudit is recorded. For each Execution Object, the history of Event Audit items can be accessed to analyze the execution history of that object. Event Audits might be published using the OMG Notification Service.

2.2 WorkflowModel module

The WorkflowModel module defines the core interfaces of the Workflow Management Facility.

```
#ifndef _WORKFLOW_MODEL_
#define _WORKFLOW_MODEL_
#include <WfBase.idl>
#pragma prefix "omg.org"
module WorkflowModel{

        // Forward declarations
            ...
        // Data Types
            ...
        // Exceptions
            ...
        // Interfaces

        interface WfRequester : WfBase::BaseBusinessObject{...};
        interface WfExecutionObject : WfBase::BaseBusinessObject {...};
        interface WfProcessMgr : WfBase::BaseBusinessObject {...};
        interface WfProcess :WfExecutionObject {...};
        interface WfProcessIterator : WfBase::BaseIterator {...};
        interface WfActivity  : WfExecutionObject, WfRequester{...};
        interface WfActivityIterator : WfBase::BaseIterator{...};
        interface WfAssignment  : WfBase::BaseBusinessObject{...};
        interface WfAssignmentIterator : WfBase::BaseIterator{...};
        interface WfResource : WfBase::BaseBusinessObject{...};
        interface WfEventAudit : WfBase::BaseBusinessObject{...};
        interface WfEventAuditIterator : WfBase::BaseIterator{...};
        interface WfCreateProcessEventAudit : WfEventAudit{...};
        interface WfStateEventAudit : WfEventAudit {...};
        interface WfDataEventAudit  : WfEventAudit {...};
        interface WfAssignmentEventAudit : WfEventAudit{...};
};
#endif
```

2.2.1 Data Structures

The WorkflowModel module defines the following data structures.

Workflow object sequences

```
typedef sequence<WfProcess> WfProcessSequence;
typedef sequence<WfActivity> WfActivitySequence;
typedef sequence<WfAssignment> WfAssignmentSequence;
typedef sequence<WfEventAudit> WfEventAuditSequence;
```

Sequences of workflow objects used for handling of relationship navigation.

Process Data

typedef WfBase::NameValueInfoSequence ProcesDataInfo;
typedef WfBase::NameValueSequence ProcessData;

Name-value pair sequences are used to handle process data associated with a WfExecutionObject; ProcessDataInfo describes the structure of these process data and ProcessData represents context and result data of an execution object. See the section on WfExecutionObject for details.

State sets

enum workflow_stateType{ open, closed };
enum while_openType{not_running, running };
enum why_not_runningType{ not_started, suspended };
enum how_closedType{ completed, terminated, aborted };
enum process_mgr_stateType{enabled, disabled };
enum assignment_stateType { potential, accepted };

These enumerations are used to describe sets of states of various workflow objects; see below for details.

2.2.2 Exceptions

The WorkflowModel module defines the following exceptions

exception InvalidPerformer{};

Is raised by an attempt to signal a WfEventAudit to a WfRequester that was not created by one of the WfProcesses associated with the WfRequester.

exception InvalidState{};

Is raised by an attempt to change the state of a WfExecutionObject to a state that is not defined for that object.

exception InvalidData{};

Is raised by an attempt to update the context of the result of a WfExecutionObject with data that do not match the signature of that object.

exception TransitionNotAllowed{};

Is raised by an attempt to perform an invalid state transition of a WfExecutionObject.

exception CannotResume{};
exception CannotSuspend{};
exception AlreadySuspended{};
exception CannotStop{};
exception NotRunning{};

These exceptions are raised by operations on a WfExecutionObject that attempt to perform invalid control operations on that object. See the section on WfExecutionObject for details.

exception HistoryNotAvailable{};

Is raised by a request for event audit history of a WfExecutionObject when the History is not available (i.e., because the implementation of the WfM Facility does not support recording of history for a specific execution object).

exception NotEnabled{};

Is raised by an attempt to create a WfProcess using a WfProcessMgr that is disabled.

exception AlreadyRunning{};
exception CannotStart{};

These exceptions are raised by an attempt to start a WfProcess that is already running or cannot be started yet.

exception ResultNotAvailable{};

Is raised when the requested result of a WfExecutionObject is not available (yet).

exception CannotComplete{};

Is raised by an attempt to complete execution of a WfActivity when it cannot be completed yet.

exception NotAssigned{};

Is raised by an attempt to release a WfResource from an assignment it is not associated with.

exception SourceNotAvailable{};

Is raised by a request for the source of a WfEventAudit when the source is no longer available.

2.2.3 Patterns

We use standard patterns to represent attributes and relationships of the workflow interfaces. All operations return a WfBase::BaseException CORBA exception in addition to the exceptions defined in this specification; see the discussion in the chapter on the WfBase module for details.

Attributes

The pattern for access operations on attributes is the following: for an attribute with name ATTRNAME and type TYPE, two operations are provided;

TYPE ATTRNAME();

returns the value of the attribute,

void set_ATTRNAME(in TYPE value)

supports updates of the attribute; the set operation is not provided for readonly attributes.

Relationships

The pattern for accessing cardinality 1 relationships is the same as for attributes. For relationships with cardinality 'many' the following pattern is applied: for a relationship with name RELNAME and type TYPE, the how_many_RELNAME() operation returns the number of elements in the relationship, get_iterator_RELNAME() returns a TYPEIterator, get_sequence_RELNAME(in long how_many) returns a TYPESequence and the is_member_RELNAME(in TYPE member) support checks for membership of an object in the relationship.

2.3 WfRequester

WfRequester is the interface that has a direct concern with the execution and results of a workflow process - it represents the request for some work to be done. Its performer, a *WfProcess,* is expected to handle its request and communicate significant status changes; in particular to inform the requester when it has completed performing the requested work. A single requester can have many processes associated with it.

Often *WfRequester* will also be the interface to the object that starts the process. As a process starter some of the control actions on the process include setting up the context, starting the process and getting results and status.

There are two usage scenarios for the association of a *WfProcess* with a *WfRequester*:

- Nesting of workflow processes. A *WfActivity* is a *WfRequester* and may therefore request that a *WfProcess* be its performer, i.e., implementation. In this case, the *WfActivity* would be registered as the requester with the implementing sub-process when the *WfProcess* is created and would receive notifications of status changes of that sub-process; upon completion of the sub-process, the *WfActivity* would enter *completed* state.

- Linking a workflow process to another (initiating or controlling) application. When used as a linked process the requester should be a *WfRequester* which is not the linking *WfActivity*. Requesters that are not activities are roles or adapters for external clients.

2.3.1 IDL

```
interface WfRequester : WfBase::BaseBusinessObject{

        long how_many_performer()
            raises (WfBase::BaseException);
        WfProcessIterator get_iterator_performer()
            raises (WfBase::BaseException);
        WfProcessSequence get_sequence_performer(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_performer(
            in WfProcess member )
            raises (WfBase::BaseException);

        void receive_event(
            in WfEventAudit event)
            raises (WfBase::BaseException, InvalidPerformer);
};
```

2.3.2 Relationships

Name	Type	Properties	Purpose
performer	WfProcess	cardinality: 0..n readonly	Associates work requests with their performers.

performer

Zero or more *WfProcesses* can be associated with a *WfRequester*. A requester is associated with a WfProcess when the process is created.

The following operations support the *performer* relationship with *WfProcess*.

```
long how_many_performer()
        raises (WfBase::BaseException);
WfProcessIterator get_iterator_performer()
        raises (WfBase::BaseException);
WfProcessSequence get_sequence_performer(
        in long max_number )
        raises (WfBase::BaseException);
boolean is_member_of_performer(
        in WfProcess member )
        raises (WfBase::BaseException);
```

2.3.3 Operations

receive_event

The following operation is used by *WfProcess* to notify its requester of workflow events. In particular the *WfProcess* must notify the requester of complete, terminate or abort events or the transition to a closed state.

The workflow event contains the source of the event; an *InvalidPerformer* exception is raised if the source of the event is not a performer associated with the *WfRequester*.

```
void receive_event(
        in WfEventAudit event)
        raises(WfBase::BaseException, InvalidPerformer);
```

2.4 WfExecutionObject

WfExecutionObject is an abstract base interface that defines common attributes, states, and operations for *WfProcess* and *WfActivity*.

It provides the capability to get and set and internal states. Operations are provided to get the current state and to make a transition from the current state into another state. Operations are also provided for specific state transitions. These operations are suspend, resume, terminate, and abort. States returned by these operations should not be confused with the “state of the process” which is calculated by the top level *WfProcess*. States returned by these operations pertain only to the object they are returned from. For example, regardless of what activity is currently enabled, a process as a whole can be paused and resumed. The propagation of state change of a *WfProcess* object down to *WfActivity* objects or subprocesses is implementation and process definition dependent.

The interface includes name, description, priority, and key attributes. It also provides an operation for monitoring *WfExecutionObject* executions by returning, based on filter specified, event audit records that represent the history of the execution. Other operations include methods for getting and setting context.

2.4.1 IDL

```
enum workflow_stateType{ open, closed };
enum while_openType{ not_running, running };
enum why_not_runningType{ not_started, suspended };
enum how_closedType{ completed, terminated, aborted };

typedef NameValueSequence ProcessData;

interface WfExecutionObject : WfBase::BaseBusinessObject {

        workflow_stateType workflow_state()
            raises (WfBase::BaseException);
```

```
        while_openType while_open()
            raises (WfBase::BaseException);
        why_not_runningType why_not_running()
            raises (WfBase::BaseException);
        how_closedType how_closed()
            raises (WfBase::BaseException);

        NameSequence valid_states()
            raises (WfBase::BaseException);
        string state()
            raises (WfBase::BaseException);
        void change_state(
            in string new_state)
            raises (WfBase::BaseException, InvalidState,
TransitionNotAllowed);

        string name()
            raises(WfBase::BaseException);
        void set_name( in string new_value)
            raises (WfBase::BaseException);
        string key()
            raises(WfBase::BaseException);
        void set_key(
            in string new_value)
            raises (WfBase::BaseException);
        string description()
            raises(WfBase::BaseException);
        void set_description(
            in string new_value)
            raises (WfBase::BaseException);
        ProcessData process_context()
            raises(WfBase::BaseException);
        void set_process_context(
            in ProcessData new_value)
            raises (WfBase::BaseException, InvalidData);
        unsigned short priority()
            raises(WfBase::BaseException);
        void set_priority(
            in unsigned short new_value)
            raises (WfBase::BaseException);

        void resume()
            raises (WfBase::BaseException, CannotResume);
        void suspend()
            raises (WfBase::BaseException,
            CannotSuspend, AlreadySuspended);
        void terminate()
            raises (WfBase::BaseException, CannotStop, NotRunning);
        void abort()
            raises (WfBase::BaseException, CannotStop, NotRunning);
```

```
        long how_many_history()
            raises (WfBase::BaseException, HistoryNotAvailable);
        WfEventAuditIterator get_iterator_history()
            raises (WfBase::BaseException, HistoryNotAvailable);
        WfEventAuditSequence get_sequence_history(
            in long max_number )
            raises (WfBase::BaseException, HistoryNotAvailable);
};
```

2.4.2 Attributes

Name	Type	Properties	Purpose
name	string		Descriptive name of a workflow execution object
key	string	readonly	(Business) identifier of an execution object that uniquely identifies it within the scope of its 'parent' object.
description	string		Information describing the execution object.
priority	unsigned short	constraint: 0 < priority < 6	A number representing the priority of the execution element.
process_context	ProcessData		The name-value pairs holding the process relevant data.

The following discusses the operations that support access to the attributes in detail.

name

Human readable, descriptive identifier of the execution object.

```
string name()
        raises(BseException);
void set_name( in string new_value)
        raises(WfBase::BaseException);
```

key

Identifier of the execution object. The key of a *WfProcess* is unique amongst the set of all *WfProcesses* created by a particular *WfProcessMgr*; the key of a *WfActivity* is unique within the set of all *WfActivities* contained in a particular *WfProcess*. A key is assigned to the execution object by its WfProcessMgr when it is created.

The key of a workflow object should not be confused with an 'object identifier'; it is used for reference to the process or activity independently of the lifetime of the execution object.

```
string key()
        raises(WfBase::BaseException);
```

description

Description of the execution object.

```
string description()
        raises(WfBase::BaseException);
void set_description( in string new_vaue)
        raises(WfBase::BaseException);
```

process_context

Process relevant data that define the context of the execution object. The process context is described by a set of named properties; the following operations support access to the context of an execution object. The NameValues structure identifies a set of property names and values matching the signature of the execution object; the signature of a *WfProcess* can be obtained using the *get_context_signature* operation provided by the *WfProcessMgr* of the process. An *InvalidContext* exception is raised when an update request does not match this signature; an *UpdateNotAllowed* exception is raised when the implementation of the WfM Facility or the specific workflow process does not allow an update of the context; see the section on *WfProcess* and *WfActivity* for details. When the set_process_context() method has been called only those name-value pairs in the parameter will be set. Several set_process_context() calls could be used to set the entire context.

```
ProcessData process_context()
        raises(WfBase::BaseException);
void set_process_context(
        in ProcessData new_value)
        raises (WfBase::BaseException, InvalidData, UpdateNotAllowed);
```

For a discussion of the context of WfActivity and WfProcess see the corresponding sections below.

priority

Relative priority of the execution element in the set of all execution objects of a given type. Valid values are numbers between one and five, with three being "normal" and one as the "highest" priority.

A request for update of the priority will raise an *InvalidPriority* exception when the specified priority is out of range; an *UpdateNotAllowed* exception is raised when the priority cannot be updated.

```
unsigned short priority()
        raises(WfBase::BaseException);

void set_priority(
        in unsigned short new_value)
        raises(WfBase::BaseException,InvalidPriority,UpdateNotAllowed);
```

2.4.3 States

We define a hierarchy of states of an execution object. The top level states are mandatory; implementations may define substates of the standard states defined here. The following section describes the standard states and the basic accessor operations.

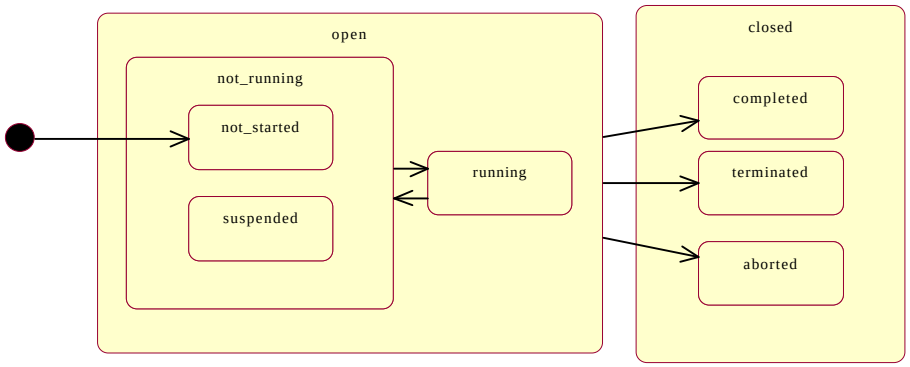

Figure 2-2 States of a WfExecutionObject

workflow_state state set

An execution object is either in state ’open’, i.e., it is active or in state ’closed’, i.e., it has finished execution.

Values	Substates	Purpose
open	while_open	To reflect that the object is active and not finished.
closed	how_closed	Reflects that the object is finished and inactive.

```
enum workflow_stateType { open, closed };

workflow_stateType workflow_state()
        raises(WfBase::BaseException);
```

while_open state set

Values	Substates	Purpose
not_running	why_not_running	Object is active and quiescent, but ready to execute.
running		The object is active and executing in the workflow.

```
enum while_openType { not_running, running };

while_openType while_ open()
        raises(WfBase::BaseException);
```

why_not_running state set

Values	Purpose
not_started	Provides a state after creation where the object is active and ready to be initialized and started.
suspended	Provides a state to temporarily pause the execution of the object. When an execution object is suspended, no execution objects depending on this object may be started.

```
enum why_not_runningType { not_started, suspended };

why_not_runningType why_not_running()
        raises(WfBase::BaseException);
```

how_closed state set

Values	Purpose
completed	When an execution object has finished its task in the overall workflow process it enters the completed state; it is assumed that all execution objects associated with that execution object are completed when it enters this state.
terminated	Indicates that enactment of the execution object was stopped before normal completion. It is assumed that all execution objects depending on this execution object (i.e., WfActivities contained in a WfProcess or a WfProcess implementing a WfActivity) are either completed or are terminated when it enters this state.
aborted	Indicates that the enactment of the execution object has been aborted before normal completion. No assumptions on the state of execution objects depending on this execution object are made when it enters this state.

```
enum how_closedType {completed, terminated, aborted };

how_closedType how_closed()
        raises(WfBase::BaseException);
```

Extended state access

The following operations support access to a potentially extended set of states; a state is represented by a 'dot-notation' representing hierarchical states, e.g., 'open.running'.

```
WfBase::NameSequence valid_states()
        raises(WfBase::BaseException);
```

Returns a list of all the valid states that can be reached from the current state; e.g., 'open.not_running.suspended' and 'closed.terminated' would be in the list of valid states if the current state was 'open.running'; 'open.not_running.not_started' probably would not be in that list.

```
string state();
```

Gets the current state of the object.

2.4.4 Relationships

Name	Type	Properties	Purpose
history	WfEventAudit	cardinality: 0..n readonly	Associates event audit data with its source execution object.

history

Zero or more *WfEventAudit* items can be associated with an execution object. An event audit item is generated (and associated with the source object) for each workflow relevant status change (change of state, context or result and change of resource assignment) of a *WfExecutionObject*; status changes can be explicitly triggered by operations that request a change of the object's status or implicitly by the workflow process logic. We will indicate which operations trigger generation of *WfEventAudit* items.

The following operations provide access to the set of all *WfEventAudit* items associated with a *WfExecutionObject*.

```
long how_many_history()
        raises (WfBase::BaseException, HistoryNotAvailable);
WfEventAuditIterator get_iterator_history(
        in string query,
        in NameValueSequence names_in_query)
        raises (WfBase::BaseException, HistoryNotAvailable);
WfEventAuditSequence get_sequence_history(
        in long max_number )
        raises (WfBase::BaseException, HistoryNotAvailable);
```

2.4.5 Operations

The following operations support execution control of the execution object; they all change the state (and potentially other features) of an execution object and its associated objects. Operations are provided to resume suspended *WfExecutionObjects*, suspend running executions, and terminate or abort open workflow execution objects.

All of these operations trigger creation of a state change event (*WfStateEventAudit*); other status changes resulting from the state change of the execution object might trigger creation of additional *WfEventAudit* items.

resume

Requests enactment of a suspended execution object to be resumed. The state is set to 'open.running' (or a substate) from 'open.not_running.suspended'.

A *CannotResume* exception is raised when the execution object cannot be resumed; for example, resuming a *WfActivity* might not be allowed when the containing *WfProcess* is suspended. A *NotSuspended* exception is raised when the object is not suspended.

```
void resume()
```

raises (WfBase::BaseException, CannotResume, NotSuspended);

terminate

Requests enactment of an execution object to be terminated before its normal completion. A terminate request is different from an abort request in its effect of execution object associated with the current execution object. See the sections on WfProcess and WfActivity for details.

The state is set to 'closed.terminated' (or one of its substates) from 'open.running' (or one of its substates).

A *CannotStop* exception is raised when the execution object cannot be terminated; for example, termination of a *WfActivity* might not be allowed when its implementation is still active and cannot be terminated. A *NotRunning* exception is raised when the object is not running.

void terminate()
raises (WfBase::BaseException, CannotStop, NotRunning);

suspend

Requests enactment of an execution object to be suspended. The state is set to 'open.not_running.suspended' (or one of its substates).

A *CannotSuspend* exception is raised when the execution object cannot be suspended; for example, an implementation of the WfM Facility might not support suspension a *WfActivity*. A *NotRunning* exception is raised when the object is not running.

void suspend()
raises (WfBase::BaseException, CannotSuspend, NotRunning);

abort

Requests enactment of a suspended execution object to be aborted before its normal completion. The state is set to 'closed.aborted'.

A *CannotStop* exception is raised when the execution object cannot be aborted. A *NotRunning* exception is raised when the object is not running.

void abort()
raises (WfBase::BaseException, CannotStop, NotRunning);

change_state

Updates the current state of the execution object. As a result the state of execution objects associated with this execution object might be updated, too. An *InvalidState* exception is raised when the *new_state* is not a valid state for the execution object; a *TransitionNotAllowed* exception is raised when the transition from the current state to *new_state* is not allowed.

void change_state(
in string new_state)

```
raises(WfBase::BaseException, InvalidState, TransitionNotAllowed);
```

2.5 WfProcessMgr

A *WfProcessMgr* represents a template for a specific workflow process; it is used to create instances of a workflow process. Logically it is the factory and locator for *WfProcess* instances. It provides access to the meta information about the context a process requires and the result a process produces.

A process manager is identified by its name which is unique within a given business domain; it could located, e.g., via name using the OMG Naming Service, via name and other attributes (e.g., category) via the OMG Trader Service, or other infrastructure mechanisms.

2.5.1 IDL

```
struct DataInfoType{
        string attribute_name;
        string type_name;
};
typedef sequence<DataInfoType> ProcessDataInfo;

enum process_mgr_stateType{enabled, disabled };

interface WfProcessMgr : WfBase::BaseBusinessObject {

        long how_many_process()
            raises (WfBase::BaseException);
        WfProcessIterator get_iterator_process()
            raises (WfBase::BaseException);
        WfProcessSequence get_sequence_process(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_process(
            in WfProcess member )
            raises (WfBase::BaseException);

        process_mgr_stateType process_mgr_state()
            raises(WfBase::BaseException);
        void set_process_mgr_state(
            in process_mgr_stateType new_state)
            raises(WfBase::BaseException, TransitionNotAllowed);

        string name()
            raises(WfBase::BaseException);
        void set_name(
            in string new_value)
            raises (WfBase::BaseException);
        string description()
            raises(WfBase::BaseException);
```

```
        void set_description(
            in string new_value)
            raises (WfBase::BaseException);
        string category()
            raises(WfBase::BaseException);
        void set_category(
            in string new_value)
            raises (WfBase::BaseException);
        string version()
            raises(WfBase::BaseException);
        void set_version(
            in string new_value)
            raises (WfBase::BaseException);

        ProcessDataInfo context_signature()
            raises (WfBase::BaseException);
        ProcessDataInfo result_signature()
            raises (WfBase::BaseException);

        WfProcess create_process(
            in WfRequester requester)
            raises (WfBase::BaseException, NotEnabled);
};
```

2.5.2 Attributes

Name	Type	Properties	Purpose
name	string	readonly	Name of the process manager.
description	WfActivity	readonly	Describes the workflow process type.
category	string	readonly	Provide an indication of the application domain the process was designed for.
version	string	readonly	Defines the version of this process manager.
context_signature	ProcessDataInfo	readonly	Describes the structure of the context data for the process
result_signature	ProcessDataInfo	readonly	Describes the structure of the result data for the process

All attributes of the WfProcessMgr are readonly; the are set when the process manager is installed. The following discusses the operations that support access to the attributes in detail.

name

Name of the process manager. The name uniquely identifies the process manager in a business domain.

string name();

description

Description of the process manager. It is set when the process manager is initialized and cannot be modified.

string description();

category

The category of a process manager is used for classification of process types. It is set when the process manager is initialized and cannot be modified.

string category();

version

The version attribute of a process manager is used to distinguish between different versions of a process model. Note that this is a means to distinguish between different process managers that have the same name; it is left to the implementation to define the format of the version attribute. It is set when the process manager is initialized and cannot be modified.

string version();

Process signature information

Meta information that defines how to set the context and return the result of an instance of this interface is returned by these operations.

The *ProcessDataInfo* structure identifies the name and the data type (IDL type represented by its string name) of the data item. *ProcessDataInfo* contains an entry for each data item in the set of context or result data for the *WfProcess*.

```
struct DataInfoType{
        string attribute_name;
        string type_name;
};

typedef sequence<DataInfoType> ProcessDataInfo;

DataInfoSequence get_context_signature();
```

Returns the meta information that defines how to set the context of an instance.

DataInfoSequence get_result_signature();
Returns the meta information that specifies how instances will return results.

2.5.3 Relationships

Name	Type	Properties	Purpose
process	WfProcess	cardinality: 0..n readonly	Locate process instances created using this WfProcessMgr.

process

Zero or more *WfProcess*es are associated with the *WfProcessMgr* that was used to create them. The association is established when a *WfProcess* is created.

The following operation support access to the set of *WfProcess*es associated with a *WfProcessMgr*.

```
long how_many_process()
        raises (WfBase::BaseException);
```

```
WfProcessIterator get_iterator_process()
        raises (WfBase::BaseException);
WfProcessSequence get_sequence_process(
        in long max_number )
        raises (WfBase::BaseException);
boolean is_member_of_process(
        in WfProcess member )
        raises (WfBase::BaseException);
```

2.5.4 States

process_ mgr_state state set

A *WfProcessMgr* can be enabled or disabled.

Values	Purpose
enabled	Indicates that creation of workflow processes is enabled.
disabled	Indicates that creation of workflow processes is disabled.

```
enum process_mgr_stateType{ enabled, disabled };
```

The following operation provide access to the state of a *WfProcessMgr*.

```
process_mgr_stateType process_mgr_state()
        raises(WfBase::BaseException);

void set_process_mgr_state(
        in process_mgr_stateType new_state)
        raises(WfBase::BaseException);
```

2.5.5 Operations

create_process

This operation is used to create instances of a process model and link its requester. When the process is created it enters state 'not_running.not_started'.

A *NotEnabled* exception is raised when the process manager is disabled.

```
WfProcess create_process(
        in WfRequester requester)
        raises (WfBase::BaseException, NotEnabled);
```

2.6 WfProcess

A *WfProcess* is the performer of a workflow request. All workflow objects that perform work implement this interface. This interface allows work to proceed asynchronously while being monitored and controlled.

The *WfProcess* interface specializes *WfExecutionObject* interface by adding an operation to start the execution of the process, an operation to obtain the result produced by the process and relationships with *WfRequester* and *WfActivity*.

Process States

When a WfProcess is created it enters 'open.not_running.not_started' state. When it has successfully finished processing, it enters 'closed.completed' state. Additional state changes can be performed using the *change_state* operation provided by *WfExecutionObject*.

Process context and results

In general, the context of a *WfProcess* is set when it has been created (using an appropriate WfProcessMgr factory) and before it is started. The context includes information about process navigation data, the resources to use and the results to produce. An implementation of the WfM Facility may or may not allow updates of the process context after the process has been started.

The result of a WfProcess is derived from the process context and from results of WfActivities contained in the WfProcess; a NULL result is possible and allowed. Derivation of result data is left to the implementation of the WfProcess.

Process Requester

A *WfProcess* is created (using a WfProcessMgr) by a user or automated resource and associated with a *WfRequester*. The *WfRequester* may be a *WfActivity* or an adapter for external clients. *WfProcess* always has one *WfRequester;* an implementation of the WfM Facility may allow for re-assignment of the WfRequester associated with a WfProcess.

A *WfProcess* will inform its *WfRequester* about status changes such as modifications of its state and its context using the requester's *receive_event* operation.

Process Steps

A *WfProcess* can contain zero or more *WfActivity* objects. The *WfActivity* objects represent steps in the process to be performed. The steps are assigned to *WfResources* or become *WfRequesters* that use and create *WfProcesses* as sub-processes. It is left to the implementation of the WfM Facility and the *WfProcess* to determine when to create and start *WfActivities*. The set of active *WfActivities* contained in a WfProcess can be obtained via the step relationship between WfProcess and *WfActivity.*

Process Monitoring and Control

The performing of the work represented by a *WfProcess* may take anywhere from seconds to months to even years for major projects. Operations are provided to monitor the status of the process and to control execution of the process.

Execution of a *WfProcess* is initiated using the start operation; execution can be suspended (and resumed) and terminated or aborted before it completes.

While the work is proceeding, the *state* operation on the *WfProcess* may be used to check on the overall status of the work. More detailed information on the status of the process can be obtained by navigating the relationship to the *WfActivities* contained in the *WfProcess* and using the status inquiries supported by this interface (see below).

The *get_result* operation may be used to request intermediate result data, which the may or may not be provided depending upon the details of the work being performed. The results are not final, until the unit of work is completed. When the status of a *WfProcess* changes, it sends a state change event to the requester informing it of the change. Notification is always delivered on “completed” or “terminated” or “aborted” events which tell the requesting object that the results could be available and the *WfProcess* object is done with its work.

WfProcess usage scenarios

In general, a *WfProcess* will represent an instance of a particular process model (e.g., ’approveCreditRequest’), the process steps being represented by WfActivities. Any other discrete unit of work, which needs to be performed asynchronously may implement this interface. It may or may not expose a fine grained structure in terms of process steps. For example, a wrapper for a legacy application could implement the *WfProcess* interface enabling that application to perform a task in another workflow process. A driver for an actual physical device, such as a numerical milling machine, could implement the *WfProcess* interface if that device were to be controlled by a workflow system.

2.6.1 IDL

```
interface WfProcess : WfExecutionObject {

        WfRequester requester()
            raises(WfBase::BaseException);
        void set_requester( in WfRequester new_value)
            raises (WfBase::BaseException, CannotChangeRequester);

        long how_many_step()
            raises (WfBase::BaseException);
        WfActivityIterator get_iterator_step()
            raises (WfBase::BaseException);
        WfActivitySequence get_sequence_step(
            in long max_number )
            raises (WfBase::BaseException);
```

```
        boolean is_member_of_step(
            in WfActivity member )
            raises (WfBase::BaseException);

        WfProcessMgr manager()
            raises(WfBase::BaseException);

        ProcessData result()
            raises (WfBase::BaseException, ResultNotAvailable);

        void start()
            raises (WfBase::BaseException, CannotStart, AlreadyRunning);
};

interface WfProcessIterator : WfBase::BaseIterator {
        WfProcess get_next_object ()
            raises (WfBase::BaseException);
        WfProcess get_previous_object()
            raises (WfBase::BaseException);
        WfProcessSequence get_next_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
        WfProcessSequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};
```

2.6.2 Attributes

Name	Type	Properties	Purpose
result	ProcessData	readonly	Result produced by the process.

The following discusses the operations that support access to the attributes in detail.

result

The result produced by the WfProcess. In general the result is undefined until the process completes, but some processes may produce intermediate results.

A *ResultNotAvailable* exception is raised when the result cannot be obtained yet.

```
ProcessData result()
        raises (WfBase::BaseException, ResultNotAvailable);
```

2.6.3 Relationships

Name	Type	Properties	Purpose
requester	WfRequester	cardinality: 1	Associate the requester of the process.
step	WfActivity	cardinality: 0..n readonly	Contain the activities of a process.
manager	WfProcessMgr	cardinality: 1 readonly	Identify the template for this instance.

requester

One WfRequester is associated with a WfProcess. The association is established when the process is created; implementations may support reassignment of the process to another requester. The following operations support the 'requester' relationship.

```
WfRequester requester()
        raises(WfBase::BaseException);

void set_requester(
        in WfRequester new_value)
        raises(WfBase::BaseException,
            CannotChangeRequester);
```

step

Zero or more *WfActivities* are associated with a *WfProcess*. The association is established when an activity is created as part of the enactment of the *WfProcess*. The following operations support the 'step' relationship.

```
long how_many_step()
        raises (WfBase::BaseException);
WfActivityIterator get_iterator_step()
        raises (WfBase::BaseException);
WfActivitySequence get_sequence_step(
        in long max_number )
        raises (WfBase::BaseException);
boolean is_member_of_step(
        in WfActivity member )
        raises (WfBase::BaseException);
```

manager

A process is associated with one *WfProcessMgr*; the association is established when the WfProcess is generated and cannot be modified. The following operation returns the *WfProcessMgr* associated with the *WfProcess*.

```
WfProcessMgr manager()
        raises(WfBase::BaseException);
```

2.6.4 Operations

start

This operation is used to initiate enactment of a *WfProcess*. The state of the process is changed from 'open.not_running.not_started' to 'open.running'.
A *CannotStart* exception is raised when the process cannot be started (e.g., because it is not properly initialized); an *AlreadyRunning* exception is raised when the process has already been started.

```
void start()
        raises (WfBase::BaseException, CannotStart, AlreadyRunning);
```

2.6.5 WfProcessIterator

The WfProcessIterator interface specializes the WfBase::BaseIterator interface and adds the event audit specific operations according to the Iterator pattern described in the section on patterns above.

The following attributes can be used in query expressions using the Trader Constraint Language: *key*, *name*, *priority*, *description*, *state*.

2.7 WfActivity

WfActivity is a step in a process that is associated, as part of an aggregation, with a single *WfProcess.* It represents a request for work in the context of the containing *WfProcess*. There can be many active *WfActivity* objects within a *WfProcess* at a given point in time.

The *WfActivity* interface specializes *WfExecutionObject* with an explicit complete operation to signal completion of the step, and with an operation to set the result of the *WfActivity*. It also adds relationships with *WfProcess* and *WfAssignment*.

Activity states

A *WfActivity* is created by the containing *WfProcess*; when it is created it enters state 'open.not_running.not_started'. It is left to the implementation of the WfM Facility or the *WfProcess* to decide when to create a *WfActivity.* The lifetime of an *WfActivity* is limited by that of its containing *WfProcess*.

When it becomes ready for execution, a *WfActivity* is transformed into state 'open.running'. It is left to the implementation of the WfM Facility or the *WfProcess* to decide when to activate a *WfActivity*.

A *WfActivity* enters state 'closed.completed' when its *complete* operation is invoked, or, if it is implemented by a *WfProcess*, when it receives a completion notification via the *receive_event* operation inherited from *WfRequester*.

Other operations are provided to modify the state of the *WfActivity* as described in the section on *WfExecutionObject*.

Activity context and result

The context of an activity is set by the containing *WfProcess* before the activity is activated; the context is derived from the context of the WfProcess and results of other activities. An implementation of the WfM Facility may support updates of the activity's context via the *set_context* operation inherited from *WfExecutionObject*.

An activity produces a result that can be used to determine which follow-on process steps to activate. It can also be used to determine the result of the WfProcess. In general, this overall result is not set until the process is closed; however, in-process, intermediate results may be available. In both cases the implementation of the workflow process sets the result in *WfProcess* and decides whether intermediate results will be available. The *set_result* operation is used to feed back activity results into the process.

Resource assignment

A *WfActivity* is a requester of work. Activities can be assigned to resources which participate in the execution of that work; a *WfAssignment* represents the association of a *WfResource* with a *WfActivity* and is used to indicate the nature of the assignment. Zero or more resources can be assigned to an activity.

It is up to the implementation of the WfM Facility, the *WfProcess*, or the owning *WfActivity* to coordinate the contributions of the resources assigned to an activity. This allows for the realization of a variety of collaboration patterns. For example, an implementation of the WfM Facility might decide to use *WfAssignment*s to offer work to a set of *WfResource*s but allow only one of them to actually perform the work; alternatively, the work might be split amongst the set of all resources that are assigned to a particular activity. Work items can be assigned to *WfResources* that accept or reject the work. Candidate resources include people or automated actors (see the section on WfResource for details).

Activity realizations

A *WfActivity* is a request for work to be done in the context of its parent workflow process. As a *WfRequester*, it can be associated with a *WfProcess*, as a subprocess, which performs the work. A *WfActivity* does not have to be performed by a subprocess, but can be performed by associated resources (e.g., people) using operations on the *WfActivity*, for instance to obtain the context of the activity, to indicate that the activity is completed, or to send the result data values.

If it is realized by a *WfProcess*, it is the responsibility of *WfActivity* to conform to the interface required by the *WfProcess* that is performing the work as a subprocess. This means that the context of the activity will be mapped to that of the subprocess using the context signature of the subprocess; also, results returned by the subprocess will be mapped to the results of the activity. The overall effect of this is that a *WfProcess* can be reused without any knowledge of its requesters. The *WfActivity* may use the meta data about the signature of the *WfProcess* provided by the *WfProcessMgr* of that process.

Process Monitoring

Given a reference to the *WfProcess*, the currently active *WfActivity* objects can be found. From each *WfActivity*, one can discover the sub *WfProcess* objects, if any, which may contain more activities. In this way, a distributed workflow of any scale can be navigated.

Status information on the process steps can be obtained using the operations to get the current state and the context of the corresponding *WfActivity*.

Activity - Process interaction

When a *WfActivity* is completed (it is told that the work is complete) the workflow process, through the use of internal logic, determines which activities are open and ready to start or resume. It is important to note that other events may also trigger a workflow system to dynamically determine its activities and their state.

2.7.1 IDL

```
interface WfActivity :
        WfExecutionObject, WfRequester {

        long how_many_assignment()
            raises (WfBase::BaseException);
        WfAssignmentIterator get_iterator_assignment()
            raises (WfBase::BaseException);
        WfAssignmentSequence get_sequence_assignment(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_assignment(
            in WfAssignment member )
            raises (WfBase::BaseException);

        WfProcess container()
            raises(WfBase::BaseException);

        ProcessData result()
            raises(WfBase::BaseException, ResultNotAvailable);
        void set_result(
                in ProcessData result)
```

```
                raises (WfBase::BaseException, InvalidData);

        void complete()
            raises (WfBase::BaseException, CannotComplete);
};

interface WfActivityIterator : WfBase::BaseIterator{

        WfActivity get_next_object ()
            raises (WfBase::BaseException);
        WfActivity get_previous_object()
            raises (WfBase::BaseException);
        WfActivitySequence get_next_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
        WfActivitySequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};
```

2.7.2 Attributes

Name	Type	Properties	Purpose
result	ProcessData		Result produced by the realization of the activity

result

Represents the result produced by the realization of the work request represented by an activity. An implementation of the WfM Facility may or may not provide access to the result of an activity; if it does not, or if the result data are not available yet, a ResultNotAvailable exception is raised by the *result* access operation.
The *set_result* operation is used to pass process data back to the workflow process. An *InvalidData* exception is raised when the data do not match the signature of the activity.

```
ProcessData result()
        raises(WfBase::BaseException, ResultNotAvailable)

void set_result(
        in ProcessData result )
        raises (WfBase::BaseException, InvalidData);
```

2.7.3 Relationships

Name	Type	Properties	Purpose
assignment	WfAssignment	cardinality: 0..n readonly	Links an activity to potential/actual resources.
container	WfProcess	cardinality: 1 readonly	Links the process this activity is part of.

assignment

Zero or more *WfAssignments* can be associated with a *WfActivity*; the association is established when the assignment is created as part of the resource selection process for the activity.

The following operations support access to the set of *WfAssignments* associated with an activity.

```
long how_many_assignment()
        raises (WfBase::BaseException);

WfAssignmentIterator get_iterator_assignment()
raises (WfBase::BaseException);

WfAssignmentSequence get_sequence_assignment(
        in long max_number )
        raises (WfBase::BaseException);

boolean is_member_of_assignment(
        in WfAssignment member )
        raises (WfBase::BaseException);
```

process

This operation returns the *WfProcess* that this activity is a part of.

```
WfProcess container()
        raises(WfBase::BaseException);
```

2.7.4 Operations

complete

This operation is used by an application to signal completion of the *WfActivity*. It will be used together with the *set_result* operation to pass results of the activity back to the workflow process.

A *CannotComplete* exception is raised when the activity cannot be completed yet.

```
void complete()
        raises (WfBase::BaseException, CannotComplete);
```

2.7.5 WfActivityIterator

The WfActivityIterator interface specializes the WfBase::BaseIterator interface and adds the event audit specific operations according to the Iterator pattern described in the section on patterns above.

The following attributes can be used in query expressions using the Trader Constraint Language: *key, name, priority, description, state*.

2.8 WfAssignment

WfAssignment links *WfActivity* requests for resources to potential or actual *WfResources.* This interface may be specialized by resource management facilities that interpret the context of the activity to create and negotiate assignments with resources.

Assignments are created as part of the resource selection process before an activity becomes ready for execution. The lifetime of an assignment is limited by that of the associated activity.

2.8.1 IDL

```
enum assignment_stateType { potential, accepted };

interface WfAssignment  : WfBase::BaseBusinessObject{

        WfActivity activity()
            raises(WfBase::BaseException);

        WfResource assignee()
            raises(WfBase::BaseException);
        void set_assignee(
            in WfResource new_value)
            raises (WfBase::BaseException);

        assignment_stateType assignment_state()
                raises(WfBase::BaseException);
        void set_assignment_state(
            in assignment_stateType new_value)
            raises (WfBase::BaseException, TransitionNotAllowed);
};

interface WfAssignmentIterator : WfBase::BaseIterator{

        WfAssignment get_next_object ()
```

```
            raises (WfBase::BaseException);
        WfAssignment get_previous_object()
            raises (WfBase::BaseException);
        WfAssignmentSequence get_next_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
        WfAssignmentSequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};
```

2.8.2 Relationships

Name	Type	Properties	Purpose
activity	WfActivity	cardinality: 1 readonly	Associate the activity this assignment exists for.
assignee	WfResource	cardinality: 1	Link the resource for this assignment.

activity

A *WfAssignment* is associated with one *WfActivity*; the association is established when the assignment is created as part of the resource selection process for the activity. The following operation returns the associated *WfActivity*.

```
WfActivity activity()
        raises(WfBase::BaseException);
```

assignee

A *WfAssignment* is associated with one *WfResource*. The association is established when the assignment is created as part of the resource selection process for the activity; the assignment can be reassigned to another resource at a later point in time. The following operations support the assignee relationship. An *InvalidResource* exception is raised by an attempt to assign an invalid resource to the assignment.

```
WfResource assignee()
        raises(WfBase::BaseException);

void set_assignee(
        in WfResource new_value)
        raises (WfBase::BaseException, InvalidResource);
```

2.8.3 States

assignment_state state set

Values	Purpose
potential	An assignment has been offered but not yet accepted.
accepted	Resource has accepted the assignment.

```
enum assignment_stateType { potential, accepted };
```

The following operations support access to and modification of the status of an assignment.

A *WfAssignmentEventAudit* is generated when the status of the assignment is changed; the *WfActivity* associated with the assignment is shown as the source of that event.

```
assignment_stateType assignment_state()
        raises(WfBase::BaseException);
void set_assignment_state(
        in assignment_stateType new_state)
        raises(WfBase::BaseException);
```

2.8.4 WfAssignmentIterator

The WfAssignmentIterator interface specializes the WfBase::BaseIterator interface and adds the event audit specific operations according to the Iterator pattern described in the section on patterns above.

The *state* attribute described for the WfAssignment interface can be used in query expressions using the Trader Constraint Language.

2.9 WfResource

WfResource is an abstraction that represents a person or thing that will potentially accept an assignment to an activity. Potential and/or accepted *WfAssignments* are links between the requesting *WfActivities* and *WfResource* objects. It is expected that this interface will be used to implement adapters for objects representing people and things implemented in user, organization, and resource models. These models are outside the scope of this specification.

2.9.1 IDL

```
interface WfResource : WfBase::BaseBusinessObject{

        long how_many_work_item()
            raises (WfBase::BaseException);
        WfAssignmentIterator get_iterator_work_item()
```

```
        raises (WfBase::BaseException);
    WfAssignmentSequence get_sequence_work_item(
        in long max_number )
        raises (WfBase::BaseException);
    boolean is_member_of_work_items(
        in WfAssignment member )
        raises (WfBase::BaseException);

    string resource_key()
        raises(WfBase::BaseException);
    string resource_name()
        raises(WfBase::BaseException);

    void release(
        in WfAssignment from_assigment,
        in string release_info)
        raises (WfBase::BaseException, NotAssigned);
};
```

2.9.2 Attributes

Name	Type	Properties	Purpose
resource_key	string	readonly	Uniquely identifies the resource.
resource_name	string	readonly	Name of an resource.

resource_key

The resource key identifies a resource within a given business domain; it is assumed that resources are defined in the same business domain as the workflow processes they are associated with.

The key is set when the object is initialized; modification of the key can be done in the context of a resource management facility.

```
string resource_key()
        raises(WfBase::BaseException);
```

resource_name

A human readable, descriptive name of the resource.

```
string resource_name()
        raises(WfBase::BaseException);
```

2.9.3 Relationships

Name	Type	Properties	Purpose
work_item	WfAssignment	cardinality: 0..n readonly	Provides a link to accepted work assignments.

work_item

Zero or more *WfAssignments* are associated with a resource; the association is established when the assignment is created as part of the resource selection process for an activity; the assignment can be reassigned to another resource at a later point in time.

The following operations provide access to the set of Wfassignments associated with a resource.

```
long how_many_work_item()
        raises (WfBase::BaseException);
WfAssignmentIterator get_iterator_work_item()
        raises (WfBase::BaseException);
WfAssignmentSequence get_sequence_work_item(
        in long max_number )
        raises (WfBase::BaseException);
boolean is_member_of_work_items(
        in WfAssignment member )
        raises (WfBase::BaseException);
```

2.9.4 Operations

release

The release operation is used to signal that the resource is no longer needed for a specific assignment. It takes the assignment that is no longer associated with the resource and a string that specifies additional information on the reason for realizing the resource as input. A *NotAssigned* exception is raised when the *WfAssignment* specified as input is not assigned to the *WfResource*.

It is assumed that this operation is invoked when an assignment is deleted or when an assignment it reassigned to another resource.

```
void release (
        in WfAssignment from_assignment,
        in string release_info)
        raises(WfBase::BaseException, NotAssigned);
```

2.10 WfEventAudit

WfEventAudit provides audit records of workflow event information. It provides information on the source of the event and contains specific event data. Workflow events include state changes, change of a resource assignment, and data changes. Workflow events are persistent and can be accessed navigating the history relationship of a *WfExecutionObject.* Workflow audit event objects are not part of the persistent state of their source workflow object.

A workflow event audit object is created when a workflow object changes its status (state change, process data change or assignment change); its lifetime is not limited by the lifetime of the event source object. Operations for managing the retention, archiving, and deletion of workflow events are not specified in this submission.

The *WfEventAudit* defines a set of event properties common to all workflow audit events; in particular it provides an identification of the source of the event in terms of (business) identifiers of the workflow entities *WfProcessMgr, WfProcess* and *WfActivity*.

2.10.1 IDL

```
interface WfEventAudit : BaseBusinessObject{

        WfExecutionObject source()
            raises(WfBase::BaseException, SourceNotAvailable);

        TimeBase :: UtcT timestamp()
            raises(WfBase::BaseException);
        string event_type()
            raises(WfBase::BaseException);
        string activity_key()
            raises(WfBase::BaseException);
        string activity_name()
            raises(WfBase::BaseException);
        string process_key()
            raises(WfBase::BaseException);
        string process_name()
            raises(WfBase::BaseException);
        string process_mgr_name()
            raises(WfBase::BaseException);
        string process_mgr_version()
            raises(WfBase::BaseException);
};

interface WfEventAuditIterator : WfBase::BaseIterator{
        WfEventAudit get_next_object ()
            raises (WfBase::BaseException);
        WfEventAudit get_previous_object()
            raises (WfBase::BaseException);
        WfEventAuditSequence get_next_n_sequence(
```

```
            in long max_number)
            raises (WfBase::BaseException);
        WfEventAuditSequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};
```

2.10.2 Attributes

Name	Type	Properties	Purpose
timestamp	TimeBase:UtcT	readonly	Records time of the event.
event_type	string	readonly	Describes the audit event type.
activity_key	string	readonly	Identifies the WfActivity associated with the event; NULL for process events.
activity_name	string	readonly	Name of the WfActivity associated with the event; NULL for process events.
process_key	string	readonly	Identifies the WfProcess associated with the event.
process_name	string	readonly	Name of the process associated with the event
process_mgr_name	string	readonly	Name of the process manager associated with the event
process_mgr_version	string	readonly	Version of the process manager

timestamp

Records the time the status change of the source occurred that triggered the event audit item to be created, using the TimeBase::UtcT data type defined by the OMG Time Service.

TimeBase::UtcT timestamp();

event_type

Identifies the specific event type. The following is a set of pre-defined event types; implementations of the WfM Facility may decide to support additional audit event types.

Name	Purpose
processCreated	A WfProcess was created
processStateChanged	The state of a WfProcess was changed
processContextChanged	The context of a WfProcess was initialized or changed
activityStateChanged	The state of a WfActivity was changed
activityContextChanged	The context of a WfActivity was changed
activityResultChanged	The result of a WfActivity was set
activityAssigmentChanged	The status or the resource assignment of a WfAssignment was initialized or changed

string event_type() raises(WfBase::BaseException);

activity_key and activity_name

If the event is triggered by a status change of a WfActivity, the key and the name of the activity is recorded with the WfEventAudit. Otherwise the activity related attributes contain a NULL value.

The following operations return the key and the name of the WfActivity associated with the event.

string activity_key() raises(WfBase::BaseException);
string activity_name() raises(WfBase::BaseException);

process_key and process_name

The key and the name of the *WfProcess* associated with the source of an event are recorded with the *WfEventAudit*. If the event was triggered by a WfActivity this is the containing WfProcess; if it was triggered by a status change of a WfProcess, it is that process.

The following operations return the key and the name of the *WfProcess* associated with the event.

string process_key() raises(WfBase::BaseException);
string process_name() raises(WfBase::BaseException);

process_mgr_name and process_mgr_version

The *WfProcessMgr* associated with the workflow object that triggered the event is identified via its name and version. If the event was triggered by a status change of an activity this is the manager of the process that contains the activity; if it was triggered by a status change of a process, this is the manager of that process.

```
string process_mgr_name() raises(WfBase::BaseException);
string process_mgr_version() raises(WfBase::BaseException);
```

2.10.3 Relationships

Name	Type	Properties	Purpose
source	WfExecutionObject	cardinality: 0..1 readonly	Associates the source of the event.

source

A WfEventAudit can be associated with the WfExecutionObject which triggered the event. Event audit items are meant to provide information on the execution history of workflow object even after the source object has been deleted; in this case, no source would be associated with the WfEventAudit.

The following operation returns the source of the event, when available; if the source is not available, a *SourceNotAvailable* exception is raised.

```
WfExecutionObject source()
        raises(WfBase::BaseException, SourceNotAvailable);
```

2.10.4 WfEventAuditIterator

The WfEventAuditIterator interface specializes the WfBase::BaseIterator interface and adds the event audit specific operations according to the Iterator pattern described in the section on patterns above.

All of the attributes described for the WfEventAudit interface can be used in query expressions using the Trader Constraint Language.

2.10.5 Publication via Notification Service

A workflow event can be published using the OMG Notification Service (note that BaseBusinessObject is a CosNotifyComm:StructuredPushSupplier). The information recorded by a WfEventAudit entity is mapped into the CosNotification :: StructuredEvent data structure as follows:

- FixedEventHeader: domain_name is set to 'workflow', event_type is set to the event_type defined here, event_name is set to NULL
- OptionalHeaderFields are used to hold the other attributes defined above; the attributes are mapped to the PropertySequence (i.e., name-value pair sequence) of the optional header in the obvious way using the attribute names to identify the properties and string-type values.
- Specialization of the WfEventAudit entity will use the body fields of the StructuredEvent; the mapping for the four specialization defined here is given below.

2.11 WfCreateProcessEventAudit

This interface specializes WfEventAudit by adding information related to creation of a WfProcess. If the process is created as a sub-process of another process that is synchronized with the main process via a WfActivity requester, information on the requester is recorded.

The event_type is set to *processCreated* for this event.

2.11.1 IDL

```
interface WfCreateProcessEventAudit : WfEventAudit{

        string p_activity_key()
            raises(WfBase::BaseException);
        string p_process_key()
            raises(WfBase::BaseException);
        string p_process_name()
            raises(WfBase::BaseException);
        string p_process_mgr_name()
            raises(WfBase::BaseException);
        string p_process_mgr_version()
            raises(WfBase::BaseException);
};
```

2.11.2 Attributes

Name	Type	Properties	Purpose
p_activity_key	string	readonly	Identify activity which is the requester for the newly created process
p_process_key	string	readonly	Identify process that contains parent activity.
p_process_mgr_name	string	readonly	Identify process manager of the parent process.
p_process_mgr_version	string	readonly	Identifies the version of the process manager of the parent process.

p_activity_key

If the requester of the newly created workflow process is a WfActivity, the key of that activity is recorded.

string p_activity_key() raises(WfBase::BaseException);

p_process_key

If the requester of the newly created workflow process is a *WfActivity*, the key of the WfProcess that contains that activity is recorded.

string p_process_key() raises(WfBase::BaseException);

p_process_mgr_name and p_process_mgr_version

If the requester of the newly created workflow process is a *WfActivity*, name and version of the process manager of the process that contains that activity is recorded.

string p_process_mgr_name() raises(WfBase::BaseException);
string p_process_mgr_version() raises(WfBase::BaseException);

2.11.3 Publication via Notification Service

The attributes defined by this specialization of the WfEventAudit are mapped into the FilterableEventBody of the StructuredEvent; mapping is straightforward, using the attribute names to identify the properties and string-type values.

2.12 WfStateEventAudit

This interface specializes *WfEventAudit* by adding state change information. A state change event is signaled when a *WfExecutionObject* changes its state; this covers both state changes resulting from a *change_state* operation request and internal state changes triggered by the execution logic of a *WfProcess* (e.g., process completes successfully, activity is suspended because the containing process was suspended, etc.). The *event_type* is *processStateChanged* or *activityStateChanged*.

2.12.1 IDL

```
interface WfStateEventAudit : WfEventAudit {

        string old_state()
            raises(WfBase::BaseException);
        string new_state()
            raises(WfBase::BaseException);
};
```

2.12.2 Attributes

Name	Type	Properties	Purpose
old_state	string	readonly	Records the previous state.
new_state	string	readonly	Records the new state.

old_state

The state of the execution object before the status change is recorded. The state is described using 'dot-notation'. The 'old' state is recorded for convenience here; it could be deduced by analyzing the history of the execution object. Recording of the old state is optional.

```
string old_state() raises(WfBase::BaseException);
```

new_state

The state of the execution object after the state change is recorded. The state is described using 'dot-notation'.

```
string new_state() raises(WfBase::BaseException);
```

2.12.3 Publication via Notification Service

The attributes defined by this specialization of the WfEventAudit are mapped into the FilterableEventBody of the StructuredEvent; mapping is straightforward, using the attribute names to identify the properties and string-type values.

2.13 WfDataEventAudit

This interface specializes *WfEventAudit* for data change events. A data change event is signaled when the context of a *WfExecutionObject* or the result of a *WfActivity* is initialized or changed. The *event_type* is *processContextChanged, activityContextChanged* or *activityResultChanged*.

2.13.1 IDL

```
interface WfDataEventAudit  : WfEventAudit {

        ProcessData old_data()
            raises(WfBase::BaseException);
        ProcessData new_data()
            raises(WfBase::BaseException);
};
```

2.13.2 Attributes

Name	Type	Properties	Purpose
old_data	ProcessData	readonly	Identifies the previous data used.
new_data	ProcessData	readonly	Records the new data to be used.

These operations return additional information about the data change event.

old_data

Records the context resp. result data of the execution object before the change; only the data items that were changed are reported. This event also records the initialization of the context of a WfProcess resp. of the result of a WfActivity; in these cases, old_data is NULL.

The 'old' data are recorded for convenience here; they could be deduced by analyzing the history of the execution object. Support for recording of old data is optional.

ProcessData old_data() raises(WfBase::BaseException);

new_data

Records the context resp. result data of the execution object after the change; only the data items that were changed are reported. This event also records the initialization of the context of a WfProcess resp. of the result of a WfActivity; in these cases, new_data contains the initial data.

ProcessData new_data() raises(WfBase::BaseException);

2.13.3 Publication via Notification Service

The information recorded in the *new_data* attribute by this specialization of the WfEventAudit are mapped into the 'remainder_of_body' part of the StructuredEvent.

2.14 WfAssignmentEventAudit

This interface specializes *WfEventAudit* for assignment change events. The event records resource and assignment status before and after the change. The *event_type* is *activityAssignmentChanged*.

An assignment change event is signaled when assignments for an activity are created (in this case the old_... data is NULL), when the status of an assignment is changed, or when an existing assignment is reassigned to another resource. The *WfActivity* associated with the assignment is reported as the source of the event.

2.14.1 IDL

```
interface WfAssignmentEventAudit : WfEventAudit{

        string old_assignment_state()
            raises(WfBase::BaseException);
        string old_resource_key()
            raises(WfBase::BaseException);
        string old_resource_name()
            raises(WfBase::BaseException);

        string new_assignment_state()
            raises(WfBase::BaseException);
        string new_resource_key()
            raises(WfBase::BaseException);
        string new_resource_name()
            raises(WfBase::BaseException);
};
```

2.14.2 Attributes

Name	Type	Properties	Purpose
old_assignment_state	string	readonly	Records assignment status before the change
old_resource_key	string	readonly	Identifies resource associated with assignment before the change.
old_resource_name	string	readonly	Name of the associated resource.
new_assignment_state	string	readonly	Records assignment status after the change
new_resource_key	string	readonly	Identifies resource associated with assignment after the change.
new_resource_name	string	readonly	Name of the associated resource.

old_assignment_state, old_resource_key and old_resource_name

The status of the assignment before the change may be recorded. This event also covers creation of a new assignment; in this case, the 'before event' information is NULL.

```
string old_assignment_state() raises(WfBase::BaseException);
string old_resource_key() raises(WfBase::BaseException);
string old_resource_name() raises(WfBase::BaseException);
```

new_assignment_state, new_resource_key and new_resource_name

The status of the assignment after the change is recorded.

```
string new_assignment_state() raises(WfBase::BaseException);
string new_resource_key() raises(WfBase::BaseException);
string new_resource_name() raises(WfBase::BaseException);
```

2.14.3 Publication via Notification Service

The attributes defined by this specialization of the WfEventAudit are mapped into the FilterableEventBody of the StructuredEvent; mapping is straightforward, using the attribute names to identify the properties and string-type values.

2.15 The WfBase Module

The WfBase module defines a set of base interfaces for the workflow interfaces. This 'base framework' is separated from the core specification to enable adaptation of this specification to the results of the ongoing work in OMG on the definition of a 'Business Component Framework'.

```
#ifndef _WF_BASE_
#define _WF_BASE_
#pragma prefix "omg.org"
module WfBase {

// Data Types

        struct NameValueInfo{
            string attribute_name;
            string type_name;
        };
        typedef sequence<NameValueInfo> NameValueInfoSequence;

        struct NameValue{
            string the_name;
            any the_value;
        };
        typedef sequence <NameValue> NameValueSequence;

        typedef sequence <string> NameSequence;

        struct BaseError {
            long exception_code;
            string exception_source;
            any exception_object;
            string exception_reason;
            any exception_data;
        };
        typedef sequence <BaseError> BaseErrorSequence;

// Exceptions

        exception BaseException {
            BaseErrorSequence errors;
        };

        exception InvalidNames{};
        exception InvalidQuery{};
        exception GrammarNotSupported{};

// Interfaces
```

```
        interface BaseBusinessObject :
            CosTransactions :: TransactionalObject,
            CosObjectIdentity :: IdentifiableObject,
            CosNotifyComm :: StructuredPushSupplier,
            CosLifeCycle :: LifeCycleObject {};

};
#endif
```

2.15.1 Data Types

NameValueInfo

```
struct NameValueInfo{
        string attribute_name;
        string type_name;
};
typedef sequence<NameValueInfo> NameValueInfoSequence;
```

The *NameValueInfo* structure provides information on the structure of a name-value pair. The attribute_name attribute provides the name of the pair, the type_name attribute identifies the (IDL) type of the value.

NameValue

```
struct NameValue{
    string the_name;
    any the_value;
};
typedef sequence <NameValue> NameValueSequence;
```

The NameValue structure is used to handle name-value pair lists; the the_name attribute holds the string name of the item, the the_value attribute is a CORBA::Any and holds the value of the item.

NameSequence

```
typedef sequence <string> NameSequence;
```

Used to handle lists of names.

Base Error

```
        struct BaseError {
            long exception_code;
            string exception_source;
            any exception_object;
            string exception_reason;
```

```
    string exception_data;
};

typedef sequence <BaseError> BaseErrorSequence;
```

The BaseError structure is used to hold information on an application error. The *exception_source* is a printable description of the source of the exception. The *exception_object* is a pass-by-value object or an object reference of the object which generated the exception. The *exception_code* is an identifier associated with the source type. The *exception_reason* is a textual string containing a description of the exception and should correspond to the code.

2.15.2 Exceptions

BaseException

```
exception BaseException {
    BaseErrorSequence errors;
};
```

BaseException is an exception that holds a sequence of BaseError structures - essentially a sequence of exceptions. The sequence is a push-down list so that the most recently occurring exception is first. This allows multiple exceptions to be returned so that multiple problems may be addressed, as where a user has a number of data entry errors or where consequential errors are recorded as a result of a low-level exception.

The *BaseException* is returned by all operations defined in this specification to support implementations of the WfM Facility to raise implementation specific exceptions.

QueryExceptions

```
exception InvalidNames{};
exception InvalidQuery{};
exception GrammarNotSupported{};
```

The *InvalidNames* exception is raised when the NameValue list provided as input for a set_names_in_expression operation on a BaseIterator.

The *InvalidQuery* exception is raised when an invalid query expression is provided as input for a set_query_expression operations on a BaseIterator.

The *GrammarNotSuported* exception is raised when the input parameter of the set_query_grammar on a BaseIterator specifies a query grammar that is not supported by the iterator.

2.16 Base Business Object Interfaces

2.16.1 BaseBusinessObject

A BaseBusinessObject is the base interface for all business object interfaces. It is a TransactionalObject, supports LifeCycle operations, is Identifiable and may publish notification events via the OMG Notification Service as a StructuredPushSupplier.

For all conformance classes except 'FrameworkSupport' this is a CORBA Object.

```
interface BaseBusinessObject :
        CosTransactions :: TransactionalObject,
        CosObjectIdentity :: IdentifiableObject,
        CosNotifyComm :: StructuredPushSupplier,
        CosLifeCycle :: LifeCycleObject {};
```

2.17 BaseIterator

The *BaseIterator* interface is used to navigate relationships of cardinality greater than 1 in this specification. It supports specification of a filter using parametrized query expressions.

2.17.1 IDL

```
interface BaseIterator {

        string query_expression()
            raises(BaseException);
        void set_query_expression(
            in string query)
            raises(BaseException, InvalidQuery);

        NameValueSequence names_in_expression()
            raises(BaseException);
        void set_names_in_expression(
            in NameValueSequence query)
            raises(BaseException, InvalidNames);

        string query_grammar()
            raises(BaseException);
        void set_query_grammar(
            in string query_grammmar)
            raises(BaseException, GrammarNotSupported);

        long how_many ()
            raises(BaseException);
        void goto_start()
            raises(BaseException);
```

```
            void goto_end()
                raises(BaseException);
};
```

2.17.2 Attributes

query_expression

Defines the query expression used to filter the contents of the iterator.

names_in_expression

Defines a set of parameters that used to substitute variables in the query_expression. The parameters are defined by name-value pairs, where the name identifies the variable and the value represents the variable value to be substituted.

query_grammar

The query_grammar attribute identifies the query grammar used to define the query expression. The Constraint Language defined by the OMG Object Trading Service is used as the mandatory query grammar in this specification; implementations of the WfM Facility may support additional query grammars. The Trader Constraint Language is identified via the string *TCL*.

For each workflow object, the set of attributes that can be used as property identifiers in queries on sets the specific object type is identified in the corresponding sections above.

2.17.3 Operations

how_many

Returns the number of elements in the collection.

goto_start

Positions the iterator such that the next “next” retrieval will retrieve the first element in the collection.

goto_end

Positions the iterator such that the next “previous” retrieval will retrieve the last element in the collection.

2.18 Interface Usage Example

The following object interaction diagram shows one possible set of interactions that illustrate the enactment of a process from creation through completion.

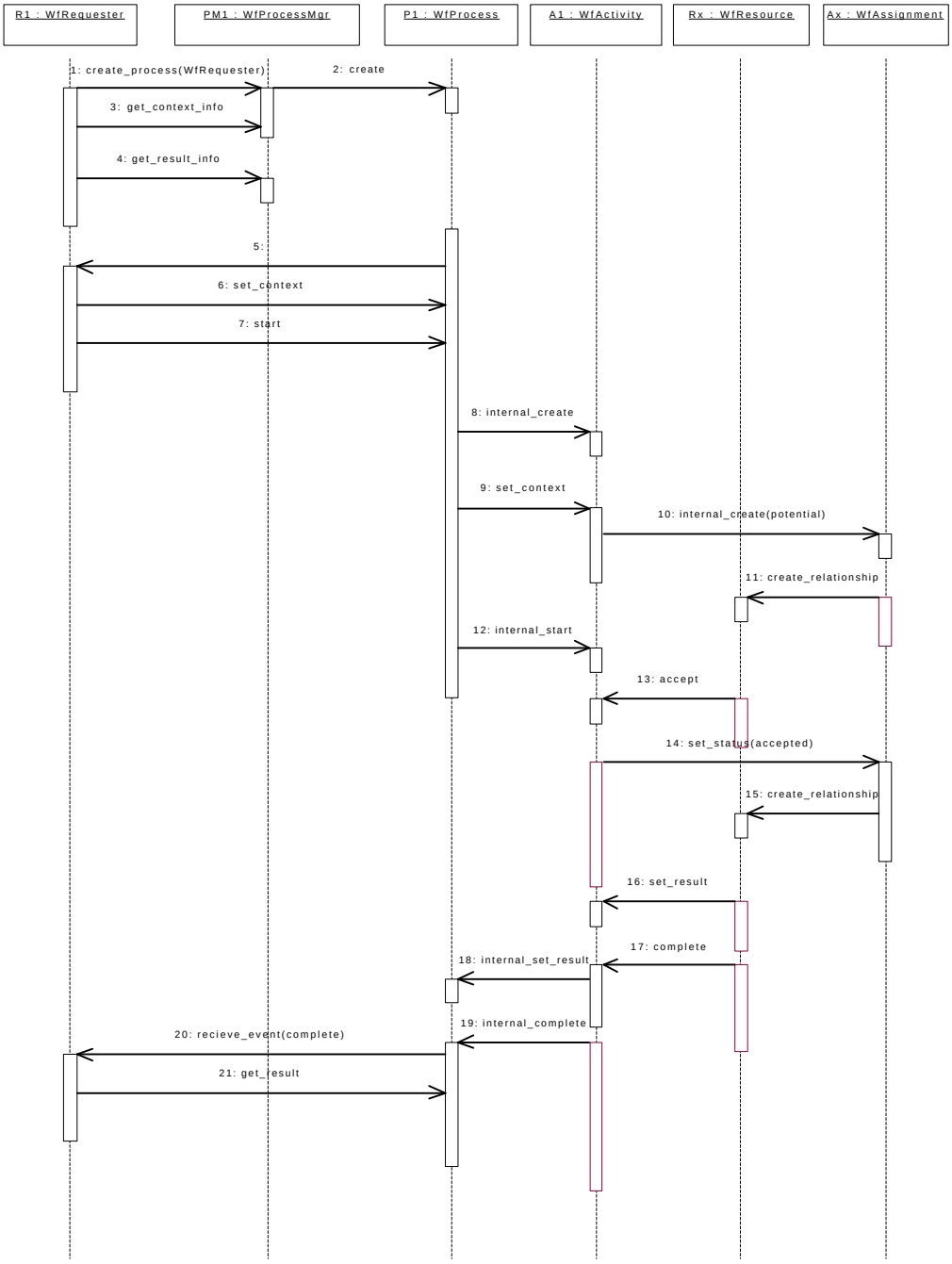

Figure 2-3 Process Creation to Completion Object Interaction Diagram

In this example, a WfProcess is created by an application implementing the WfRequester interface; an appropriate WfProcessMgr is identified and the WfProcess is created using the create_process operation on this process manager. The process manager creates a new WfProcess and returns a reference to the requester.

The requester retrieves information about the signature of the process using the get_context_info and get_result_info operations on the process manager and uses the information on the structure of the process context to initialize the process context using the set_context operation on the WfProcess.

Next, the requester initiates enactment of the process using the start operation of the WfProcess; as a result, the process determines the activities to be activated (in our example it is only one, there might be more), creates a WfActivity that represents the first step in the process and sets the context of that activity using the data that were provided during initialization of the process and potentially additional information.

In this scenario, the WfActivity then establishes an association with a WfResource that can potentially perform the work request represented by the activity. The association is establish be creating a WfAssignment which establishes an association with an appropriate resource using internal knowledge about resource selection. Note that it could use, for example, a resource selection WfProcess to perform this task; the resource selection mechanism is not subject of this specification. Note also that instead of assigning the activity to a resource, the activity could also be realized by another workflow process which essentially performs the same operations that are performed by the resource in our scenario.

Next, the process starts the activity and the potential assignment is changed into an ’actual’ one because the resource decided to accept the assignment (and changed the state of the assignment accordingly).

Then, the resource (or some application) performs the work request represented by the activity, returns the result to the activity and invokes the complete operation on the activity to signal that the task has been completed.

The activity informs the process about the status change and passes the result on for further processing by the process. The process could use the information to determine the next activities to be activated; in our example, however, the process decides that the work is done and signals completion to its requester using the result of the activity to determine the overall process result.

The requester receives the process completion notification and retrieves the process result using the get_result operation on the process.

Additional Information 3

3.1 Summary of Optional versus Mandatory Interfaces

All interfaces, at each compliance level, are mandatory.

3.2 Proposed Compliance Points

All implementations of this specification require that interfaces be implemented in a CORBA environment and can be invoked through the Internet Inter-Operability Protocol (IIOP). Operations are to be invoked in a transactional context and the effects of those operations will be made persistent or rolled back through commit or rollback of associated transactional resources.

The levels of compliance described below recognize that legacy or otherwise incompatible systems may provide lesser levels of compliance which still provide value.

Base Level

Provides interfaces for requesting and obtaining status of a process.

- Provides the WfProcessMgr and WfProcess interfaces
- Responds to the WfRequester interface.

Master process

Invokes other processes through the requester-process protocol.

- Provides activities with the WfRequester interface.
- Can invoke external processes through the WfProcessMgr and WfProcess interfaces.

Full compliance

All interfaces defined in this specification are supported.

3.3 Changes or Extensions to Adopted OMG Specifications

There are no specific changes or extensions to adopted OMG specifications.

3.4 Complete IDL Definitions

The following lists the complete IDL for the proposed Workflow Management Facility.

3.4.1 Consolidated IDL

```
#ifndef _WF_BASE_
#define _WF_BASE_
#include <orb.idl>
#include <CosNotification.idl>
#include <CosTransactions.idl>
#include <CosLifeCycle.idl>
#include <ObjectIdentity.idl>
#pragma prefix "omg.org"
module WfBase {

// DataTypes

        struct NameValueInfo{
            string attribute_name;
            string type_name;
        };
        typedef sequence<NameValueInfo> NameValueInfoSequence;
        struct NameValue{
            string the_name;
            any the_value;
        };
        typedef sequence <NameValue> NameValueSequence;
        typedef sequence <string> NameSequence;
        struct BaseError {
            long exception_code;
            string exception_source;
            any exception_object;
            string exception_reason;
        };
        typedef sequence <BaseError> BaseErrorSequence;

// Exceptions

        exception BaseException {
            BaseErrorSequence errors;
        };
        exception InvalidNames{};
        exception InvalidQuery{};
        exception GrammarNotSupported{};

// Interfaces

        interface BaseBusinessObject :
```

```
            CosTransactions :: TransactionalObject,
            CosObjectIdentity :: IdentifiableObject,
            CosNotifyComm :: StructuredPushSupplier,
            CosLifeCycle :: LifeCycleObject {};

        interface BaseIterator {

            string query_expression()
                raises(BaseException);
            void set_query_expression(
                in string query)
                raises(BaseException, InvalidQuery);

            NameValueSequence names_in_expression()
                raises(BaseException);
            void set_names_in_expression(
                in NameValueSequence query)
                raises(BaseException, InvalidNames);

            string query_grammar()
                raises(BaseException);
            void set_query_grammar(
                in string query_grammmar)
                raises(BaseException, GrammarNotSupported);

            long how_many ()
                raises(BaseException);
            void goto_start()
                raises(BaseException);
            void goto_end()
                raises(BaseException);
        };
};
#endif

#ifndef _WORKFLOW_MODEL_
#define _WORKFLOW_MODEL_
#include <WfBase.idl>
#include <TimeBase.idl>
#pragma prefix "omg.org"

module WorkflowModel{

// Forward declarations

        interface WfExecutionObject;
        interface WfProcess;
        interface WfProcessIterator;
        interface WfRequester;
        interface WfProcessMgr;
        interface WfActivity;
```

```
        interface WfActivityIterator;
        interface WfResource;
        interface WfAssignment;
        interface WfAssignmentIterator;
        interface WfEventAudit;
        interface WfEventAuditIterator;
        interface WfCreateProcessEventAudit;
        interface WfStateEventAudit;
        interface WfAssignmentEventAudit;

    // DataTypes

        typedef sequence<WfProcess> WfProcessSequence;
        typedef sequence<WfActivity> WfActivitySequence;
        typedef sequence<WfAssignment> WfAssignmentSequence;
        typedef sequence<WfEventAudit> WfEventAuditSequence;
        typedef WfBase::NameValueInfoSequence ProcessDataInfo;
        typedef WfBase::NameValueSequence ProcessData;

        enum workflow_stateType{ open, closed };
        enum while_openType{not_running, running };
        enum why_not_runningType{ not_started, suspended };
        enum how_closedType{ completed, terminated, aborted };
        enum process_mgr_stateType{enabled, disabled };
        enum assignment_stateType { potential, accepted };

    // Exceptions

        exception InvalidPerformer{};
        exception InvalidState{};
        exception InvalidData{};
        exception TransitionNotAllowed{};
        exception CannotResume{};
        exception CannotSuspend{};
        exception AlreadySuspended{};
        exception CannotStop{};
        exception NotRunning{};
        exception HistoryNotAvailable{};
        exception NotEnabled{};
        exception AlreadyRunning{};
        exception CannotStart{};
        exception ResultNotAvailable{};
        exception CannotComplete{};
        exception NotAssigned{};
        exception SourceNotAvailable{};

    // Interfaces

        interface WfRequester : WfBase::BaseBusinessObject{
            long how_many_performer()
                raises (WfBase::BaseException);
```

```
        WfProcessIterator get_iterator_performer()
            raises (WfBase::BaseException);
        WfProcessSequence get_sequence_performer(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_performer(
            in WfProcess member )
            raises (WfBase::BaseException);
        void receive_event(
            in WfEventAudit event)
            raises (WfBase::BaseException, InvalidPerformer);
    };

    interface WfExecutionObject : WfBase::BaseBusinessObject {
        workflow_stateType workflow_state()
            raises (WfBase::BaseException);
        while_openType while_open()
            raises (WfBase::BaseException);
        why_not_runningType why_not_running()
            raises (WfBase::BaseException);
        how_closedType how_closed()
            raises (WfBase::BaseException);
        WfBase::NameSequence valid_states()
            raises (WfBase::BaseException);
        string state()
            raises (WfBase::BaseException);
        void change_state(
            in string new_state)
            raises (WfBase::BaseException, InvalidState,
            TransitionNotAllowed);
        string name()
            raises(WfBase::BaseException);
        void set_name( in string new_value)
            raises (WfBase::BaseException);
        string key()
            raises(WfBase::BaseException);
        void set_key(
            in string new_value)
            raises (WfBase::BaseException);
        string description()
            raises(WfBase::BaseException);
        void set_description(
            in string new_value)
            raises (WfBase::BaseException);
        ProcessData process_context()
            raises(WfBase::BaseException);
        void set_process_context(
            in ProcessData new_value)
            raises (WfBase::BaseException, InvalidData);
        unsigned short priority()
            raises(WfBase::BaseException);
```

```
        void set_priority(
            in unsigned short new_value)
            raises (WfBase::BaseException);
        void resume()
            raises (WfBase::BaseException, CannotResume);
        void suspend()
            raises (WfBase::BaseException, CannotSuspend,
            AlreadySuspended);
        void terminate()
            raises (WfBase::BaseException, CannotStop, NotRunning);
        void abort()
            raises (WfBase::BaseException, CannotStop, NotRunning);
        long how_many_history()
            raises (WfBase::BaseException, HistoryNotAvailable);
        WfEventAuditIterator get_iterator_history()
            raises (WfBase::BaseException, HistoryNotAvailable);
        WfEventAuditSequence get_sequence_history(
            in long max_number )
            raises (WfBase::BaseException, HistoryNotAvailable);
    };

    interface WfProcessMgr : WfBase::BaseBusinessObject {
        long how_many_process()
            raises (WfBase::BaseException);
        WfProcessIterator get_iterator_process()
            raises (WfBase::BaseException);
        WfProcessSequence get_sequence_process(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_process(
            in WfProcess member )
            raises (WfBase::BaseException);
        process_mgr_stateType process_mgr_state()
            raises(WfBase::BaseException);
        void set_process_mgr_state(
            in process_mgr_stateType new_state)
            raises(WfBase::BaseException, TransitionNotAllowed);
        string name()
            raises(WfBase::BaseException);
        void set_name(
            in string new_value)
            raises (WfBase::BaseException);
        string description()
            raises(WfBase::BaseException);
        void set_description(
            in string new_value)
            raises (WfBase::BaseException);
        string category()
            raises(WfBase::BaseException);
        void set_category(
            in string new_value)
```

```
            raises (WfBase::BaseException);
        string version()
            raises(WfBase::BaseException);
        void set_version(
            in string new_value)
            raises (WfBase::BaseException);
        ProcessDataInfo context_signature()
            raises (WfBase::BaseException);
        ProcessDataInfo result_signature()
            raises (WfBase::BaseException);
        WfProcess create_process(
            in WfRequester requester)
            raises (WfBase::BaseException, NotEnabled);
    };

    interface WfProcess : WfExecutionObject {
        WfRequester requester()
            raises(WfBase::BaseException);
        void set_requester( in WfRequester new_value)
            raises (WfBase::BaseException);
        long how_many_step()
            raises (WfBase::BaseException);
        WfActivityIterator get_iterator_step()
            raises (WfBase::BaseException);
        WfActivitySequence get_sequence_step(
            in long max_number )
            raises (WfBase::BaseException);
        boolean is_member_of_step(
            in WfActivity member )
            raises (WfBase::BaseException);
        WfProcessMgr manager()
            raises(WfBase::BaseException);
        ProcessData result()
            raises (WfBase::BaseException, ResultNotAvailable);
        void start()
            raises (WfBase::BaseException, CannotStart,
            AlreadyRunning);
    };

    interface WfProcessIterator : WfBase::BaseIterator {
        WfProcess get_next_object ()
            raises (WfBase::BaseException);
        WfProcess get_previous_object()
            raises (WfBase::BaseException);
        WfProcessSequence get_next_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
        WfProcessSequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};
```

```
interface WfActivity  : WfExecutionObject, WfRequester {
    long how_many_assignment()
        raises (WfBase::BaseException);
    WfAssignmentIterator get_iterator_assignment()
        raises (WfBase::BaseException);
    WfAssignmentSequence get_sequence_assignment(
        in long max_number )
        raises (WfBase::BaseException);
    boolean is_member_of_assignment(
        in WfAssignment member )
        raises (WfBase::BaseException);
    WfProcess container()
        raises(WfBase::BaseException);
    ProcessData result()
        raises(WfBase::BaseException, ResultNotAvailable);
    void set_result(
        in ProcessData result)
        raises (WfBase::BaseException, InvalidData);
    void complete()
        raises (WfBase::BaseException, CannotComplete);
};

interface WfActivityIterator : WfBase::BaseIterator {
    WfActivity get_next_object ()
        raises (WfBase::BaseException);
    WfActivity get_previous_object()
        raises (WfBase::BaseException);
    WfActivitySequence get_next_n_sequence(
        in long max_number)
        raises (WfBase::BaseException);
    WfActivitySequence get_previous_n_sequence(
        in long max_number)
        raises (WfBase::BaseException);
};

interface WfAssignment  : WfBase::BaseBusinessObject {
    WfActivity activity()
        raises(WfBase::BaseException);
    WfResource assignee()
        raises(WfBase::BaseException);
    void set_assignee(
        in WfResource new_value)
        raises (WfBase::BaseException);
    assignment_stateType assignment_state()
        raises(WfBase::BaseException);
    void set_assignment_state(
        in assignment_stateType new_value)
        raises (WfBase::BaseException, TransitionNotAllowed);
};
```

```
interface WfAssignmentIterator : WfBase::BaseIterator {
    WfAssignment get_next_object ()
        raises (WfBase::BaseException);
    WfAssignment get_previous_object()
        raises (WfBase::BaseException);
    WfAssignmentSequence get_next_n_sequence(
        in long max_number)
        raises (WfBase::BaseException);
    WfAssignmentSequence get_previous_n_sequence(
        in long max_number)
        raises (WfBase::BaseException);
};

interface WfResource : WfBase::BaseBusinessObject {
    long how_many_work_item()
        raises (WfBase::BaseException);
    WfAssignmentIterator get_iterator_work_item()
        raises (WfBase::BaseException);
    WfAssignmentSequence get_sequence_work_item(
        in long max_number )
        raises (WfBase::BaseException);
    boolean is_member_of_work_items(
        in WfAssignment member )
        raises (WfBase::BaseException);
    string resource_key()
        raises(WfBase::BaseException);
    string resource_name()
        raises(WfBase::BaseException);

    void release(
        in WfAssignment from_assigment,
        in string release_info)
        raises (WfBase::BaseException, NotAssigned);
};

interface WfEventAudit : WfBase::BaseBusinessObject {

    WfExecutionObject source()
        raises(WfBase::BaseException, SourceNotAvailable);

    TimeBase :: UtcT timestamp()
        raises(WfBase::BaseException);
    string event_type()
        raises(WfBase::BaseException);
    string activity_key()
        raises(WfBase::BaseException);
    string activity_name()
        raises(WfBase::BaseException);
    string process_key()
        raises(WfBase::BaseException);
    string process_name()
```

```
            raises(WfBase::BaseException);
        string process_mgr_name()
            raises(WfBase::BaseException);
        string process_mgr_version()
            raises(WfBase::BaseException);
};

interface WfEventAuditIterator : WfBase::BaseIterator {
        WfEventAudit get_next_object ()
            raises (WfBase::BaseException);
        WfEventAudit get_previous_object()
            raises (WfBase::BaseException);
        WfEventAuditSequence get_next_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
        WfEventAuditSequence get_previous_n_sequence(
            in long max_number)
            raises (WfBase::BaseException);
};

interface WfCreateProcessEventAudit : WfEventAudit {
        string p_activity_key()
            raises(WfBase::BaseException);
        string p_process_key()
            raises(WfBase::BaseException);
        string p_process_name()
            raises(WfBase::BaseException);
        string p_process_mgr_name()
            raises(WfBase::BaseException);
        string p_process_mgr_version()
            raises(WfBase::BaseException);
};

interface WfStateEventAudit : WfEventAudit {
        string old_state()
            raises(WfBase::BaseException);
        string new_state()
            raises(WfBase::BaseException);
};

interface WfDataEventAudit  : WfEventAudit {
        ProcessData old_data()
            raises(WfBase::BaseException);
        ProcessData new_data()
            raises(WfBase::BaseException);
};

interface WfAssignmentEventAudit : WfEventAudit {
        string old_assignment_state()
            raises(WfBase::BaseException);
        string old_resource_key()
```

```
                raises(WfBase::BaseException);
            string old_resource_name()
                raises(WfBase::BaseException);
            string new_assignment_state()
                raises(WfBase::BaseException);
            string new_resource_key()
                raises(WfBase::BaseException);
            string new_resource_name()
                raises(WfBase::BaseException);
        };
};
#endif
```

CDL A

The following describes the workflow model in terms of the Component Definition Language (CDL) that is part of the proposed Business Object Component Architecture (Boca). The specification is included to illustrate the relationship of this specification with the Boca; it is not a normative part of the specification.

A.1 Consolidated CDL

```
#include <BocaFramework.cdl>
collection_kind Manager {};
#define ReducedIdl

module WorkflowModel  {
   //Forward references
   business_object WfExecutionObject;
   process WfProcess;
   business_object WfActivity;
   business_event WfEventAudit;

   struct DataInfoType {
      string attribute_name;
      string type_name;
   };

   typedef sequence<DataInfoType> ProcessDataInfo;

   struct NameValue {
      string aname;
      any avalue;
   };
```

```
typedef sequence<NameValue> ProcessData;

//Forward references
business_object WfProcessMgr;
business_object WfExecutionObject;
process WfProcess;
business_object WfActivity;
business_object WfRequester;
business_event WfEventAudit;
business_object WfResource;
entity WfAssignment;

[is_abstract]
business_object WfRequester {
   [is_read_only]
   relationship performer Many WfProcess inverse requester;
   // Operations invoked from related workflows
   void receive_event( in WfEventAudit event );
};

[is_abstract]
business_object WfProcessMgr  {
   exception NotEnabled {};

   [is_read_only]
   relationship process Aggregates WfProcess inverse manager;

   WfProcess create_process (in WfRequester requester)
         raises (NotEnabled);

   [is_read_only] attribute boolean enabled;
   [is_read_only] attribute string name;
   [is_read_only] attribute string description;
   [is_read_only] attribute string category;
   [is_read_only] attribute string version;

   ProcessDataInfo get_context_signature();
   ProcessDataInfo get_result_signature();
};

[is_abstract, keys={key}]
business_object WfExecutionObject  {
   exception CannotSuspend {};
   exception AlreadySuspended {};
   exception CannotStop {};
   exception NotRunning {};
   exception CannotResume {};
   exception InvalidState {};

   // Workflow state model
   state_set workflow_state { open, closed };
```

```
during (open) {
      state_set while_open { not_running, running };
      during (not_running) {
            state_set why_not_running { not_started, suspended };
            during (suspended) {
               signal resume()raises (CannotResume);
            };
      };
      during (running) {
         signal suspend() raises (CannotSuspend, CurrentlySuspended);
      };
      signal terminate() raises (CannotStop, NotRunning);
      signal abort() raises (CannotStop, NotRunning);
};
during (closed) {
      state_set how_closed {completed, terminated, aborted };
};

// Attributes
attribute string name;
[is_read_only] attribute string key;
attribute string description;
attribute ProcessData process_context;
//[annotation="Lower numbers have greater priority"]
[constraint=((priority>=1) && (priority<=5))]
attribute unsigned short priority = 3;
[is_read_only] attribute valid_states;


[is_read_only]
relationship history Aggregates WfEventAudit inverse source;


// Dynamic state transitions
[is_query]
string get_current_state();
void change_state( in string new_state ) raises (InvalidState);

// Rules
apply StateTransitionRule terminate_trans {
      trigger = {terminate};
      source = open;
      target = terminated;
};
apply StateTransitionRule abort_trans {
      trigger = {abort};
      source = open;
      target = aborted;
};
apply StateTransitionRule suspend_trans {
      trigger = {suspend};
```

```
            source = running;
            target = suspended;
        };
        apply StateTransitionRule resume_trans {
            trigger = {resume};
            source = suspended;
            target = running;
        };
};

[is_abstract]
process WfProcess : WfExecutionObject {
    exception CannotStart {};
    exception AlreadyRunning {};
     exception ResultNotAvailable {};

    [INITIALIZED]
    relationship requester References 1..1 WfRequester inverse performer;
    [is_read_only]
    relationship step Aggregates WfActivity inverse container;
    relationship manager IsPartOf WfProcessMgr inverse process;

    signal start() raises(CannotStart,AlreadyRunning);

    ProcessData get_result() raises(ResultNotAvailable);

    // Rules
    apply StateTransitionRule start_trans {
            trigger = {start};
            source = not_running;
            target = running;
    };
};

[is_abstract]
business_object WfActivity : WfExecutionObject, WfRequester {
            exception CannotComplete {};

    [is_read_only]
    relationship assignment Aggregates WfAssignment inverse activity;
    [is_read_only]
    relationship container IsPartOf WfProcess inverse step;

    void set_result( in NameValues result );

    signal complete() raises(CannotComplete);

    apply StateTransitionRule complete_trans {
            trigger = {complete};
            source = open;
            target = completed;
```

```
   };
};

// treat status as a state rather than enum
enum /*state_set*/ assignment_state { potential, accepted };

[keys={activity, assignee}, is_abstract]
entity WfAssignment  {
exception InvalidResource {};

   [is_read_only, INITIALIZED]
   relationship activity IsPartOf WfActivity inverse assignment;
   relationship assignee References 1..1 WfResource inverse work_item;
   attribute assignment_state assignment_state;

   // create new relationship to a different resource
   void reassign ( in WfResource new_resource,
                   in AssignmentStatus new_status ) raises(InvalidResource);
};

typedef sequence<WfAssignment> WfAssignmentSequence;

[keys={resource_key},is_abstract]
business_object WfResource  {
   [is_read_only]
   relationship work_item Many WfAssignment inverse assignee;

   attribute string resource_key;
   attribute string resource_name;

   // Inform resource that the workflow no longer needs it
   void release( in WfAssignment from_assigment
                 in string release_info);
};

[FROZEN, is_abstract]
business_event WfEventAudit {
   [is_read_only] relationship source IsPartOf WfExecutionObject;
   readonly attribute TimeBase::UtcT timestamp;
   readonly attribute string event_type;
   readonly attribute string activity_key;
   readonly attribute string activity_name ;
   readonly attribute string process_key ;
   readonly attribute string process_mgr_name;
   readonly attribute string process_mgr_version;
   readonly attribute string domain_id; // BSD of source
};

[is_abstract]
business_event WfCreateProcessEventAudit : WfEventAudit {
   readonly attribute string activity_key;
```

```
        readonly attribute string process_key ;
        readonly attribute string process_mgr_name;
        readonly attribute string process_mgr_version;
        readonly attribute string domain_id; // BSD of parent
    };

    [is_abstract]
    business_event WfStateEventAudit : WfEventAudit {
        readonly attribute string old_state;
        readonly attribute string new_state;
    };

    [is_abstract]
    business_event WfDataEventAudit : WfEventAudit {
        readonly attribute ProcessData old_data;
        readonly attribute ProcessData new_data;
    };

    [is_abstract]
    business_event WfAssignmentEventAudit : WfEventAudit {
            readonly attribute string old_assignment_state;
            readonly attribute string old_resource_key;
            readonly attribute string old_resource_name;
            readonly attribute string new_assignment_state;
            readonly attribute string new_resource_key;
            readonly attribute string new_resource_name;
    };
}; // End - Workflow
```