Negotiation Facility Specification

Version 1.0
March 2002

PATENT

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

NOTICE

The information contained in this document is subject to change without notice. The material in this document details an Object Management Group specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents & Specifications, Report a Bug/Issue.

Contents

Contents

Contents

Contents

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 600 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

OMG Documents

The OMG documentation is organized as follows:

OMG Modeling

- ***Unified Modeling Language (UML) Specification*** defines a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.
- ***Meta-Object Facility (MOF) Specification*** defines a set of CORBA IDL interfaces that can be used to define and manipulate a set of interoperable metamodels and their corresponding models.
- ***OMG XML Metadata Interchange (XMI) Specification*** supports the interchange of any kind of metadata that can be expressed using the MOF specification, including both model and metamodel information.

Object Management Architecture Guide

This document defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.

CORBA: Common Object Request Broker Architecture and Specification

Contains the architecture and specifications for the Object Request Broker.

OMG Interface Definition Language (IDL) Mapping Specifications

These documents provide a standardized way to define the interfaces to CORBA objects. The IDL definition is the contract between the implementor of an object and the client. IDL is a strongly typed declarative language that is programming language-independent. Language mappings enable objects to be implemented and sent requests in the developer's programming language of choice in a style that is natural to that language. The OMG has an expanding set of language mappings, including Ada, C, C++, COBOL, IDL to Java, Java to IDL, Lisp, and Smalltalk.

CORBAservices

Object Services are general purpose services that are either fundamental for developing useful CORBA-based applications composed of distributed objects, or that provide a universal-application domain-independent basis for application interoperability.

These services are the basic building blocks for distributed object applications. Compliant objects can be combined in many different ways and put to many different uses in applications. They can be used to construct higher level facilities and object frameworks that can interoperate across multiple platform environments.

Adopted OMG Object Services are collectively called CORBAservices and include specifications such as *Collection*, *Concurrency*, *Event*, *Externalization*, *Naming*, *Licensing*, *Life Cycle*, *Notification*, *Persistent Object*, *Property*, *Query*, *Relationship*, *Security*, *Time*, *Trader*, and *Transaction*.

CORBAfacilities

Common Facilities are interfaces for horizontal end-user-oriented facilities applicable to most domains. Adopted OMG Common Facilities are collectively called CORBAfacilities and include specifications such as *Internationalization and Time*, and *Mobile Agent Facility.*

Object Frameworks and Domain Interfaces

Unlike the interfaces to individual parts of the OMA “plumbing” infrastructure, Object Frameworks are complete higher level components that provide functionality of direct interest to end-users in particular application or technology domains.

Domain Task Forces concentrate on Object Framework specifications that include Domain Interfaces for application domains such as Finance, Healthcare, Manufacturing, Telecoms, E-Commerce, and Transportation.

Currently, specifications are available in the following domains:

- *CORBA Business*: Comprised of specifications that relate to the OMG-compliant interfaces for business systems.
- *CORBA Finance*: Targets a vitally important vertical market: financial services and accounting. These important application areas are present in virtually all organizations: including all forms of monetary transactions, payroll, billing, and so forth.
- *CORBA Healthcare*: Comprised of specifications that relate to the healthcare industry and represents vendors, healthcare providers, payers, and end users.
- *CORBA Manufacturing*: Contains specifications that relate to the manufacturing industry. This group of specifications defines standardized object-oriented interfaces between related services and functions.
- *CORBA Telecoms*: Comprised of specifications that relate to the OMG-compliant interfaces for telecommunication systems.
- *CORBA Transportation*: Comprised of specifications that relate to the OMG-compliant interfaces for transportation systems.

Obtaining OMG Documents

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters

250 First Avenue

Needham, MA 02494

USA

Tel: +1-781-444-0404

Fax: +1-781-444-0320

pubs@omg.org

http://www.omg.org

Summary of Key Features

The CORBA Electronic Commerce Domain architecture is comprised of specifications that relate to the OMG-compliant interfaces for distributed electronic commerce systems. Currently, there are four frameworks established as a result of the Negotiation Facility RFP2. These include the Session Framework, Community Framework, Collaboration Framework, and DomFramework.

The Framework Specification presented in Chapters 3 and 4 are targeting potential developers of this facility. Information is presented in the form of a breakdown of modules, interfaces, and types. For each interface, details of attributes, operations, events and additional semantics are provided. The documentation assumes that readers are familiar with the object model defined under the Task/Session specification (formal/00-05-03), and have familiarity with the notion of structured events as defined by CosNotification.

1. Negotiation and Contract Criteria - The specification of three collaboration criteria instances covering:
   - bilateral negotiation
   - multilateral negotiation
   - promissory commitment
2. The Collaboration Framework chapter contains the definition of Collaboration, a process through which different models of collaboration rules can be managed. The **CollaborationFramework** is defined extensively on interfaces from **CommunityFramework**.

3. Community Framework under contains extensions to the Task and Session support communities of collaborating users. It defines the abstract Membership interface and concrete types - Community, Agency.

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Acknowledgments

The following companies have submitted to or have supported submissions contributing to this specification:

- Fraunhofer Institut Materialfluss und Logistik
- Imperial College of Science Technology and Medicine
- In-Line Software
- OSM SARL
- Sprint - Technology Planning and Integration
- Xerox Corporation

Collaboration Criteria 1

Contents

This chapter contains the following sections.

1.1 Introduction

This chapter describes three collaboration models dealing with bilateral negotiation, multilateral negotiation, and promissory commitment. Each model is presented with a general description of the model purpose and characteristics, followed by the specification of the structure and values of **Criteria** instances used to represent the model.

Criteria descriptions are defined through construction of instances based on valuetypes defined in the *CollaborationFramework* and *CommunityFramework* chapters of this document. Composition of valuetypes is described using the Digital Product Modeling Language (DPML) XML schema. DPML is a non-normative supplement to this specification that allows a more complete representation of Criteria instances than is possible under IDL. The DPML 2.0 DTD and mapping to CommunityFramework and CollaborationFramework valuetypes is presented in Section 1.3, “DPML Schema Specification,” on page 1-17.

Table 1-1 Criteria Descriptions

Model label	Model Description
bilateral	A model of collaboration in which two parties can interact through offers, requests, suggestions, and proposals leading to an agreed or non-agreed conclusion.
multilateral	A model of collaboration in which an initiating party can establish a motion, a reciprocating party can second the motion, supporting actions enable motion amendment (through amendment motions), leading to a vote on the motion and possible establishment of an agreed result.
promissory	A model of collaboration in which an initiating party can establish a promise towards another party, where the reciprocating party can call the promise, thereby establishing an obligation on the promising party, leading to the launching of a fulfillment process (defined under a separate processor model).

1.2 Collaborative Process Models

1.2.1 Bilateral Negotiation

This section describes a model of collaboration in which two parties can interact through offers, requests, suggestions, and proposals leading to an agreed or non-agreed conclusion. The model expressed here in DPML defines the structure and values of a **CollaborationModel** contained within a **ProcessorCriteria** that may be executed under a **CollaborationProcessor**. The mappings between DPML elements and criteria valuetype are presented in Section 1.3, “DPML Schema Specification,” on page 1-17. Definition of the control valuetypes and the supporting interfaces are presented in Chapter 2 “*CollaborationFramework*” and Chapter 3 “*CommunityFramework*.”

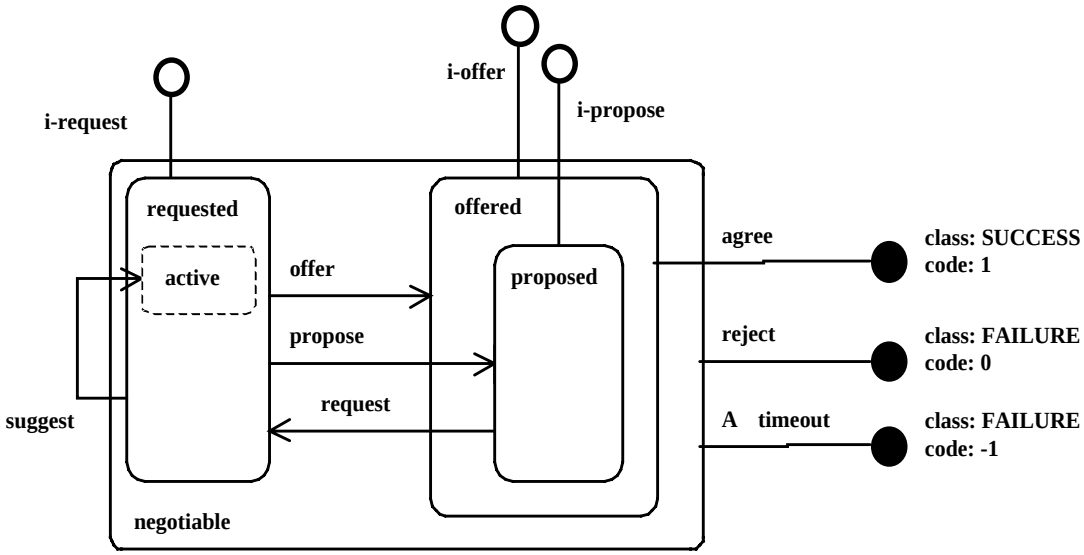


Figure 1-1 Schematic Representation

1.2.1.1 DPML Specification

<DPML>

<collaboration label="bilateral" note="Bilateral negotiable agreement process model.">

This model defines a bilateral process through which two parties may attempt to establish an agreement through a pattern of interaction similar to the classic notions of peer-to-peer negotiation. The process enables the establishment of a negotiation subject, an initial offered, proposed, or requested state, and transitions supporting the escalation of the level of mutual agreement between parties qualified by the implicit PARTICIPANT roles of INITIATOR and RESPONDENT. The model demonstrates the application of input and output declarations, a simple state hierarchy, initializations, transitional actions, terminations, and usage directives.

<input tag="subject" required="TRUE"
type="IDL:omg.org/Session/AbstractResource:2.0" />

The establishment of the subject of a negotiation is controlled by the addition of an input usage constraint on the **CollaborationModel** (refer to “InputDescriptor” on page 1-18 and 1-29). This input descriptor declares a requirement for the association of a tagged usage link named “subject” when initializing the hosting process. Initialization of the process is achieved by invoking **apply_arguments** on the hosting **CollaborationProcessor**. The client passing a string identifying an initialization argument (one of the values of “**init.offer**,” “**init.propose**,” or “**init.request**”) and an

ApplyArgument value containing the name of the input usage constraint (in this case “subject”) together with an instance of **AbstractResource** that will constitute the initial subject of the collaboration.

<state label="negotiable" >

The **negotiable** state is a parent state to the two states **proposed** and **requested**. Transitions declared on the **negotiable** state enable the explicit rejection of a **subject** by a user through the **reject** termination. A second characteristic of the **negotiable** state is the association of a **timeout** transition that will close the negotiation after a predetermined period of inactivity.

```
<trigger label="reject" >
    <launch mode="PARTICIPANT" />
    <termination class="FAILURE" code="0" />
</trigger>
```

The reject trigger declares the possibility to any PARTICIPANT to terminate the collaboration under a **FAILURE** status. A **reject** transition may be invoked against any **open** (**proposed**, **requested**, or **offered**) state.

```
<trigger label="timeout" >
    <clock timeout="3600000" />
    <termination class="FAILURE" code="-1" />
</trigger>
```

The **timeout** trigger declares a default termination condition, armed when the negotiation state becomes active. The value represents the period between arming and firing by a **CollaborationProcessor** implementation. DPML represents time periods in micro-seconds.

<state label="requested" >

```
<trigger label="init.request" >
    <launch mode="INITIATOR" />
    <initialization/>
</trigger>
```

The **requested** state exposes transitions that allow a respondent to transition to the **offered** or **proposed** states using the **offer** or **propose** transitions, or to continue in the **requested state** through application of the **suggest** transition.

```
<trigger label="suggest" >
    <launch mode="RESPONDENT"/>
    <local reset="TRUE">
        <input tag="subject" required="TRUE" implied=”FALSE”
            type="IDL:omg.org/Session/AbstractResource:2.0" />
    </local>
</trigger>
```

The **suggest** transition is a local transition with **reset** semantics enabled. Semantically it is equivalent to the **request** transition except that it is initiated under the **requested** state. **Suggest** is used as an exploratory mechanism through which two members can continue to invoke suggestions towards each other relative to the subject, until such time that at least one party is ready to migrate to a higher level of commitment as expressed under the **proposed** or **offered** states.

```
<trigger label="offer" >
    <launch mode="RESPONDENT"/>
    <transition target="offered">
                <input tag="subject"
        required="TRUE" implied="FALSE"
        type="IDL:omg.org/Session/AbstractResource:2.0" />
                    </transition>
</trigger>
```

An **offer** is a transition from the **requested** state to the **offered** state. Invoking **offer** is on one hand an expression of agreement by the offering party, but on the other hand, restricts the potential for further negotiation (as compared to propose).

```
    <trigger label="propose" >
        <launch mode="RESPONDENT"/>
        <transition target="proposed">
                    <input tag="subject"
                    required="TRUE" implied="FALSE"
                    type="IDL:omg.org/Session/AbstractResource:2.0" />
        </transition>
    </trigger>

</state>
```

Propose is a transition from the **requested** to **proposed** states that introduces the commitment by the proposing party in that the subject of the proposal may be agreed to by the correspondent. This is distinct to the requested state where, in comparison, no agreement is implied.

```
<state label="offered" >

    <trigger label="init.offer" >
        <launch mode="INITIATOR" />
        <initialization/>
    </trigger>
```

The **offered state** enables a respondent to **agree** or **reject** an agreement to the subject of the collaboration. Invoking **agree** leads to the firing of a successful terminal transition expressing agreement by both parties to the **subject** of the **Collaboration**.

```
<trigger label="agree" >
    <launch mode="RESPONDENT" />
    <move source="subject" target="result" switch="TRUE"/>
    <termination class="SUCCESS" code="1">
```

```
            <output tag="result"
        type="IDL:omg.org/Session/AbstractResource:2.0" />
    </termination>
</trigger>
```

The **agree** Trigger is available to a respondent under the **offered** and **proposed** states. **Agree** signifies the agreement by the respondent to an **offer** or **proposal** raised by the issuing user. The **agree** transition establishes a collaboration process under an **agreed** termination, expressing the agreement by both parties to the **subject** of a collaboration. Agree contains an output descriptor declaring the **result** tag, established under the move directive.

```
<state label="proposed" >

    <trigger label="init.propose" >
        <launch mode="INITIATOR" />
        <initialization/>
    </trigger>
```

The **proposed** state extends the semantics of the **offered** state by introducing the possibility of change to the subject of the collaboration. Through application of the **request** transition, a respondent may change the subject of the collaboration to a new value and establish the active state as **requested**.

```
    <trigger label="request" >
        <launch mode="RESPONDENT"/>
        <transition target="requested">
                <input tag="subject"
                required="TRUE" implied="FALSE"
                type="IDL:omg.org/Session/AbstractResource:2.0" />
        </transition>
    </trigger>
```

Request is a simple transition that can be applied under the **proposed** state. **Request** enables a respondent to change the subject of a negotiation and the context from the **proposed** to **requested** state. A **request** transition does not signify the commitment of the requesting party, however, it opens the possibility for the counterpart to respond with **propose** or **offer** against the **subject** under the **requested** state.

```
            </state>
        </state>
    </state>
</collaboration>
</DPML>
```

1.2.2 Multilateral Agreement

A Multilateral agreement model describes a collaboration criteria in which an initiating party can establish a motion, a reciprocating party can second that motion, and supporting actions that enable motion amendment (through amendment motions), leading

to a vote on the motion and possible establishment of an agreed result. This model demonstrates the application of compound transitions dealing with voting and motion amendment. In the case of motion amendment, the compound transition is an example of a model recursion (multilateral declares amend which is a compound transition that references multilateral as the controlling model).

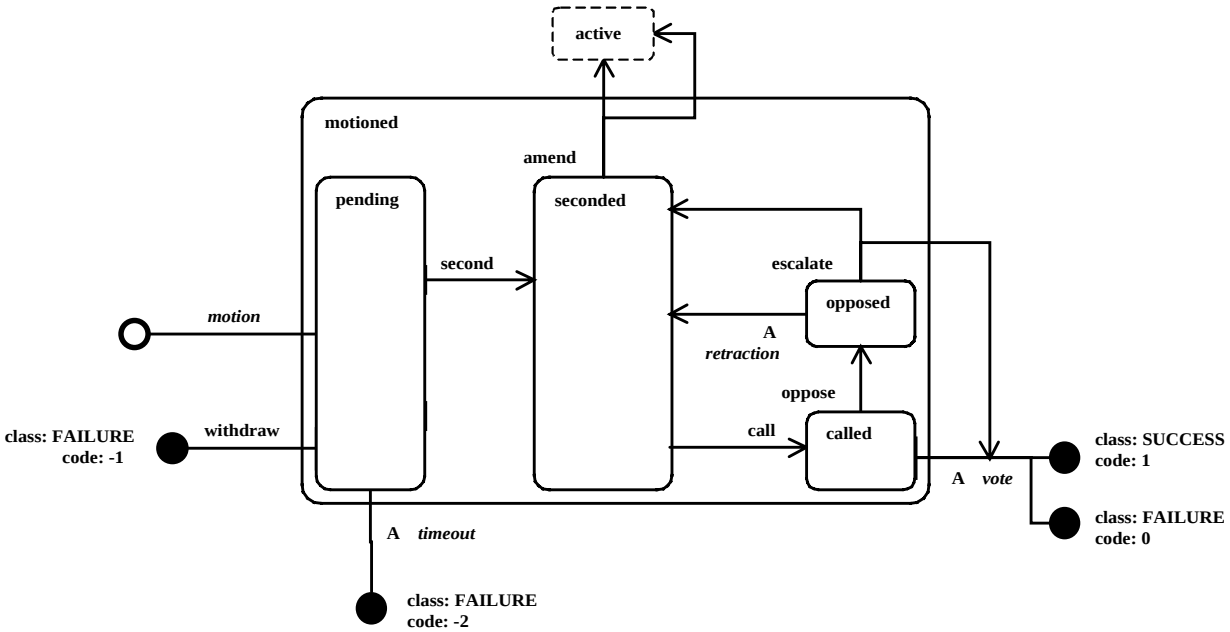


Figure 1-2 Schematic Representation

1.2.2.1 DPML Specification

```
<DPML>
<collaboration label="multilateral"
    note="Multilateral agreement through motion, amendment and voting">
```

A **motion**-based negotiation is a collaborative process model dealing with interactions between a group of two or more participants. It provides a framework within which a user can initiate a **motion** with an arbitrary **subject** under which agreement can be established through a consensus process.

```
<input tag="subject"
    required="TRUE" implied="FALSE"
    type="IDL:omg.org/Session/AbstractResource:2.0" />
```

The subject input declaration requires that the **Task** associated to the hosting processor must be explicitly associated with a named usage tag (prior to processor start or during initalization). This resource represents the motion being raised.

<state label="motioned" >

The motioned state is the parent of two principal states, **pending** and **seconded** that through interaction between participants may lead to any of the terminal transitions of **agree**, **reject**, or **withdraw**. Initialization of a multilateral **motion** is established through a **motion** trigger, establishing the invoking user as the INITIATOR. Under the pending state two actions are possible:

- The **withdraw** transition may be launched either directly by the initiator, or though a **timeout** referral that will raise the withdraw termination.

  or

- A RECIPROCATING user (any user other than the user raising the motion) may **second** the motion leading to a transition to the seconded state.

Once a **pending** motion is **seconded**, any user may invoke the **amend** or **call** triggers. Amend is executed as a full motion process whereas the call transition changes the active state to called. Under the called state the process may be opposed resulting in the potential withdrawal of the call or call escalation. The **escalate** trigger forces a 2/3 majority vote-to-vote, the successful outcome of which is mapped to a compound transition involving a formal vote. The failure of the vote-to-vote is mapped to a transition back to the seconded state. The **vote** trigger fires a voting compound transition that contains a **ProcessorCriteria** containing a **VoteModel** instance as the sub-processor definition. The sub-processor, an instance of **VoteProcessor** exposes a **vote** operation under which participants may register YES, NO, and ABSTAIN. The success or failure of a vote processor is mapped to an **agree** and **reject** termination that signal the success or failure of the multilateral process.

<state label="pending" >

The **pending** state signifies the agreement by one party to a motion, expressed as the **subject** of **Collaboration** and the expression of the interest of that party in the reaching of agreement to the associated subject. The issuing user may **withdraw** a motion at any time prior to **second** transition. A **timeout** terminal transition will fire after a predetermined interval if a motion is not seconded. A **second** transition establishes the motion as a valid motion to the **Membership**.

```
<trigger label="motion" >
    <launch mode="INITIATOR" />
    <initialization/>
</trigger>
```

Initialization using **motion** establishes the collaboration with the **pending** state and all parents as the **active-state** path. A motion is raised with the express interest of gaining the agreement (or rejection) of the membership to the subject of the motion. For a motion to be successful, the motion must be seconded and voted upon prior to the timeout of the withdraw action. At any time before a motion vote is initiated the principal raising the motion may actively withdraw the motion. A potential risk of raising a motion is that the subject of the motion, if seconded, may be amended at the discretion of the group.

```
<trigger label="second" >
    <launch mode="RESPONDENT" />
    <transition target="seconded" />
</trigger>
```

The **second** transition is a simple transition that may be invoked by a **respondent** in support of a **pending** motion. The second transition will result in the establishment of the **seconded** state and all **parent** states as the active-state path. Once a motion is seconded it may no longer be withdrawn and may be subject to amendment by the members of the collaboration.

```
<trigger label="withdraw" >
    <launch mode="INITIATOR" />
    <termination class="FAILURE" code="0" />
</trigger>
```

The initiator of a motion may withdraw the motion at any time prior to the occurrence of a second action.

```
<trigger label="timeout" >
    <clock timeout="120000" />
    <termination class="FAILURE" code="-1" />
</trigger>
```

A **timeout** trigger will force termination of the process in the absence of a second to the motion.

```
</state>
```

```
<state label="seconded">
```

The **seconded** state establishes the process in a mode that disables the potential for motion withdrawn and raises the possibility for amendment of the motion or potential calling of a vote on the motion.

```
<trigger label="amend" >
        <launch mode="PARTICIPANT"/>
        <move source="subject" target="subject.pending" />
        <external label="amending"
            public="-OSM//XML Model::MULTILATERAL//EN"
            system="http://home.osm.net/dpml/multilateral.xml">
        </external>
```

The **amend** Trigger contains a compound transition defined by a subsidiary collaboration process using the **motion** model; that is, this model. To circumvent recursion restrictions within XML, the external element is used to indirectly reference the multilateral agreement model. Using the **apply_arguments** operation on **CollaborationProcessor**, the client passes in an identifier referencing the **Trigger** label (amend) together with an **ApplyArgument** value containing the “subject” usage label and an object representing the amendment. An amendment is executed as a sub-process under which the amended subject is raised as a new motion, subject to a second, and subsequent vote by the membership.

```
<on class="SUCCESS">
    <remove source="subject.pending"/>
    <move source="result"
        target="subject" switch="TRUE"/>
    <local reset="TRUE"/>
</on>
```

On conclusion of the amendment process, a successful **result** of the underlying process will cause the completion of the transition by changing the **active-state** to **seconded** and the assertion of the sub-process result as the seconded subject.

```
<on class="FAILURE">
    <remove source="subject" />
    <move source="subject.pending" target="subject"/>
    <local reset="TRUE"/>
</on>
```

In the case of failure of the sub-process, the subject of the amendment sub-process is removed and the original subject is reinstated using the remove and move usage directives.

```
</trigger>
```

```
<trigger label="call" >
    <launch mode="PARTICIPANT" />
    <transition target="called" />
</trigger>
```

The **call** trigger may be invoked by any participant. It moves the process to a state that prevents further amendment.

```
</state>
```

```
<state label="called" >
```

The **called** state contains a vote clock, armed when the called state becomes active. The automatic launching of a vote can be disabled through the oppose trigger, forcing the **Collaboration** into an opposed state. If no participant opposes the call, a vote process will be automatically established.

```
<trigger label="vote" >
    <clock timeout="120000"/>
    <vote label="voting"
        policy="AFFERMATIVE" numerator="1" denominator="2">
        <input tag="subject" required="TRUE" implied="TRUE"
        type="IDL:omg.org/Session/AbstractResource:2.0" />
    </vote>
```

The **vote** trigger is guarded by a timeout condition. It is armed when the containing state enters the active state path. The model declares a **ProcessorCriteria** value containing a vote model (refer VoteModel) and an input pre-condition that implicitly associates the current subject as the subject of the voting process.

```
<on class="SUCCESS">
    <move source="subject" target="result" switch="TRUE" />
    <termination class="SUCCESS" code="1">
        <output tag="result"
        type="IDL:omg.org/Session/AbstractResource:2.0" />
    </termination>
</on>
```

Post-conditions of the vote are expressed under the “on” statements (representing Map instances). On **SUCCESS** the subject usage link of the collaboration’s task is moved to “result.” The switch attribute signifies that the **Collaboration** implementation will switch the link containing the subject from consumed (input) to produced (output) as a post-condition to termination execution prior to process completion.

```
<on class="FAILURE">
    <termination class="FAILURE" code="0" />
</on>
```

On **FAILURE** of the vote, the process is terminated with its own failure status.

```
</trigger>

<trigger label="oppose" >
    <launch mode="RESPONDENT" />
    <transition target="opposed" />
</trigger>
```

The **oppose Trigger** enables declaration of opposition to the calling of a vote by transition to the **opposed State**.

```
</state>

<state label="opposed" >
```

The **opposed** state supports automatic retraction of a call under a timeout condition. Any member of the collaboration can intercept automatic timeout by invoking the escalate **Trigger**, forcing a vote-to-vote.

```
<trigger label="retraction" >
    <clock timeout="120000" />
    <transition target="seconded" />
</trigger>
```

The **retraction** trigger is armed when the opposed state enters the active state path. It declares a simple transition to the seconded state. Automatic retraction may be intercepted by the **escalate** trigger.

```
<trigger label="escalate" >
        <launch mode="RESPONDENT" />
```

The **escalate** trigger forces suspension of a retraction countdown by launching a **vote-to-vote** sub-processor.

```
<vote label="vote-to-vote"
    policy="AFFERMATIVE"
    numerator="2"
    denominator="3"/>
```

The **vote-to-vote** is a compound transition containing policy that defines vote rules to be applied, in this case an affirmative 2/3 majority is required for the vote processor to conclude with a successful result.

```
<on class="SUCCESS">
    <referral action="voting" />
</on>
```

On success of the **vote-to-vote** sub-processor, a referral action launches a normal vote process, which will establish a finalization of the processor in a successful or failed state.

```
<on class="FAILURE">
    <transition target="seconded" />
</on>
```

On failure of the **vote-to-vote** a simple transition to the seconded state is fired, enabling a resumption of subject amendment.

```
            </trigger>
        </state>
    </state>
</collaboration>
</DPML>
```

1.2.3 Promissory Contract Fulfillment

The promissory contractual fulfillment model demonstrates the use of named roles as preconditions to trigger invocation. The model also includes reuse of the bilateral negotiation model as the means by which a commercial contract fulfillment process may be disputed and the means through which obligations of the contracting parties may be waived.

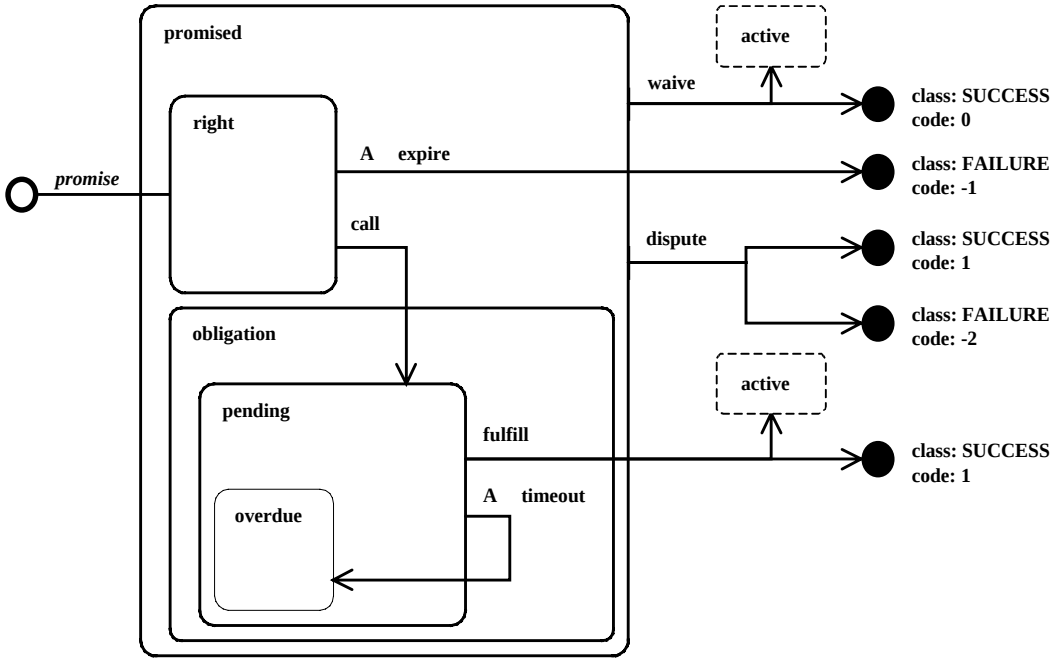


Figure 1-3 Schematic Representation

1.2.3.1 DPML Specification

```
<DPML>
<collaboration label="promissory" note="Promissory contract process model.">

    <role label="party" abstract="TRUE">
        <role.policy ceiling="1" quorum="1" assessment="STRICT"
                policy="CONNECTED" />
        <role label="supplier" abstract="FALSE"/>
        <role label="consumer" abstract="FALSE"/>
    </role>
```

The promissory contract fulfillment model contains a number of triggers that restrict the use of implicit role declarations such as INITIATOR and RESPONDENT. In this model the role guarding the call and fulfillment triggers are qualified by an explicit role declaration. One abstract role named “party” is defined as a container of two concrete roles named “supplier” and “consumer.” Both supplier and consumer role policies are implied by the policy definition of the containing party role. In this example both are declared with a quorum and ceiling of one. This means that the maximum number of members associated with this role is one and the minimum number is one. The policy

description also states that both users must be connected (refer to the *Task and Session* specification, User, Connected State section) and that quorum assessment shall be strictly applied.

```
<input tag="contract"
    required="TRUE"
    type="IDL:omg.org/Session/AbstractResource:2.0" />
```

The promissory contract model is defined with a required **Consumption** association between the coordinating Task and the processor with a tag corresponding to “contract.” This declaration establishes the requirements on a supplier to ensure that a tagged consumes link with the value “contract” is available prior to or during initialization of the hosting processor.

```
<state label="promised">
```

The **promissory** model defines a bilateral collaborative interaction. An initiator invoking a **promise** trigger establishes a **Collaboration** under the **right** state. Once initialized as a **right**, a respondent may call the promise by invoking a **call** transition. This corresponds to a **respondent** requesting fulfillment of the promise. An initiator of the promise (now in the role of respondent) fulfills a promise by applying the **fulfill** transition, itself a compound transition defined by a **bilateral** negotiation. Success of the negotiation leads to the **fulfilled** state whereas failure leads to the **rejected** state.

```
<trigger label="waive" >
```

The **waive** trigger may be invoked by either consumer or provider. It is a compound transition referencing a bilateral or multilateral negotiation that if successful results in a transition to the terminal **waived** state. A failure of the negotiation will result in the continuation of the process under the active state established prior to the initiation of the **waive** transition.

```
<launch mode="PARTICIPANT" />
    <external label="waiving"
        public="-OSM//XML Model::BILATERAL//EN"
        system="http://home.osm.net/dpml/bilateral.xml">
    </external>
```

An implementation of **Collaboration** establishes a new sub-process using the declared criteria – in this case the DPML supplies criteria references a bilateral negotiation process using an external (**ExternalCriteria**) declaration.

```
<on class="SUCCESS">
    <termination class="SUCCESS" code="0" />
</on>
```

A successful result of a negotiation by the participants is mapped to a successful termination of the promissory contract.

```
<on class="FAILURE">
    <local reset="FALSE"/>
 </on>
```

A failure result of a negotiation by the to participants in the attempt to waive the promise is mapped to a local transition to the last active state established prior to the initiation of the waive action.

```
</trigger>
```

```
<trigger label="dispute" >
    <launch mode="PARTICIPANT" />
    <copy source="contract" target="subject " />
    <external label="disputing"
        public="-OSM//XML Model::BILATERAL//EN"
        system="bilateral.xml">
    </external>
```

A **dispute** between a supplier and consumer can be established through applying the dispute trigger. A dispute may be initiated by either consumer or supplier. Prior to the initiation of the dispute sub-process, the contract association representing the promise is copied to a new link with the tag “subject,” required as an input to a bilateral negotiation process. In this example a bilateral negotiation is defined as the dispute resolution mechanism.

```
<on class="SUCCESS">
    <move source="result" target="contract" switch="TRUE"/>
    <local reset="TRUE"/>
</on>
```

At successful conclusion of a dispute the “subject.pending” link is removed and the result of the negotiation process is established as the active subject. Process execution is returned to the last active state.

```
<on class="FAILURE">
    <termination class="FAILURE" code="-2" />
 </on>
```

On failure of the dispute the process is terminated with a failed result.

```
</trigger>
```

A promise made by a provider towards a consumer under which the provider commits to the willingness to fulfill the promise at the request of the consumer.

```
 <state label="right">

    <trigger label="promise" >
        <launch role="supplier" />
        <initialization/>
    </trigger>
```

Initialization is achieved using the **promise** Trigger leading raised by a **supplier** facilitating the establishment of the promise offered under the subject of the process as a callable right of the **consumer**.

```
<trigger label="expire" >
    <clock timeout="12000000" />
    <termination class="FAILURE" code="-1" />
</trigger>
```

The **expire** trigger exposes a timeout value that will trigger the expiry of the consumer's right to invoke a **request** for fulfillment against a provider.

```
<trigger label="call" >
    <launch role="consumer" />
    <transition target="pending" />
</trigger>
```

The **call** trigger contains a transition to the pending state that is available to the consumer. Invoking the **call** transition establishes the promise as a **pending** obligation against the promise supplier.

```
 </state>

 <state label="obligation">
```

The **obligation** state establishes a collaborative context under which a promise constitutes an obligation of the provider to fulfill.

```
<state label="pending">
```

The **pending** state is a state under which a provider is obliged to **fulfill** on a promise through invocation of the fulfill transition.

```
 <trigger label="fulfill" >
    <launch role="supplier" />

    <external label="fulfillment"
        public="-OSM//XML Model::BILATERAL//EN"
        system="http://home.osm.net/dpml/bilateral.xml">
    </external>
```

Fulfill is available to a provider under the obligation **pending** state. A **fulfill** transition is defined as a compound transition that uses a bilateral negotiation criteria. A subsidiary **Collaboration** is instantiated that, on resolution, defines the success or failure condition used to determine the conclusion of the fulfillment action.

```
    <on class="SUCCESS">
        <move source="result" target="deliverable" />
        <termination class="SUCCESS" code="1">
            <output tag="deliverable"
            type="IDL:omg.org/Session/AbstractResource:2.0" />
```

```
            </termination>
        </on>
```

On successful completion of the fulfillment sub-process, the result of the fulfillment is established under a link tagged as the fulfillment “deliverable.” The implementation fires a success termination of the process, indicating satisfactory fulfillment of the promissory contract process.

```
 <on class="FAILURE">
    <local reset="TRUE"/>
</on>
```

On failure of the fulfillment sub-process a local transition is enabled following which the supplier is able to re-attempt fulfillment or potentially enter into a dispute resolution process or request a waive of the promise.

```
 </trigger>

<trigger label="timeout" >
    <clock timeout="240000000" />
    <transition target="overdue" />
</trigger>
```

Timeout is a clock controlled simple transition that changes an existing **obligation pending** to **obligating pending** and **overdue**.

```
<state label="overdue"/>
```

The **overdue** state is a sub-state of pending which is established by an implementation of **Collaboration** when a pending obligation timeout transition expires.

```
            </state>
        </state>
    </state>
</collaboration>
</DPML>
```

1.3 DPML Schema Specification

Digital Product Modeling Language (DPML) DTD specification 2.0.
Copyright OSM, 1999-2000
http://www.osm.net

This DTD defines the structural semantics of the data types used in the construction of digital products supporting distributed collaborative business process descriptions. This schema is a non-normative supplement supporting declaration of criteria composition related to this specification’s Collaboration and Community Frameworks. Descriptions of attributes and elements contained within this section are provided as a convenience. The formal specification of objects models and associated semantics are defined under

the specification of valuetypes and interfaces within Chapter 2- "*CollaborationFramework*" and Chapter 3- "*CommunityFramework*" based on the mapping of element to types contained at the end of this section.

Criteria

The criteria **ENTITY** is defined as the set of concrete criteria types that can be contained as the root element within a DPML document. The DPML root **ELEMENT** declaration defines the set of elements types that can be declared as a root element. The elements

- generic (**GenericCriteria**),
- community (**CommunityCriteria**),
- agency (**AgencyCriteria**),
- encounter (**EncounterCriteria**),
- external (**ExternalCriteria**), and
- processor (**ProcessorCriteria**)

all map directly to criteria valuetypes. In the case of vote, engage, and collaboration the elements map to an instance of **ProcessorCriteria** where the contained model is an instance of **VoteModel**, **EngagementModel**, and **CollaborationModel** respectively.

```
<!ENTITY % criteria "
(generic|community|agency|encounter|processor|external|vote|engagement|collaboration)">
<!ELEMENT DPML (%criteria;)>
```

Control

The control **ENTITY** is a declaration that defines an identifying name and description attribute. These attribute declarations correspond to the state fields of the base type **Control** from the **CommunityFramework**.

IDL:omg.org/CommunityFramework::Control:2.0.

```
<!ENTITY % label "label ID #IMPLIED">
<!ENTITY % note "note CDATA #IMPLIED">
<!ENTITY % control "%label; %note;">
```

Input and Output

The input and output elements define consumption and production statements that can be associated to process centric criteria. Both input and output are derived from the abstract **UsageDescriptor** exposed by a **ProcessorModel** usage state field. The value contained by the type field shall be consistent with the XMI Production Rules, specifically, types shall be declared in accordance with their IDL interface repository identifier.

For example, a **GenericResource** would be identified by the string **IDL:omg.org/CommunityFramework:GenericResource:2.0**.

The value of the tag field corresponds to the tag attributed to a usage link (refer to *Production and Consumption* in the *Task and Session Specification*). The implied attribute states that a usage link of the tag is required as distinct from optional. The implied attribute, if true, states that if the tagged link already exists on the controlling **Task**, that link is implied; whereas, a false value states that the link must be explicitly set (possibly resulting in the replacement of an existing link with the same tag value).

IDL:omg.org/CommunityFramework::InputDescriptor:2.0.
IDL:omg.org/CommunityFramework::OutputDescriptor:2.0.

```
<!ENTITY % tag "tag CDATA #REQUIRED">
<!ENTITY % required "required (TRUE|FALSE) 'TRUE'">
<!ENTITY % implied "implied (TRUE|FALSE) 'TRUE'">
<!ENTITY % type "type CDATA #REQUIRED">

<!ELEMENT input EMPTY >
<!ATTLIST input
    %tag; %required; %implied; %type;
>

<!ELEMENT output EMPTY >
<!ATTLIST output
    %tag; %type;
>
```

remove, copy, move and create

The copy, move, create, and remove directives are instructions that can be declared within the scope of a referral, a trigger, or an on post-condition statement. These directives declare actions to be taken by an implementation of **CollaborationProcessor** that effect tagged usage relationships on the coordinating **Task** or **Encounter**. Usage directives enable the declaration of operators that result in the manipulation of usage associations such as renaming or duplication of an association, inversion of an association from consumption to production, or retraction of an association.

IDL:omg.org/CollaborationFramework::Remove:2.0 // remove
IDL:omg.org/CollaborationFramework::Duplicate:2.0 // copy
IDL:omg.org/CollaborationFramework::Move:2.0 // move
IDL:omg.org/CollaborationFramework::Constructor:2.0 // create

```
<!ENTITY % source "source CDATA #REQUIRED">
<!ENTITY % target "target CDATA #REQUIRED">
<!ENTITY % switch "switch (TRUE|FALSE) 'FALSE'">
<!ENTITY % directive.attributes "%source; %target; %switch;">

<!ELEMENT copy EMPTY>
<!ATTLIST copy
    %directive.attributes;
>
```

```
<!ELEMENT move EMPTY>
<!ATTLIST move
    %directive.attributes;
>

<!ELEMENT create (target,%criteria;) >
<!ATTLIST create
    %target;
>

<!ELEMENT remove EMPTY >
<!ATTLIST remove
    %source;
>
<!ENTITY % directive.content "((create|copy|move|remove)*)" >
```

initialization

An initialization **ELEMENT** is a type of transitional action. It qualifies the containing state as a candidate for establishment of the active-state when starting a processor. A processor may be initialized through the apply operation on the abstract **Collaboration** interface, or implicitly through starting a **CollaborationProcessor**.

IDL:omg.org/CollaborationFramework::Initialization:2.0

```
<!ELEMENT initialization (input*) >
<!ATTLIST transition
    %control;
>
```

transition

A transition **ELEMENT** declares a target state facilitating modification of a **CollaborationProcessor** active state path. Modification of the active state path establishes a new collaborative context, enabling a new set of triggers, guard conditions, and timeouts based on declared clocks. A transition element may also contain any number of input statements enabling declaration of required or optional arguments to be supplied under the **Collaboration apply_arguments** operation.

IDL:omg.org/CollaborationFramework::SimpleTransition:2.0

```
<!ELEMENT transition (input*) >
<!ATTLIST transition
    %control;
    target IDREF #IMPLIED
>
```

local

The local **ELEMENT** defines a transition to the current active-state and exposes a clock timeout reset policy. If the reset policy is true, all timeout conditions established under the active state path shall be re-initialized. A local transition element may also contain any number of input statements enabling declaration of required or optional arguments to be supplied under the **Collaboration apply_arguments** operation.

IDL:omg.org/CollaborationFramework::LocalTransition:2.0

```
<!ELEMENT local (input*) >
<!ATTLIST local
    %control;
    reset (TRUE|FALSE) "FALSE"
>
```

termination

A termination declares a processors termination within completion status. The **ENTITY** completion declares a completion class and code. It is used within a termination element to declare a SUCCESS or FAILURE result status and implementation specific result code. The termination element can contain any number of output declarations.

IDL:omg.org/CollaborationFramework::Completion:2.0
IDL:omg.org/CollaborationFramework::TerminalTransition:2.0

```
<!ENTITY % class "class (SUCCESS|FAILURE) 'SUCCESS'">
<!ENTITY % code "code CDATA #IMPLIED">
<!ENTITY % completion "%class; %code;">
<!ELEMENT termination (output*)  >
<!ATTLIST termination
    %control;
    %completion;
>
```

generic

The generic **ELEMENT** is used to define the valuetype **GenericCriteria**, used as an argument to a **ResourceFactory** to construct resources containing arbitrary content contained within a **CORBA any**. Instances of **GenericResource** provide a convenience container for arbitrary resource association (such as the subject of a negotiation or XML document defining contractual terms).

IDL:omg.org/CommunityFramework::GenericCriteria:2.0

```
<!ELEMENT generic (nvp*) >
<!ATTLIST generic
    %control;
>
```

community

The community **ELEMENT** describes an instance of **CommunityCriteria**. **CommunityCriteria** may be used as an argument to a **ResourceFactory** to construct a new instance of **Community**. **Community** is a type of **Workspace** (refer to the *Task and Session* specification) that supports the abstract **Membership** interface.

IDL:omg.org/CommunityFramework::CommunityCriteria:2.0

```
<!ELEMENT community (membership, (nvp*)) >
<!ATTLIST community
    %control;
>
```

agency

The agency **ELEMENT** represents the **AgencyCriteria** valuetype that may be passed as an argument to a **ResourceFactory** resulting in creation of a new Agency instance. **Agency** is a type of **Community** with inheritance from **LegalEntity**. **Agency** represents a community against which supplementary implementation specific policy can be associated (such as an applicable legal domain).

IDL:omg.org/CommunityFramework::AgencyCriteria:2.0

```
<!ELEMENT agency (membership, (nvp*)) >
<!ATTLIST agency
    %control;
>
```

encounter

The encounter **ELEMENT** defines an **EncounterCriteria** against which new instances of **Encounter** can be created using a **ResourceFactory**. **Encounter** is a type of Task that serves as a controller of **Processor** instances. **Encounter**, as a **Membership**, may be associated to many users. Through inheritance of Task exactly one User is associated as the owner of an **Encounter**.

IDL:omg.org/CollaborationFramework::EncounterCriteria:2.0

```
<!ELEMENT encounter (membership, nvp*) >
<!ATTLIST encounter
    %control;
>
```

external

External describes the **ExternalCriteria** valuetype. **ExternalCriteria** contains a public and system identifier of a remote resource. The public and system identifiers contained within an external declaration are factory dependent. For example, a factory implementation with knowledge of DPML can use external criteria as a means through

which criteria can be inferred. Other examples of external criteria application include embedding of interoperable naming URLs. An external element may include any number of input and output statements.

IDL:omg.org/CommunityFramework::ExternalCriteria:2.0

```
<!ELEMENT external ((input|output)*,nvp*)>
<!ATTLIST external
    %control;
    public CDATA #IMPLIED
    system CDATA #REQUIRED
>
```

processor

The processor element contains input and output declarations and a named value pair sequence defining factory criteria. Input and output declarations define the resources that a processor implementation requires as input, and the resources that will be produced by the processor. Supplementary processor criteria is contained under the **nvp** (named value pair) sequence. An implementation is responsible for mapping of **nvp** values to a named value pair sequence as defined by the **CosLifeCycle** Criteria type specification.

IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0

```
<!ELEMENT processor ( (input|output)*, nvp*)>
<!ATTLIST processor
    %control;
>
```

vote

The vote element defines **ProcessorCriteria** containing a **VoteModel** (referred to as vote criteria). Vote criteria, when passed to a **ResourceFactory**, results in the establishment of a new instance of **VoteProcessor**. Using a **VoteProcessor**, members of a coordinating **Encounter** can register votes in support of, in opposition to, or abstain relative to a subject. **VoteProcessor** raises a result status indicating the successful or failure status of a voting process.

IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0
IDL:omg.org/CollaborationFramework::VoteModel:2.0 // model

```
<!ENTITY % numerator "numerator CDATA #REQUIRED" >
<!ENTITY % denominator "denominator CDATA #REQUIRED" >
<!ENTITY % quorum "%numerator; %denominator;" >
<!ELEMENT vote ((input|output)*, nvp*)>
<!ATTLIST vote
    %control;
    %quorum;
```

```
    policy (AFFERMATIVE|NON_ABSTAINING) "AFFERMATIVE"
    single (TRUE|FALSE) "TRUE"
    lifetime CDATA #IMPLIED
>
```

engagement

Engagement defines a **ProcessorCriteria** that contains an **EngagementModel**. When passed as an argument to a **ResourceFactory**, such a criteria will result in the creation of a new instance of **EngagementProcessor**. **EngagementProcessor** declares policy enabling the attribution of proofs and evidence in the establishment of binding agreements.

IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0

```
<!ELEMENT engagement ((input|output)*)>
<!ATTLIST engagement
    %control;
    policy CDATA #IMPLIED
>
```

collaboration

The collaboration element defines **ProcessorCriteria** criteria containing a **CollaborationModel** (referred to as Collaboration Criteria). Collaboration criteria, when passed as an argument to a **ResourceFactory** results in the creation of a new instance of **CollaborationProcessor**. **CollaborationProcessor** is a type of **Processor** that contains a **CollaborationModel** as the definition of the rules of engagement between a set of members associated under a controlling **Encounter**.

IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0
IDL:omg.org/CollaborationFramework::CollaborationModel:2.0 // model

```
<!ELEMENT collaboration ((input|output)*, role?, state, nvp*) >
<!ATTLIST collaboration
    %control;
>
```

launch

The launch element defines a **Launch** valuetype, itself a type of **Guard** that is contained by a **Trigger**. **Guards** establish preconditions to the activation of actions contained within triggers. In the case of **Launch**, the preconditions concern the implicit role of a user and optionally explicit association of a user under a particular role. Implicit preconditions declare three enumeration values:

- **INITIATOR**, the principal that invoked that last collaborative action, or in the case of no prior action, a member of the controlling **Encounter**;
- **RESPONDENT**, any principal other than the initiator; and
- **PARTICIPANT**, any principal associated to the controlling **Encounter**.

These implicit roles are dynamically maintained by an implementation of **CollaborationProcessor**. Implicit roles can be further qualified by declaration of a role name that a principal must be associated to under the coordinating **Encounter** (such as “customer,” “supplier,” etc.).

IDL:omg.org/CollaborationFramework::Launch:2.0
IDL:omg.org/CollaborationFramework::TriggerMode:2.0

```
<!ENTITY % mode "mode (INITIATOR|RESPONDENT|PARTICIPANT) 'PARTICIPANT'">
    <!ELEMENT launch EMPTY >
    <!ATTLIST launch
        %mode;
        role IDREF #IMPLIED
    >
```

clock

A clock defines a **Clock** valuetype. **Clock** contains a timeout declaration. When the containing state enters the Active-state path the clock countdown is enabled. Clock resetting is possible through invocation of a local transition. **Clock** disabling is possible by changing the active state path such that the containing state is no longer active. On timeout of a clock, an implementation of **CollaborationProcessor** is responsible for invoking the action contained by the **Trigger** containing the clock declaration. A typical application of the clock operator is to automatically trip a state transition after a predetermined period of in-activity.

IDL:omg.org/CollaborationFramework::Clock:2.0

```
<!ELEMENT clock EMPTY >
<!ATTLIST clock
    timeout CDATA #IMPLIED
>
```

referral

A **referral** references the ID of an action to apply. An implementation of **Collaboration** is responsible for management of the branching of the collaboration state to the identified action and in the case of an action defined as a compound transition, to execute on statements arising from sub-process conclusion.

IDL:omg.org/CollaborationFramework::Referral:2.0

```
<!ELEMENT referral %directive.content; >
<!ATTLIST referral
    action IDREF #REQUIRED
>
```

compound

A compound transition is not directly represented in the DPML scheme as an element. Instead, it is represented in terms of an **ENTITY** content rule associating a processor criteria (or element expandable to a processor criteria) and result mapping. While simplifying DPML structure, the flattening of criteria and action results in the requirement for a compound action label to be equivalent to the model contained by a compound action.

IDL:omg.org/CollaborationFramework::CompoundTransition:2.0

```
<!ENTITY % compound "((external|process|collaboration|vote|engagement), (on+))">
```

trigger

A trigger contains a guard, directive operators, an action, and a priority attribute. **Triggers** are referenced by their label under the **Collaboration** interface apply operation. An implementation of **Collaboration** takes trigger labels as execution instructions that enable clients to manipulate collaborative context. An implementation of apply is responsible for assessing guard preconditions, following which apply requests and associated usage directives are queued relative to **Trigger** priorities. On execution and implementation is responsible for executing usage directives before executing the action contained within the trigger.

IDL:omg.org/CollaborationFramework::Guard:2.0
IDL:omg.org/CollaborationFramework::Trigger:2.0

```
<!ENTITY % guard "(launch*, clock*)">
<!ENTITY % priority "priority CDATA #IMPLIED">
<!ENTITY % transitional "(initialization|transition|local|termination)">
<!ENTITY % action "(%transitional;|referral|%compound;)">

<!ELEMENT trigger (%guard;,%directive.content;,%action;)>
<!ATTLIST trigger
    %control;
    %priority;
>
```

on

A compound transition content declaration associates processor criteria that may be executed as a sub-process with a set of on statements. Each on statement declares an action to apply given a particular result of the process executed as a result of criteria expansion. On statements are defined by class and result code. An implementation of collaboration is responsible for matching sub-process result class and sub-codes and subsequent firing of the declared action.

IDL:omg.org/CollaborationFramework::Map:2.0

```
<!ELEMENT on (%directive.content;,%action;) >
<!ATTLIST on
```

```
    %class;
    %code;
>
```

state

A “state” is an element containing a set of sub-states and associated triggers. State elements are the basic building blocks for collaborative context. Each state element can contain sub-states and each state element can contain any number of Trigger declarations. A **Collaboration** implementation maintains the notion of active-state following initialization of the collaboration and tracks active-state relative to the last transition that has been invoked. The active state path is the set of states between the active state and the root-state of the **CollaborationModel**. All triggers declared within the active-state path are considered candidates relative to the apply operation. By modifying the active state (and by consequence the active-state path) the collaborative content and available trigger options available to the associated membership are modified relative to the constraints and directives declared under exposed triggers.

IDL:omg.org/CollaborationFramework::State:2.0

```
<!ELEMENT state ((trigger|state)*)>
<!ATTLIST state
    %control;
>
```

membership

Membership is a model of the policy and roles that establishes the notion of a group of users sharing the same set of rules. This element is used within the structural definition of criteria such as community, agency, and encounter.

IDL:omg.org/CommunityFramework::MembershipModel:2.0

```
<!ELEMENT membership (membership.policy?, role) >
```

membership.policy

The **membership.policy ELEMENT** declares privacy and exclusivity constraints on the membership. The **membership.policy** element is contained within the **membership** element. **MembershipPolicy** declares an exclusivity attribute that if true, ensures that all members of a membership are uniquely represented in terms of identifiable principals; that is, no principal may be represented more than once. The privacy attribute qualifies the level of information that may be disclosed about the business roles attributed to a given member via operation of the **Membership** abstract interface.

IDL:omg.org/CommunityFramework::MembershipPolicy:2.0

```
<!ELEMENT membership.policy EMPTY>
<!ATTLIST membership.policy
    privacy (PUBLIC|RESTRICTED|PRIVATE) "PUBLIC"
```

```
    exclusivity (TRUE|FALSE) "TRUE"
>
```

role

Role is a specification of the state of a business role that may be abstract or concrete depending on the value of the abstract attribute. A role element exposes a quorum and ceiling through the contained **role.policy** element. Business roles such as “supplier” or “customer” can be packaged under higher-level roles such as “signatory.” Association of the status of “signatory” to both supplier and customer can be achieved by locating supplier and customer as sub-roles of a parent role named “signatory.” Roles can then be used as conditional guards concerning access to triggers within the body of collaboration models.

IDL:omg.org/CommunityFramework::Role:2.0

```
<!ELEMENT role (role.policy?,role*) >
<!ATTLIST role
    %control;

    abstract (TRUE|FALSE) "FALSE"
>
```

role.policy

Role policy is an element that defies the state of a **RolePolicy** valuetype**. RolePolicy** is used as a container of the policy attributed to a specific name business role that includes ceiling and quorum values, policy concerning quorum assessment, and policy concerning the connection status of a user relative to quorum calculations.

IDL:omg.org/CommunityFramework::RolePolicy:2.0

```
<!ELEMENT role.policy EMPTY >
<!ATTLIST role.policy
    ceiling CDATA #IMPLIED
    quorum CDATA #IMPLIED
    assessment (STRICT|LAZY) "LAZY"
    policy (SIMPLE|CONNECTED) "SIMPLE"
>
```

nvp

Named value pairs are used as descriptive arguments to generic resource criteria. A sequence of nvp elements can be mapped to a **CosLifeCycle::Criteria** type as exposed by the **Criteria** type.

IDL:omg.org/CosLifeCycle::NameValuePair:1.0

While interpretation of **nvp** values is implementation dependent, the following rules shall apply to values expressing IDL types:

1. Basic IDL types are represented by a string containing the name of the type. The type is derived from the CORBA TypeCode's **TCKind** by deleting the leading "tk_". This rule follows the convention used in section 5.3.10.2 (CorbaTypeName) of the XMI 1.0 specification (formal/00-06-01).

   Example: the string representation of the type **long** is "long;" that of **unsigned long long** is "ulonglong."

2. Sequences of basic IDL types are represented by a string containing the type-specifier in IDL syntax without any spaces. That is, a sequence of **XXXs** is coded as "sequence<XXX>" where XXX is the name of the string found using rule 1.

   Example: A sequence of **longs** is represented by ""sequence<long>."

3. For other data types, the repository ID is used.

   Example: the **CollaborationProcessor** is represented by "**IDL:omg.org/CollaborationFramework/CollaborationProcessor:2.0**."

```
<!ELEMENT nvp (ANY) >
<!ATTLIST nvp
    name CDATA #REQUIRED
>
```

1.4 Element to IDL Type Mapping

Element	IDL Type
input	**IDL:omg.org/CommunityFramework::InputDescriptor:2.0**
output	**IDL:omg.org/CommunityFramework::OutputDescriptor:2.0**
copy	**IDL:omg.org/CollaborationFramework::Duplicate:2.0**
move	**IDL:omg.org/CollaborationFramework::Move:2.0**
create	**IDL:omg.org/CollaborationFramework::Constructor:2.0**
remove	**IDL:omg.org/CollaborationFramework::Remove:2.0**
initialization	**IDL:omg.org/CollaborationFramework::Initialization:2.0**
transition	**IDL:omg.org/CollaborationFramework::SimpleTransition:2.0**
local	**IDL:omg.org/CollaborationFramework::LocalTransition:2.0**
termination	**IDL:omg.org/CollaborationFramework::TerminalTransition:2.0**
generic	**IDL:omg.org/CommunityFramework::GenericCriteria:2.0**
community	**IDL:omg.org/CommunityFramework::CommunityCriteria:2.0**
agency	**IDL:omg.org/CommunityFramework::AgencyCriteria:2.0**
encounter	**IDL:omg.org/CollaborationFramework::EncounterCriteria:2.0**
external	**IDL:omg.org/CommunityFramework::ExternalCriteria:2.0**
processor	**IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0** **IDL:omg.org/CollaborationFramework::ProcessorModel:2.0**

vote	**IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0** **IDL:omg.org/CollaborationFramework::VoteModel:2.0**
engagement	**IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0** **IDL:omg.org/CollaborationFramework::EngagementModel:2.0**
collaboration	**IDL:omg.org/CollaborationFramework::ProcessorCriteria:2.0** **IDL:omg.org/CollaborationFramework::CollaborationModel:2.0**
launch	**IDL:omg.org/CollaborationFramework::Launch:2.0**
clock	**IDL:omg.org/CollaborationFramework::Clock:2.0**
referral	**IDL:omg.org/CollaborationFramework::Referral:2.0**
compound	**IDL:omg.org/CollaborationFramework::Compound:2.0**
trigger	**IDL:omg.org/CollaborationFramework::Trigger:2.0**
on	**IDL:omg.org/CollaborationFramework::Map:2.0**
state	**IDL:omg.org/CollaborationFramework::State:2.0**
membership	**IDL:omg.org/CommunityFramework::MembershipModel:2.0**
membership.policy	**IDL:omg.org/CommunityFramework::MembershipPolicy:2.0**
role	**IDL:omg.org/CommunityFramework::Role:2.0**
role.policy	**IDL:omg.org/CommunityFramework::RolePolicy:2.0.**
nvp	**IDL:omg.org/CosLifeCycle::NameValuePair:1.0**

1.5 Related DPML Documents

Additional information concerning DPML development and additional DPML documents are maintained under the following URL:

http://home.osm.net/dpml

The latest version of DPML can be located under the following URL:

http://home.osm.net/dpml/dpml.dtd

Collaboration Framework 2

Contents

This chapter contains the following sections.

2.1 Introduction

The **CollaborationFramework** defines a sharable **Task** named **Encounter**, formalizes the definition of a **Processor**, and introduces three types of **Processor**s dealing with the application level requirements covering contractual engagement, voting, and collaboration against which business processes supporting contract negotiation, fulfillment, and settlement can be defined, simulated, and executed.

Principal interfaces defined under this specification include:

- **EngagementProcessor**, a processor supporting the registration of **Evidence** and generation of **Proof** by a membership.

- **VoteProcessor**, a processor supporting the registration of votes by a membership.
- **CollaborationProcessor**, a processor supporting collaborative interaction between members of an **Encounter**.

These interfaces build upon the specifications established under the **CommunityFramework**, in particular the notion of **Membership** is reused as the basis for the definition of a shared Task associated to a common **Processor**. The **CollaborationFramework** continues the Model/Simulator pattern established under the **CommunityFramework** specifications as the mechanisms for separation of configuration and execution policy from the IDL computational interface.

Table 2-1 Core Interfaces - Summary Table

Interface	Description
Processor	Processor is a base type for interfaces dealing with contractual engagement voting and collaboration. Processor is associated to a Task and can expose a sub-processor hierarchy.
ProcessorModel	A valuetype derived supporting the abstract Model interface used to describe preconditions to Processor execution.
UsageDescriptor	An abstract valuetype inherited by valuetypes contained by a ProcessorModel that declares a usage (input, output) constraint.
InputDescriptor	Declaration of an input resource (consumed) that a processor requires on its associated task.
OutputDescriptor	Declaration of an output (produced) resource that a processor generates on its associated task.
ProcessorCriteria	A type of Criteria used by a ResourceFactory to construct a new Processor instance based on the contained ProcessorModel.
Encounter	An Encounter is a type of Task that incorporates the abstract Membership interface.
EncounterModel	A valuetype extending the abstract CommunityFramework MembershipModel that contains the policy and role model of a membership.
EncounterCriteria	A type of Criteria used by a ResourceFactory to construct a new Encounter instance.

Table 2-2 Application Interfaces - Summary Table

Interface	Description
Engagement	Abstract definition of engagement.
EngagementProcessor	A type of Processor supporting the association of Proof and Evidence by a set of collaborating users based on the abstract Engagement interface.
EngagementModel	A valuetype containing implementation dependent policy of an Engagement processor.

Table 2-2 Application Interfaces - Summary Table

Vote	Abstract interface defining vote registration and vote aggregation operations.
VoteProcessor	A type of processor supporting the registration of votes by members of an associated Encounter based on the Vote abstract interface.
VoteModel	A valuetype containing the ceiling, count, and multiple registration policy applicable to a VoteProcessor.
Collaboration	An abstract interface defining operations through which a client can interact with a collaborative state model.
CollaborationProcessor	A type of Processor supporting collaborative interaction relative to a CollaborationModel rule base using the abstract Collaboration interface.
CollaborationModel	A valuetype defining state, sub-states, transitions, compound actions, and role related policies.

Table 2-3 CollaborationModel Related Valuetypes - Summary Table

Interface	**Description**
State	A valuetype defining a state hierarchy against which Triggers and sub-States can be associated.
Trigger	A container of an invocation guard, preconditions, and an action.
Action	Base valuetype for Transition, CompoundTransition, and Referral.
Transition	A type of action that is a base type to all actions related to modification of a collaborative state context. A transition may declare changes to rules concerning inputs of a processor.
Transitional	Abstract valuetype contained by a Transition. This is the base type to Initialization, SimpleTransition, LocalTransition, and TerminalTransition.
Initialization	A transitional valuetype used to declare a candidate initial state.
SimpleTransition	A type of Transitional supporting the modification of the active state of a collaboration.
LocalTransition	A type of Transitional supporting loop-back transition functionality.
TerminalTransition	A type of Transitional that defines a processor result value.
CompoundTransition	A type of Action that declares a transition that is executed as a sub-process associated with an independent processor model. A compound transition may have multiple possible result states.
Referral	A type of Action used to redirect a result to a locally defined action.
Map	A valuetype contained by a CompoundTransition. Used to associate compound transition sub-process results to explicit actions.

2.2 Processor and Related Interfaces

The *Task and Session* specification (formal/00-05-03) defines the notion of a processor as the source of execution relative to a Task. The **CollaborationFramework** establishes a formal definition of **Processor** as abstract base type for interfaces dealing with collaboration, engagement, and voting.

- Section 2.2.1, “Processor,” on page 2-5 presents the definition of the **Processor** interface that serves as a base type to **CollaborationProcessor**, **VoteProcessor**, and **EngagementProcessor**.
- Section 2.2.2, “Master, Slave, and the Control Link,” on page 2-7 defines the **Master** and **Slave** abstract interfaces and their relationship to the **Control** link through which one processor can be associated as a sub-processor to another.
- Section 2.2.3, “StateDescriptor,” on page 2-9 presents **StateDescriptor**, a valuetype exposed by an instance of **Processor** that contains information about a processor execution state including declaration of problems arising during configuration and execution.
- Section 2.2.4, “ProcessorModel and Related Constraint Declarations,” on page 2-10 details the **ProcessorModel** valuetype used to declare configuration preconditions and the **ProcessorCriteria** valuetype used by a **ResourceFactory** in the creation of new processor instances.
- Section 2.2.5, “Coordination Link Family,” on page 2-13 defines a set of abstract and concrete link types used to describe the coordination relationship between a Task and a Processor.

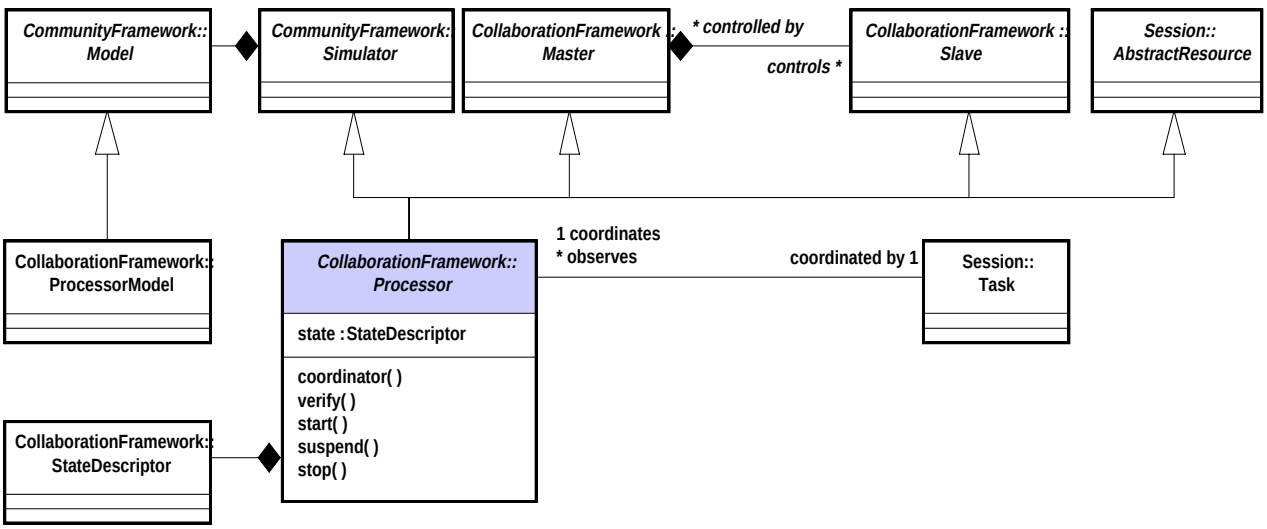


Figure 2-1 Processor Object Model

2.2.1 Processor

A processor is responsible for applying input arguments (associated consumed and produced resource selection) declared by a coordinating **Task** in the execution of a service. Operations exposed by **Processor** are largely defined by the implied semantics documented under the *Task and Session* specification (formal/00-05-03). A processor is responsible for notification of state change towards its associated **Task** and handling start, suspend, and stop requests in accordance with the Task Session state model. **Processor** inherits from **AbstractResource** (consistent with the *Task and Session* specification of a processor).

As a **Simulator**, a **Processor** exposes a valuetype that supports the **Model** interface. A **Processor** specialization is required to return an instance of **ProcessorModel** under the **model** operation from the inherited abstract **Simulator** interface. Through inheritance of both **Slave** and **Master** abstract interfaces, a **Processor** can expose subsidiary and parent processors associated through **Coordination** links to a single managing **Task**. As such, a **Task** can be viewed as the coordinator of the processor hierarchy.

2.2.1.1 IDL Specification

```
interface Processor :
    Session::AbstractResource,
    CommunityFramework::Simulator,
    Master, Slave
    {

    readonly attribute StateDescriptor state;

    Session::Task coordinator(
    ) raises (
        Session::ResourceUnavailable
    );

    CommunityFramework::Problems verify( );

    void start (
    ) raises (
        Session::CannotStart,
        Session::AlreadyRunning
    );

    void suspend (
    ) raises (
        Session::CannotSuspend,
        Session::CurrentlySuspended
    );

    void stop (
    ) raises (
```

```
                Session::CannotStop,
                Session::NotRunning
        );
};
```

Table 2-4 Processor Attribute Table

Name	Type	Properties	Purpose
state	StateDescriptor	readonly	Declaration of the state of a Processor – see Section 2.2.3, “StateDescriptor,” on page 2-9.

Table 2-5 Processor Operation Table

Name	Returns	Description
coordinator	Task	The coordinator operation returns the Task acting as coordinator of the processor. If no task is associated to the processor, the operation raises the **ResourceUnavailable** exception.
verify	Problems	Operations returns a sequence of Problem instances concerning configuration of a processor relative to the constraints defined under the associated ProcessorModel.
start	void	Moves a processor into the running state. Semantically equivalent to the Task start operation (refer to the *Task and Session* specification). If the start operation raises the **CannotStart** exception, a client can access supplementary information under the StateDescriptor instance returned from the processor state attribute.
suspend	void	Moves a processor into a suspended state. Semantically equivalent to the Task suspend operation (refer to the *Task and Session* specification).
stop	void	Stops a processor. Semantically equivalent to the Task stop operation (refer to the *Task and Session* specification).

Table 2-6 Processor Structured Event Table

Event	Description		
state	Notification of the change of state of a Processor.		
	Supplementary properties:		
	value	StateDescriptor	Description of the current state and any associated problems.

2.2.1.2 Processor creation and Task association

The following sequence concerning **Processor** instantiation is strongly influenced by the *Task and Session Specification* and factory operation pattern defined under the **CommunityFramework** module.

1. Client creates a new concrete instance of **Processor** by passing a **Criteria** valuetype as an argument to a **ResourceFactory create** operation.
2. **Client** creates a new **Task**, passing the created processor as an argument to the **create_task** operation on User (refer to the *Task and Session* specification, User and Task).
   - **Task** implementation binds to processor using a **Coordinates** link referencing itself under the **resource** state field.
   - **Processor** establishes internal reference to coordinating **Task** using the supplied link by creating and maintaining a **CoordinatedBy** link that references the coordinating **Task**.
3. **Task** establishes initial state from **Processor** using the **state** attribute.
4. **Client** is responsible for ensuring that any usage preconditions to processor execution are resolved using the **verify** operation.
5. **Client** invokes the **start** operation on Task that in turn invokes **start** on the controlled processor.

2.2.1.3 Verification of processor configuration

The **Processor verify** operation returns a sequence of **Problem** instances related to configuration of a processor relative to the constraints defined under the associated **ProcessorModel**. This operation is provided so that a client can validate proper and complete configuration of a processor prior to execution. For example, a **ProcessorModel** may declare input and output resource associations that must be established by a controlling task before invocation of the **start** operation. The **verify** operation enables verification of a **Processor** configuration and readiness to start.

Problems verify( );

2.2.2 Master, Slave, and the Control Link

The abstract interfaces **Master** and **Slave** are used in conjunction with an abstract valuetype named **Management** that defines the base type for the concrete links **Controls** and **ControlledBy**. **Controls** is a link held by an implementation of **Master** that references zero to many **Slave** instances. **ControlledBy** is a link held by a **Slave** implementation that references zero to one **Master** instances. The relationship from master to slave is one of strong aggregation – removal of the **Master** implies removal of all **Slaves**. Using the control relationship, it is possible for a **Processor** to expose a sub-process hierarchy that can be navigated by a client. Both **Master** and **Slave** define convenience operations concerning access to the respective sub-processors and parent processor. **Master** interface defines the **slaves** operation that returns an iterator and a

sequence of **Slave** sub-processors. The maximum length of the **Slaves** sequence is controlled by the input argument **max_number**. The **Slave** interface defines the readonly attribute **master** that returns a reference to the controlling **Master**. In the event of a top-level processor, the master attribute will return a null object reference.

2.2.2.1 IDL Specification

```
abstract interface Master {
    SlaveIterator slaves (
        in long max_number,
        out Slaves slaves
    );
};

abstract interface Slave {
    readonly attribute CollaborationFramework::Master master;
};

abstract valuetype Management : Session::Link{ };

valuetype Controls : Management {
    public Slave resource;
};

valuetype ControlledBy : Management {
    public Master resource;
};
```

Table 2-7 Controls Link State Table

Name	Type	Properties	Purpose
resource	Slave	public	A reference to an AbstractResource implementing the Slave interface. An implementation of Master may hold 0..* Controls link instances, representing the strong aggregation relationship from a Master to subsidiary Slaves.

Table 2-8 ControlledBy Link State Table

Name	Type	Properties	Purpose
resource	Master	public	A reference to an AbstractResource implementing the Master interface. An implementation of Master may hold 0..1 ControlledBy link instances representing the parent processor.

2.2.3 *StateDescriptor*

Processor state is accessible through the **state** attribute. The **state** attribute returns an instance of **StateDescriptor**, a valuetype containing an enumeration value of the process state equivalent to the state model defined under the *Task and Session* specification. **StateDescriptor** also contains a state field named **problems** that exposes any standing problems concerning processor configuration or execution.

Completion is a valuetype contained within **StateDescriptor**. When a processor completes (signalled by the establishment of the closed processor state), the completion field contains a **Completion** instance that qualifies the closed state as either a logical business level success or failure. For example, a processor supporting vote aggregation can declare a distinction between a successful and unsuccessful result towards a client. In this example, failure could arise as a result of an insufficient number of affirmative votes, or through failure of the group to establish quorum. In both cases, the failure is a business level failure and should not be confused with technical or transaction failure. An implementation dependent identifier may be attributed to a **Completion** instance to further classify a success or fail result. Prior to a processor reaching a closed state the completion field shall return a null value.

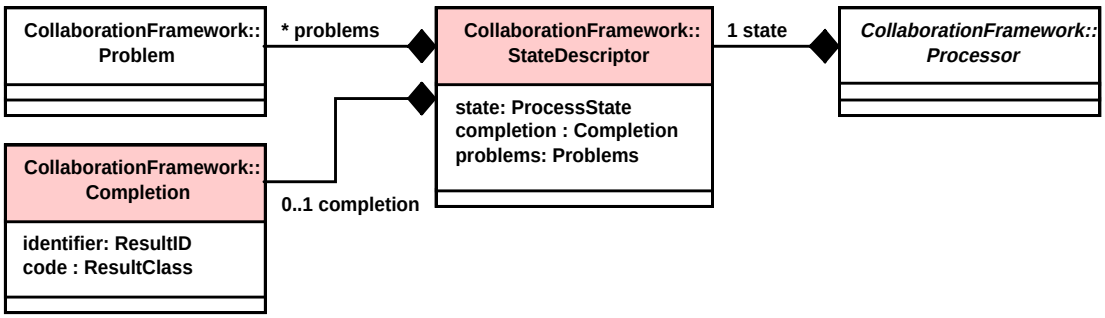


Figure 2-2 StateDescriptor Object Model

2.2.3.1 *IDL Specification*

```
valuetype ResultID unsigned long ;
valuetype ResultClass boolean;

valuetype Completion
    {
    public ResultClass result;
    public ResultID code;
};

valuetype ProcessorState Session::task_state;

valuetype StateDescriptor
    {
```

```
    public ProcessorState state;
    public CollaborationFramework::Completion completion;
    public CommunityFramework::Problems problems;
};
```

Table 2-9 StateDescriptor State Table

Name	Type	Properties	Purpose
state	ProcessorState	public	An enumeration of process state values open, not_running, notstarted, running, suspended, terminated, completed, and closed (refer to the *Task and Session* specification, Task state description).
problems	Problems	public	A sequence of Problem instances (of possibly zero length) attributable to the current execution state of the processor.
completion	Completion	public	Declaration of a success or fail completion condition together with a numeric application defined result identifier.

Table 2-10 Completion State Table

Name	Type	Properties	Purpose
code	ResultID	public	An implementation specific identifier of a completion state.
result	ResultClass	public	A boolean value indicating a business level notion of success or failure of the process.

2.2.4 ProcessorModel and Related Constraint Declarations

The **ProcessorModel** valuetype defines a set of usage (input and output) towards its controlling **Task**. These declarations are expressed as a set of **UsageDescriptor** instances (equivalent to the declaration of argument parameters). Collectively, the set of **UsageDescriptor** instances declare the naming convention to be applied to tagged **Usage** links held by the co-ordinating **Task**. **Usage** declarations are defined through the valuetypes **InputDescriptor** and **OutputDescriptor**. Both valuetypes contain the declaration of a **tag** name (corresponding to the usage tag string) and a **type** field containing a **TypeCode** value. The **OutputDescriptor** contains an additional **required** field that if true, states that the link must exist or be supplied. If false, the input declaration can be considered as an optional argument.

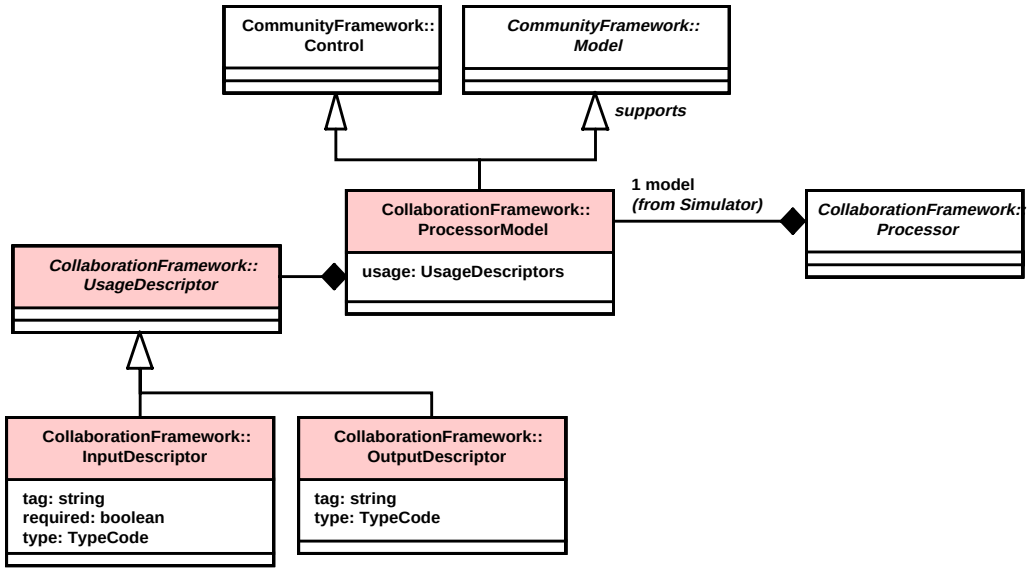


Figure 2-3 Processor Model and Usage Descriptor

Using the control structures it is possible for a processor model to define constraints such as "the processor must be associated to a controlling **Task** with a resource of type **User** associated as a consumed resource declared under the tag "customer" before this processor can be started. Such a requirement can be expressed by the creation of an **InputDescriptor** exposing the following:

- The text string "customer" under the **tag** field.
- The boolean value true under the **required** field (indicating that a **Usage** link tagged as subject must be associated to the controlling **Task** before attempting to start a processor).
- A **UsageSource** instance under the **source** field that declarers a type precondition on the **Usage** link's resource field – in this example, the value would be the **Session::User** type code.

Collectively, these constraints represent the processor signature and facilitate plug-and-play interoperability between process descriptions defined in and executing under different technical and administrative domains.

A new instance of **Processor** may be created by passing an instance of **ProcessorCriteria** to a resource factory (refer to **CommunityFramework::ResourceFactory create** operation).

2.2.4.1 *IDL Specification*

```
valuetype TypeCode CORBA::TypeCode;
```

```
abstract valuetype UsageDescriptor { };

valuetype InputDescriptor :
    UsageDescriptor
    {
    public string tag;
    public boolean required;
    public boolean implied;
    public TypeCode type;
};

valuetype OutputDescriptor :
    UsageDescriptor
    {
    public string tag;
    public TypeCode type;
};

valuetype ProcessorModel :
    CommunityFramework::Control
    supports CommunityFramework::Model
    {
    public UsageDescriptors usage;
};

valuetype ProcessorCriteria :
    CommunityFramework::Criteria
    {
    public ProcessorModel model;
};
```

Table 2-11 ProcessorModel State Table

Name	Type	Properties	Purpose
usage	UsageDescriptors	public	A sequence of valuetypes derived from UsageDescriptor, each defining usage links conditions relative to the associated Task.

Table 2-12 InputDescriptor State Table

Name	Type	Properties	Purpose
tag	string	public	The name to be set as the tag value of Usage link that can be established on the controlling Task.
required	boolean	public	If true, the usage association must exist under the coordinating Task before attempting to start the processor. Default value is true.

Table 2-12 InputDescriptor State Table

implied	boolean	public	A qualifier used under a CollaborationModel. If true, the usage association may be implicitly inferred by an existing link with the same tag name, if false, the link must be explicitly passed as an ApplyArgument (refer Collaboration apply operation) establishing or replacing an existing tag link of the same name. Default value is true.
type	TypeCode	public	Declaration of the type of resource to be bound under a Consumes usage association on the controlling Task.

Table 2-13 OutputDescriptor State Table

Name	Type	Properties	Purpose
tag	string	public	OutputDescriptor declares an association that will be established by a Processor on normal completion. The tag value declares the value of the usage tag value that will be created.
type	TypeCode	public	Declaration of the type of resource that will be created by the processor on the controlling Task.

Table 2-14 ProcessorCriteria State Table

Name	Type	Properties	Purpose
model	ProcessorModel	public	Declaration of processor consumption and production usage constraints. An implementation of ResourceFactory is responsible for assessing the type of Model contained within a ProcessorCriteria to determine the type of Process to create. For example, a ProcessorCriteria containing an instance of CollaborationModel will return a type of CollaborationProcessor.

2.2.5 Coordination Link Family

The **Execution** link defined under the *Task and Session* specification declares an abstract association between an **AbstractResource**, acting as a processor, and a **Task**. The abstract **Execution** relationship is used as the base for definition of an abstract **Coordination** relationship. Coordination serves as the base for the concrete links named **Monitors**, **Coordinates**, and **CoordinatedBy**.

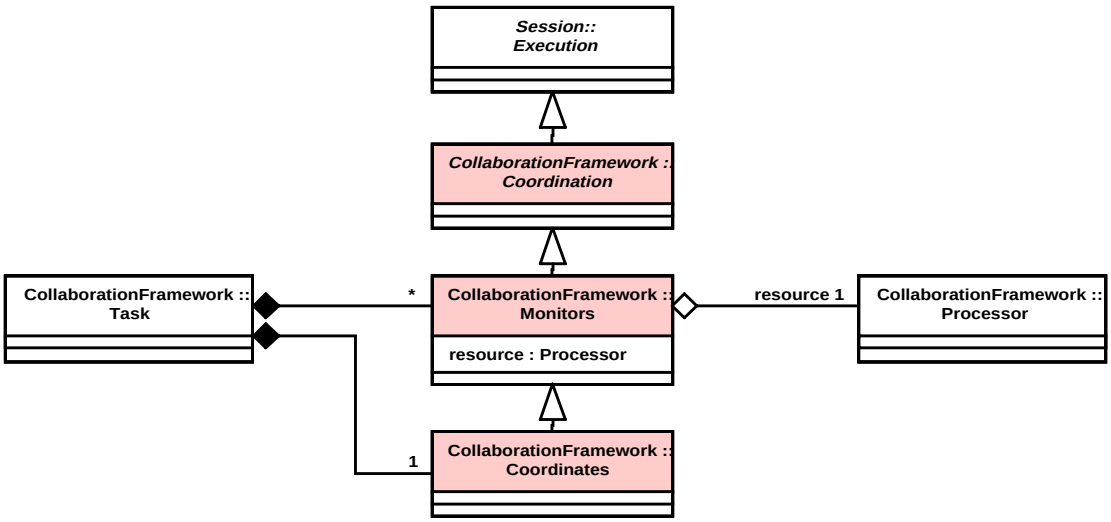


Figure 2-4 The Abstract Coordination and Concrete Monitors and Coordinates Link

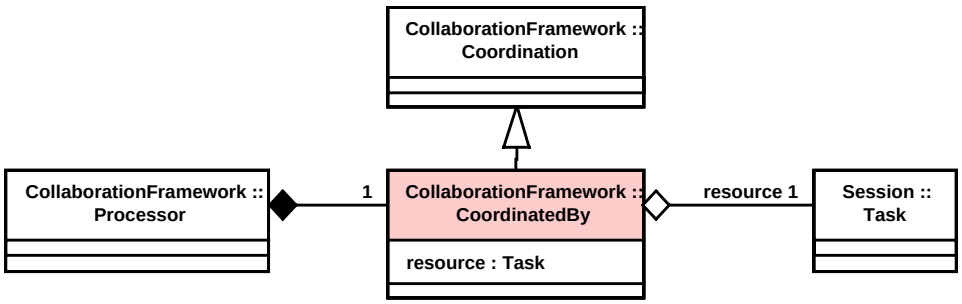


Figure 2-5 Inverse CoordinatedBy Link

2.2.5.1 IDL Specification

```
abstract valuetype Coordination : Session::Execution{ };

valuetype Monitors : Coordination {
    public Processor resource;
};

valuetype Coordinates : Monitors {};

valuetype CoordinatedBy : Coordination {
    public Session::Task resource;
};
```

Table 2-15 Coordination Link Family Cardinality Table

Type holding the link	Link type	Type referenced by Link	Description
Task	Monitors	Processor	An instance of Task monitors 0..* Processors.
Task	Coordinates	Processor	Coordinates is a type of Monitor. An instance of Task coordinates 0..1 Processor instance.
Processor	CoordinatedBy	Task	A Processor is coordinated by 0..1 Task instances.

Table 2-16 Monitors State Table

Name	Type	Properties	Purpose
resource	Processor	public	A reference to a Processor that the Task holding this link monitors.

Table 2-17 CoordinatedBy State Table

Name	Type	Properties	Purpose
resource	Task	public	A reference to a Task that is coordinating the processor that is holding this link instance. The Task is maintaining either a Monitors or Coordinates link towards the Processor holding this link.

2.3 Encounter

The *Task and Session* specification defines a **Task** as a type corresponding to a view of a processor. The specification of a **Task** is focused extensively towards a single user. The **CollaborationFramework** extends this notion through the introduction of a **Task** type called **Encounter** that is owned and managed by a single **User** but associated by reference to other **Users** though **Member** links (refer to the *CommunityFramework* chapter).

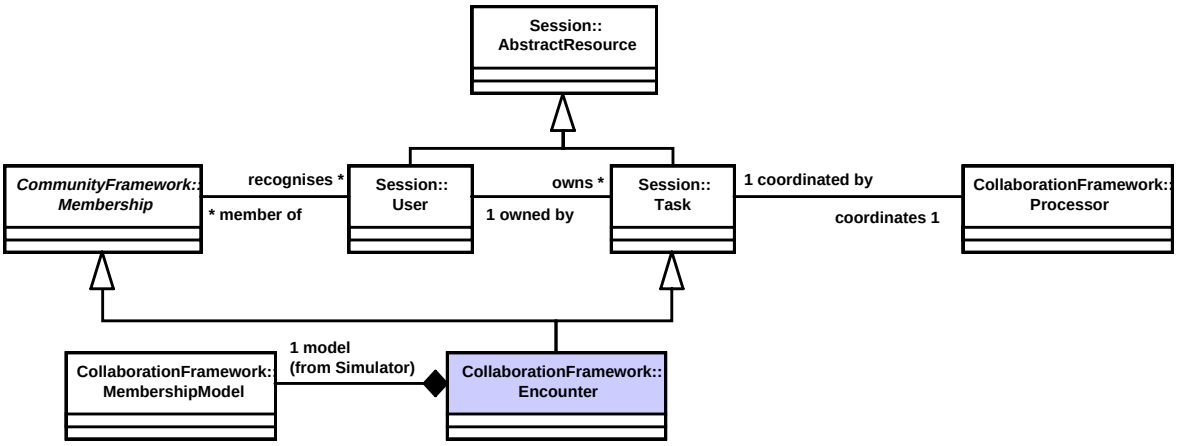


Figure 2-6 Encounter Object Model

In effect an **Encounter** can be considered as a **Task** managed by its owner where the state of the **Task** is available to a closed community of members. This model enables the association of multiple users within a collaborative execution context defined by an associated processor. An **Encounter** is defined as both a **Task** (refer to the *Task and Session* specification) and **Membership** (refer to the *CommunityFramework* chapter). As a **Task** it supports full lifecycle semantics, can be referenced as a resource within a workspace or community, and exposes a relationship to an assigned processor. As a **Membership**, the **Encounter** aggregates a set of members, representing a set of collaborating **Users**. **Encounter**, through inheritance of **Simulator**, is required to return a valuetype supporting the abstract **Model** interface. In the case of **Encounter**, the valuetype returned must be an instance of **MembershipModel** (a valuetype supporting the abstract **Model** interface). Implementations of **Processor** associated to an **Encounter** can interrogate an **Encounter** to establish the roles attributed to members of the **Encounter**. This information can be used by processor implementations to enforce preconditions on role related actions.

2.3.1 Encounter and EncounterCriteria

An **Encounter** is a type of **Task** that incorporates the abstract **Membership** interface. As a **Membership** an **Encounter** is associated to possibly many **Users** through **Member** links. As a **Task** an **Encounter** is associated to exactly one owner, possibly multiple consumed and produced resources, and a single processor. As such, **Encounter** can be considered as a shared view of a collaborative process context under the coordination of a single **User**. New instances of **Encounter** may be created using a **ResourceFactory** by passing an instance of **EncounterCriteria** as the criteria argument.

2.3.1.1 IDL Specification

```
interface Encounter :
    Session::Task,
    CommunityFramework::Membership
    {
};

valuetype EncounterCriteria :
    CommunityFramework::Criteria
    {
    public CommunityFramework::MembershipModel model;
};
```

Table 2-18 EncounterCriteria State Table

Name	Type	Properties	Purpose
model	MembershipModel	public	Declaration of the membership model instance to be associated to the created Encounter.

2.4 VoteProcessor and VoteModel

VoteProcessor is a **Processor** extended to include the abstract **Vote** interface. The **Vote** interface declares an attribute **vcount** through which the last vote count can be accessed, and a single **vote** operation supporting the registration of a vote by a client. **Vote** registration is achieved though supply of one of the enumerated value YES, NO, or ABSTAIN as defined by **VoteDescriptor**. The **vote** operation returns an implementation defined **Proof** to the client. The **vcount** attribute returns a **VoteCount** instance that holds a summation of yes, no, and abstain votes registered at the time of the invocation.

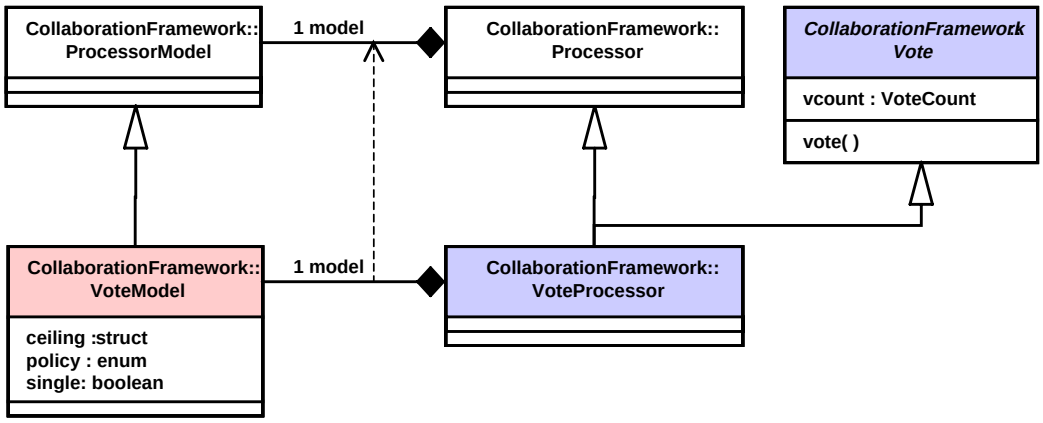


Figure 2-7 VoteProcessor and VoteModel

2.4.1 *Supporting Structures*

Four supporting structures are used in the definition of a voting process. **VoteCount** is a valuetype containing the summation of yes, no, and abstain votes under a voting process at a particular time. **VoteDescriptor** is an enumeration of vote value, YES, NO, and ABSTAIN. **VoteStatement**, a valuetype containing a **VoteDescriptor**, is passed as an input argument to a **VoteProcessor**'s **vote** operation. The **vote** operation returns a **VoteReceipt** to a client following invocation of the **vote** operation. **VoteReceipt** contains a copy of the supplied vote together with a timestamp value corresponding to the date and time of the operation invocation.

2.4.1.1 *IDL Specification*

```
valuetype VoteCount {
    public Session::Timestamp timestamp;
    public long yes;
    public long no;
    public long abstain;
};

enum VoteDescriptor{
    NO,
    YES,
    ABSTAIN
};

abstract valuetype Proof {};
abstract valuetype Evidence {};

valuetype VoteStatement :
```

```
Evidence
{
    public VoteDescriptor vote;
};

valuetype VoteReceipt :
Proof
{
    public Session::Timestamp timestamp;
    public VoteStatement statement;
};
```

Table 2-19 VoteCount State Table

Name	Type	Properties	Purpose
timestamp	Session::Timestamp	public	Timestamp of the last vote registration.
yes	long	public	The summation of YES votes registered under a process.
no	long	public	The summation of NO votes registered under a process.
abstain	long	public	The summation of ABSTAIN votes registered under a process.

Table 2-20 VoteStatement State Table

Name	Type	Properties	Purpose
vote	VoteDescriptor	public	One of the enumerated values YES, NO or ABSTAIN.

Table 2-21 VoteReceipt State Table

Name	Type	Properties	Purpose
timestamp	Session::Timestamp	public	Date and time of registration of the VoteStatement by a VoteProcessor.
statement	VoteStatement	public	Copy of the VoteStatement instance passed into the vote operation.

2.4.2 *VoteProcessor*

A **VoteProcessor** is a type of **Processor** supporting operations defined under the abstract **Vote** interface. **Vote** exposes an attribute named **vcount** that returns a **VoteCount** instance. The **VoteCount** instance must be updated following each valid vote invocation. The **vote** operation supports registration of a **VoteStatement** and

returns a **VoteReceipt** to a client. New instances of **VoteProcessor** may be created using a **ProcessorCriteria** passed as an argument to a **ResourceFactory** where the contained **ProcessorCriteria** model is an instance of **VoteModel**.

2.4.2.1 IDL Specification

```
abstract interface Vote
    {
    readonly attribute VoteCount vcount;

    VoteReceipt vote(
        in VoteDescriptor value
    );
};

interface VoteProcessor:
    Vote,
    Processor
    {
};
```

Table 2-22 Vote Attribute Table

Name	Type	Properties	Purpose
vcount	VoteCount	readonly	Summation of yes, no and abstain votes registered with the processor.

Table 2-23 VoteProcessor Structured Event Table

Event	Description		
vote	Notification of modification of the vcount attribute value.		
	Supplementary properties:		
	value	VoteCount	Summation of yes, no, and abstain vote.

2.4.3 VoteModel

The **VoteModel** valuetype contains the policy to be applied by a **VoteProcessor**. **VoteModel** is accessed through **VoteProcessor** under the **model** operation on the inherited **Simulator** interface. **VoteModel** contains three fields described in the following table that define the rules applicable to the vote process execution.

2.4.3.1 IDL Specification

```
valuetype Duration {
    public TimeBase::TimeT value;
```

```
};

struct VoteCeiling{
    short numerator;
    short denominator;
};

enum VotePolicy{
    AFFIRMATIVE_MAJORITY,
    NON_ABSTAINING_MAJORITY
};

valuetype VoteModel :
    ProcessorModel
    {
    public VoteCeiling ceiling;
    public VotePolicy policy;
    public boolean single;
    public Duration lifetime;
};
```

Table 2-24 VotePolicy Enumeration Table

Element	Description
AFFIRMATIVE_MAJORITY	Indicating that the number of yes votes must be equal to or greater than (VoteCeiling * number of votes registered).
NON_ABSTAINING_MAJORITY	Indicating that the number of yes votes must be equal or greater than (VoteCeiling * (number of votes registered less the total number of abstaining votes)).

Table 2-25 VoteModel State Table

Name	Type	Properties	Purpose
ceiling	VoteCeiling	public	The ceiling exposes a fractional value indicating the proportion of YES votes required to conclude a vote process successfully. Values of ceiling such as or are expressed by the VoteCeiling structure in the form of a numerator and denominator value.
policy	VotePolicy	public	Policy to apply to vote counting – refer to Table 2-24.
single	boolean	public	If true, a vote may not be recast; that is, one vote only. If false, a client may recast a vote.

Table 2-25 VoteModel State Table

lifetime	Duration	public	The maximum lifetime of the vote process commencing on transition of the process to a running state. A zero, negative or null value is equivalent to no constraint on process lifetime.
unilateral	boolean	public	If true, the process of voting shall be considered as binding on all members. If false, then the result of the vote process is considered as binding on members that have voted.

2.5 *EngagementProcessor and EngagementModel*

EngagementProcessor is a type of **Processor** that defines an **engage** operation. The **Engage** operation, defined under the inherited abstract **Engage** interface, is used to facilitate the establishment of **Proof** of agreement between a set of collaborating clients. **EngagementProcessor** contains an **EngagementModel**, exposed through the inherited model operation from the abstract **Model** interface. **EngagementModel** contains a root **Role** used to qualify the number of engagements required for an engagement process to be considered as binding.

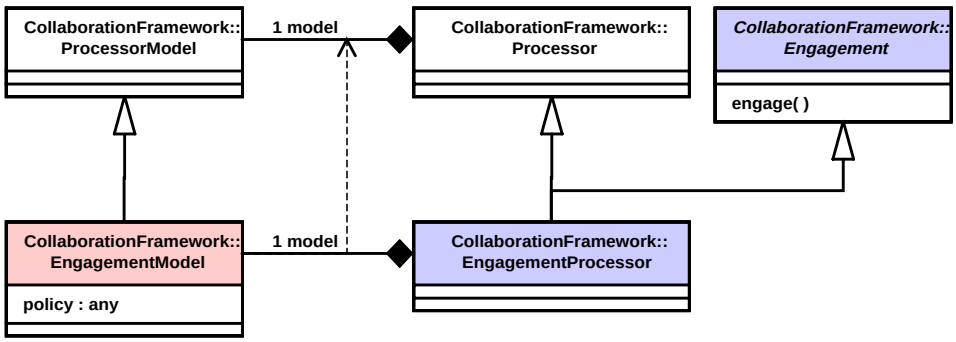


Figure 2-8 EngagementProcessor and EngagementModel

2.5.1 *EngagementProcessor*

An **EngagementProcessor** supports the registration of **Evidence** by a client and return of **Proof** of the act of engagement. **Proof** and **Evidence** are abstract valuetypes that may be specialized to support implementation specific engagement models. **Engagement** policy, also implementation specific is exposed as an instance of **EngagementModel** by the inherited model operation from the abstract **Model** interface under **EngagementProcessor**. New instances of **EngagementProcessor** may be created using a **ProcessorCriteria** passed as an argument to a **ResourceFactory**, where the contained model is an instance of **EngagementModel**.

2.5.1.1 IDL Specification

```
abstract interface Engagement
    {
    Proof engage(
        in CollaborationFramework::Evidence evidence
    ) raises (
        EngagementProblem
    );
};

interface EngagementProcessor :
    Engagement,
    Processor
    {
};
```

Table 2-26 EngagementProcessor Structured Event Table

Event	Description		
vote	Notification of modification of the vcount attribute value.		
	Supplementary properties:		
	value	VoteCount	Summation of yes, no and abstain vote,

Table 2-27 Exceptions Related to the Engage Operation

Exception	Reason
EngagementProblem	Raised following an attempt to invoke engage before the processor is running, or as a result of passing an invalid Evidence valuetype (where validity is implementation defied).

2.5.2 EngagementModel

EngagementModel extends **ProcessorModel** through the addition of three values, a **Role** used to qualify the engagement context, a declaration of the maximum lifetime of an **Engagement** process, and a value indicating if the engagement has a unilateral implication on the members of an associated **Encounter**.

2.5.2.1 IDL Specification

```
valuetype Duration {
    public TimeBase::TimeT value;
};

valuetype EngagementModel :
    ProcessorModel
```

```
    {
    public CommunityFramework::Role role;
    public Duration lifetime;
    public boolean unilateral;
};
```

Table 2-28 EngagementModel State Table

Name	Type	Properties	Purpose
role	Role	public	The value of quorum under this Role indicates the number of engagements required following which engagement is considered as binding.
unilateral	boolean	public	If true, the process of engagement shall be considered as binding on all members. If false, then the act of engagement is considered as binding on members that have actively engaged. Members that have not invoked the engage operation shall not be considered as bound to the engagement.
lifetime	Duration	public	The maximum lifetime of the process commencing on transition of the process to a running state. A zero, negative or null value is equivalent to no constraint on process lifetime.

2.6 CollaborationProcessor, CollaborationModel, and Supporting Types

CollaborationProcessor is a type of **Processor** that contains a model supporting the declaration of states and state transitions. This state model defines a set of rules concerning the way in which a membership can interact towards achievement of a joint conclusion. Examples of collaboration models defined within this specification include bilateral negotiation, multilateral voting, and promissory engagement. The specification approach of separation of structural IDL from a semantic model ensures that the framework can be applied to a range of collaborative processes through the creation of collaboration models that reflect the business rules within different enterprises and across different vertical domains.

We commence with the definition of a **CollaborationProcessor** under Section 2.6.1, “CollaborationProcessor,” on page 2-25, followed by specification of a number of supporting structures under Section 2.6.2, “Supporting Structures,” on page 2-28. Section 2.6.3, “CollaborationModel,” on page 2-30 defines **CollaborationModel** under which the notions of an initialization and state are introduced, together with a description of the relationship to business roles. Section 2.6.4, “State Declaration,” on page 2-31 through Section 2.6.7, “Transition and Related Control Structures,” on page 2-36 present the **CollaborationModel** control structures supporting state declaration under Section 2.6.4, “State Declaration,” on page 2-31, and the semantics of an abstract trigger, an initialization, and the relationship between a Trigger and an action under Section 2.6.5, “Trigger and supporting valuetypes,” on page 2-32. Section 2.6.6, “Action,” on page 2-35 details a set of supported action declarations, including simple

transitions, recursive or local transitions and commands. Section 2.6.7, “Transition and Related Control Structures,” on page 2-36 details the valuetypes used in the definition of a compound transition, a structure that can be used to cause the establishment of a sub-processor and declare the implication of that sub-processor towards the active processor.

The specifications under Section 2.6, “CollaborationProcessor, CollaborationModel, and Supporting Types,” on page 2-24 establish the framework for the definition of a broad range of collaboration models. Chapter 1 of this document details three instances of collaboration criteria (a **ProcessorCriteria** containing a **CollaborationModel**) covering formal negotiation, bilateral interaction leading to a unilateral agreement between a group, and contractual fulfillment.

2.6.1 CollaborationProcessor

CollaborationProcessor is type of **Processor** that contains an instance of **CollaborationModel** (exposed under the model operation on the inherited **Simulator** interface). Operations defined under the inherited abstract **Collaboration** interface provide the ability for a client to modify the state of the processor relative to constraints established under the associated model. In the case of **CollaborationProcessor**, the model defines a nested state hierarchy, and associated transitions. A client can establish an initial collaborative state though invocation of the **apply** operation on the **Collaboration** interface, passing the identifier of a preferred initialization, following which members of an associated membership can invoke the **apply** and **apply_arguments** operations to achieve modification of the collaborative context through state-transitions. Following initialization, the collaboration is established in a running state exposed under the **Collaboration active_state** attribute. The **active_state** attribute is the identifier of a deepest state in a **CollaborationModel** state hierarchy referenced by a proceeding initialization or transition. Establishing an active state has an important implication on the membership associated to the collaboration. Every state from the deepest state referenced by the **active_state** attribute, up through all containing states, until the highest root-state are considered as active. Once a state is classified as active, any **Trigger** instances (transition holders) associated with that state are considered as candidates for subsequent reference under the apply operation.

Triggers contain actions such as transitions and are also associated to business roles that act as guards to the trigger. **Triggers** can be declared as timeout (automatically activated) or launch trigger (explicit activation). Timeout based triggers are activated as a result of modification of the active state path and declared as active under the **CollaborationProcessor timeout_list** attribute.

New instances of a **CollaborationProcessor** may be created by passing a **ProcessorCriteria** instance to a **ResourceFactory create** operation, where the model contained by the **ProcessorCriteria** is an instance **CollaborationModel**.

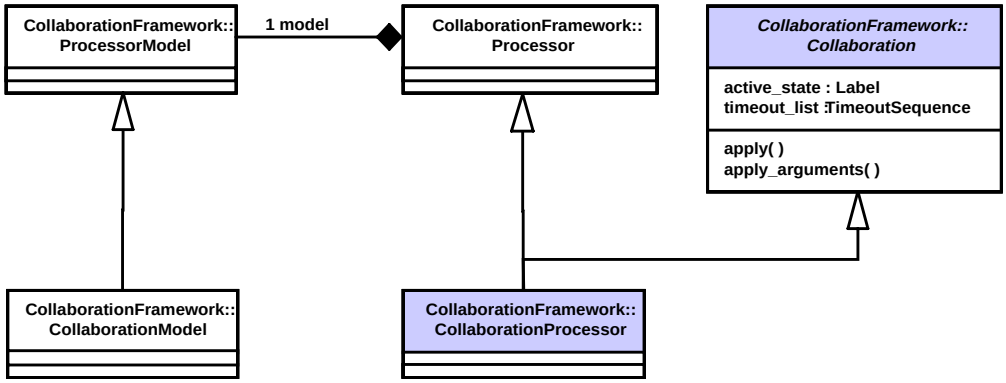


Figure 2-9 Collaboration and CollaborationProcessor

2.6.1.1 IDL Specification

```
abstract interface Collaboration
    {

    readonly attribute Label active_state;
    readonly attribute TimeoutSequence timeout_list;

    void apply(
        in Label identifier
    ) raises (
        InvalidTrigger,
        ApplyFailure
    );

    void apply_arguments(
        in Label identifier,
        in ApplyArguments args
    ) raises (
        InvalidTrigger,
        ApplyFailure
    );
 };

interface CollaborationProcessor :
    Collaboration,
    Processor
    {
};
```

Table 2-29 Collaboration Attribute Table

Name	Type	Properties	Purpose
active_state	Label	readonly	Identifier of the state resulting from an initialization or subsequent transition. All states between the active state and the root top level state constitute the active state path.
timeout_list	TimeoutSequence	readonly	A sequence of Timeout valuetypes corresponding to current activated timeout conditions in place.

Table 2-30 Collaboration Operation Table

Name	Returns	Description
apply	void	Used by a client to modify the state of a collaborative process by passing in a reference to a Trigger in the active state path. Typically used to invoke a transition resulting in the modification of the collaboration context.
apply_arguments	void	Equivalent to apply except that the operation takes a series of arguments corresponding to change request to be applied to the usage relationships associated to the Encounter coordinating the Collaboration.

Table 2-31 Exceptions Related to the Operations Named apply and apply_arguments

Exception	Reason
InvalidTrigger	Raised following an attempt to invoke apply against a Collaboration with an Label that does not correspond to an identified Trigger within the CollaborationModel associated to the Collaboration instance.
ApplyFailure	Raised if a client attempts to invoke apply against the collaboration processor in contravention with the implied or explicit rules exposed by the CollaborationProcess state and associated CollaborationModel.

Table 2-32 CollaborationProcessor Structured Event Table

Event	Description
active	Notification of modification of the active_state attribute value.
	Supplementary properties:

Table 2-32 CollaborationProcessor Structured Event Table

	value	Label	Identifier of the state referenced as a target by an initialization or last transition established under the apply operation.
	timeout	TimeoutSequence	Timeout sequence established as a result of a change in active state.

2.6.2 *Supporting Structures*

2.6.2.1 *Structures Supporting Apply*

The **CollaborationProcessor** interface defines two operations, named **apply** and **apply_arguments**. Both operations concern the modification of the state of a collaboration processor in accordance with the rules and constraints defined in the associated **CollaborationModel** instance. The **apply_arguments** operation takes a sequence of **ApplyArgument** valuetypes as operation arguments. This sequence of **ApplyArgument** instances declares to the processor a set of changes to be applied to the input and output relationships of the attached **Encounter**. For example, a collaboration processor supporting amendment of a standing motion needs to receive the declaration of the amended motion. This is equivalent to modification of the **Usage** links associated with a controlling **Task** (**Encounter**) while a processor is running. **ApplyArgument** is a valuetype that contains the declaration of a **Usage** link **tag** name and a value containing a reference to an **AbstractResource** to be associated to the **Encounter** coordinating the **Collaboration** under a new or existing usage link with the same tag name.

2.6.2.2 *Structures supporting timeout declarations*

A second supporting structure exposed by a **CollaborationProcessor** is a **TimeoutSequence**. A **CollaborationModel** associated to a **CollaborationProcessor** defines a hierarchy of states. Within this hierarchy there may be any number of actions that are configured to execute after a certain delay (refer Clock). The set of active timeout conditions is exposed through the **CollaborationProcessor timeout_list** attribute. A timeout condition is defined through the valuetype **Timeout**. **Timeout** contains an identifier of a Trigger within the **CollaborationModel** associated to the processor, together with a **Timestamp** value indicating the date and time under which the timeout will occur (causing an implementation to automatically invoke the **Action** contained by the **Trigger** referenced by the **Timeout** label).

2.6.2.3 *IDL Specification*

```
valuetype ApplyArgument
    {
    public CollaborationFramework::Label label;
    public Session::AbstractResource value;
};
```

```
valuetype ApplyArguments sequence <ApplyArgument>;

valuetype Timeout
{
    public Label identifier;
    public Session::Timestamp timestamp;
};

valuetype TimeoutSequence sequence <Timeout>;
```

Table 2-33 Timeout State Table

Name	Type	Properties	Purpose
identifier	Label	public	Identifier of a Trigger within the CollaborationModel contained by the CollaborationProcessor that will be fired at the date and time indicated by the timestamp value.
timestamp	Timestamp	public	The date and time that a timeout will be triggered. Timeout conditions may be modified by modification of an active state of a collaboration processor (refer active_state).

Table 2-34 ApplyArgument State Table

Name	Type	Properties	Purpose
tag	Label	public	An ApplyArgument is a valuetype that can be passed into an apply operation. The tag value must be equal to a tag value declared under the processors input usage list (declaration of InputDescriptor values exposed by ProcessorModel usage field). Following assessment of any preconditions associated with a referenced Trigger, an implementation of apply will create or replace an existing consumption link resource value on the associated Task with the value field of the ApplyArgument valuetype.
value	AbstractResource	public	The AbstractResource to associate under a tagged consumption link with the Task associated as coordinator to the Collaboration.

2.6.3 *CollaborationModel*

CollaborationModel is the valuetype that defines the bulk of the semantics behind an instance of **CollaborationProcessor**. **CollaborationModel** extends **ProcessorModel** through addition of a role hierarchy and **State** hierarchy. The entire collaboration model is structurally centered on a state hierarchy, the root of which is defined by the **State** instance exposed under the **state** field. The root-state and sub-states contain the declaration of available triggers (transitions holders) that can be referenced by clients through **apply** operations on the **Collaboration** interface. The **state** field named role contains a **Role** valuetype that represents the root of a role hierarchy that can be referenced by **Trigger** instances (contained by **State** instances) as preconditions to activation. For example, a transition (exposed as **Trigger**) may reference a role as a guard, which in turn introduces a constraint on the invoking client to be associated with the **Encounter** membership under an equivalent role.

As a valuetype, a **CollaborationModel** can be passed between different domains and treated as a self-contained structure that can be readily reused by trading partners. The structural information contained in the inherited **ProcessorModel** defines the logical wiring of a processor towards its coordinating task, while the extensions introduced under **CollaborationModel** define the semantics of collaborative interaction.

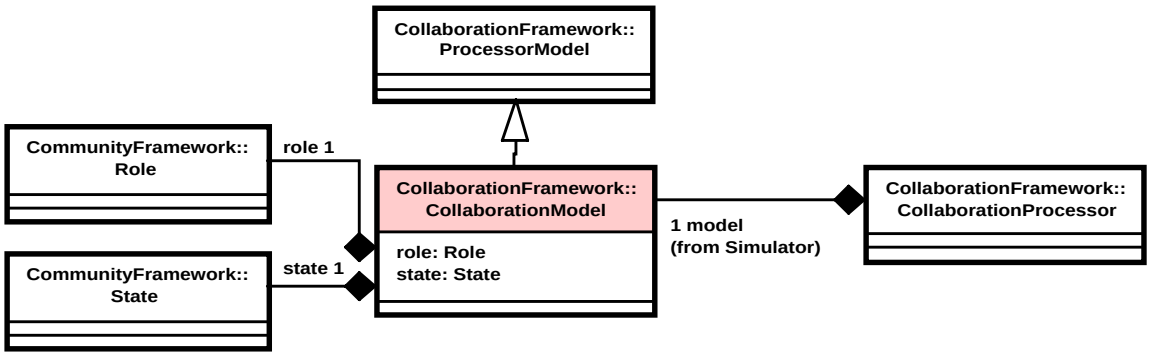


Figure 2-10 CollaborationModel Object Model

2.6.3.1 *IDL Specification*

```
valuetype CollaborationModel :
    ProcessorModel
    {
    public CommunityFramework::Role role;
    public CollaborationFramework::State state;
};
```

Table 2-35 CollaborationModel State Table

Name	Type	Properties	Purpose
role	Role	public	A Role valuetype (refer CommunityFramework) that defines a hierarchy of business roles that may be referenced by other control structures within a CollaborationModel (refer Trigger) for the purpose of establishing membership and quorum preconditions towards an invoking client. This value may be null if all Trigger guard value are also null.
state	State	public	A non-null value defining the root state of the collaboration model. A State is itself a container of other states within which Triggers are contained. Triggers act as constraint guards relative to the Actions they contain.

2.6.4 State Declaration

The primary valuetype used in the construction of a **CollaborationModel** is the **State** valuetype. A **State** is a container of sub-states and **Trigger** valuetypes. An instance of **State** has an identifier label (from the inherited **Control** valuetype), that may be exposed by a **CollaborationProcessor** under the **active_state** attribute. A **State** is activated as a result of a transition action applied through the **apply** operation or through implicit initialization using the **start** operation (from the abstract **Processor** interface inherited by **Collaboration**).

The **Collaboration** declares an **active_state** attribute and a corresponding structured event named **active**. The value of the event and attribute is an identifier of the state referenced in the last valid action (such as an initialization or simple transition). Once an active state has been established, the state containing an active state is considered as active, and as such, its parent, until the root-state is reached. This set of states is referred to as the active state path of the **Collaboration** processor. For every state in the active state path, all directly contained **Triggers** are considered as candidates with respect to the **apply** and **apply_arguments** operations on **CollaborationProcessor**. That is to say that a client may invoke any **Trigger** exposed by a state in the active state path, providing that preconditions to **Trigger** activation are satisfied.

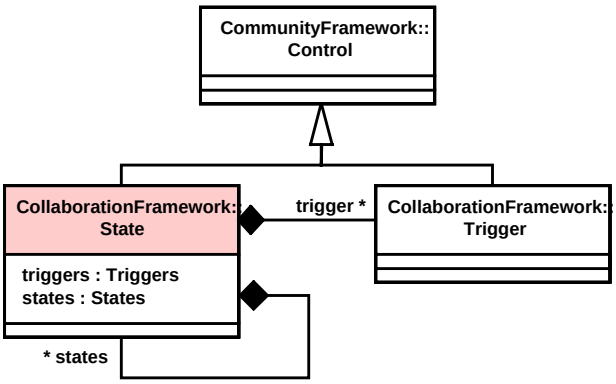


Figure 2-11 State Object Model

2.6.4.1 IDL Specification

```
valuetype State :
    CommunityFramework::Control
    {
    public CollaborationFramework::Triggers triggers;
    public CollaborationFramework::States states;
};
```

Table 2-36 State Valuetype State Table

Name	Type	Properties	Purpose
triggers	Triggers	public	A sequence of Trigger instances that each define constraint conditions relative to a contained Action.
states	States	public	A sequence of sub-states forming a state hierarchy.

2.6.5 Trigger and supporting valuetypes

A **Trigger** is a valuetype contained by a **State** that is used to define an activation constraint (referred to as a **guard**), declarations of implementation actions to fire before action execution (referred to as **directives**), the **action** that a collaboration implementation applies to the collaborative state, and an action **priority**. **Trigger** labels are candidate arguments to the **Collaboration apply** operation when the **State** containing the **Trigger** is within the active state path. The value of guard is a valuetype that qualifies the functional role of the trigger. Two types of **Guard** are defined. A **Clock**, representing a timeout condition that is automatically armed by a **Collaboration** implementation whenever the containing trigger is a candidate (within the active state path). A second type of Guard is a **Launch** that contains a **mode** constraint (one of INITIATOR, RESPONDENT, or PARTICIPANT) and a reference to a **role** that qualifies accessibility of the **Trigger** relative to Members of an associated **Encounter**. A **Trigger** containing a **Clock** is managed by a **Collaboration**

implementation. A **Trigger** containing a **Launch** may be explicitly referenced by a client through the apply operations on the **Collaboration** interface providing the client meets any mode and role constraints associated with the **Trigger**.

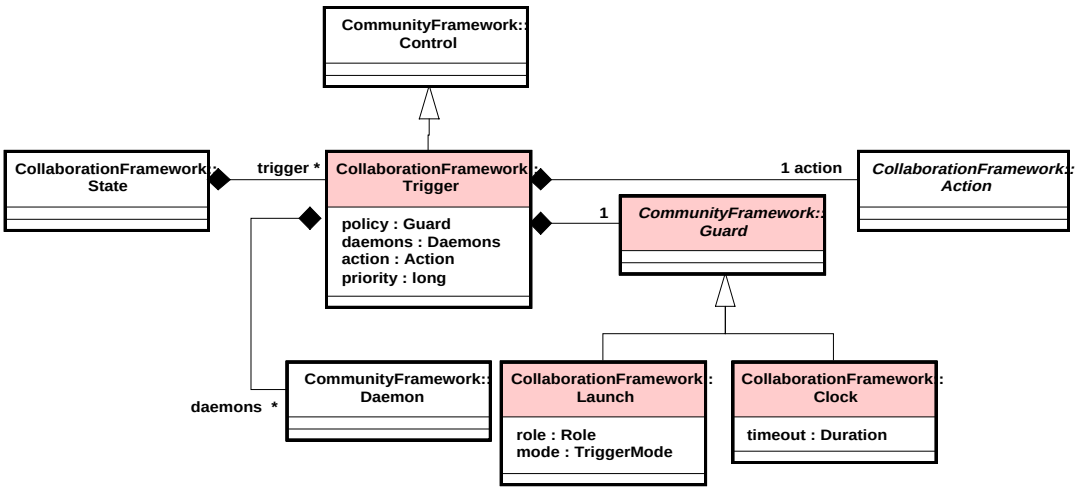


Figure 2-12 AbstractTrigger, Trigger, and Initialization

2.6.5.1 *IDL Specification*

```
valuetype Trigger :
    CommunityFramework::Control
    {
    public long priority;
    public CollaborationFramework::Guard guard; // constraint
    public CollaborationFramework::Directives directives; // preconditions
    public CollaborationFramework::Action action;
};

abstract valuetype Guard {};

valuetype Clock :
    Guard
    {
    public Duration timeout;
};

valuetype Launch :
    Guard
    {
    public TriggerMode mode;
    public CommunityFramework::Role role;
};
```

```
enum TriggerMode{
    INITIATOR,
    RESPONDENT,
    PARTICIPANT
};
```

Table 2-37 Trigger State Table

Name	Type	Properties	Purpose
action	Action	public	An Action valuetype that describes the action to take following client invocation of the apply operation. Argument to apply reference the label that corresponds to the Trigger label state filed inherited from Control.
guard	Guard	public	An instance of Clock or Launch that defines the Trigger activation policy.

Table 2-38 Clock State Table

Name	Type	Properties	Purpose
timeout	Duration	public	Declaration of the delay between establishment of the containing trigger as a candidate (the moment the Trigger's containing state enters the active state path) and the automatic invocation of the action contained by the containing Trigger by a Collaboration implementation.

Table 2-39 Launch State Table

Name	Type	Properties	Purpose
mode	TriggerMode	public	A value corresponding to one of INITIATOR, RESPONDENT or PARTICIPANT.

Table 2-39 Launch State Table

priority	long	public	An implementation of apply is responsible for queuing apply requests relative to trigger priority and invocation order. Higher priority triggers will be fired ahead of lower priority triggers irrespective of apply invocation order. An implementation is responsible for retractions of apply requests following the disassociation of a containing state from the active state path.
role	Role	public	If the role value is not null, a client invoking the containing trigger must be associated to the Encounter under a role with a label equal to the role identifier.

2.6.6 *Action*

The **Action** valuetype is a base type for **Transition**, **CompoundAction**, and **Referral**. Examples of transitions include initialization, simple transition, local transition, and terminal transition. **Transition** can be considered as atomic in that there is no subsequent redirection involved. In comparison, **CompoundTransition** and **Referral** redirects execution towards another action.

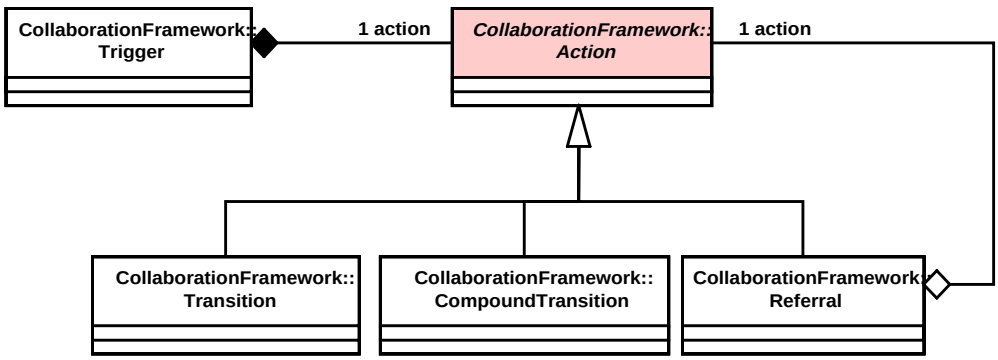


Figure 2-13 Action object model

2.6.6.1 *IDL Specification*

```
abstract valuetype Action
    {
};
```

2.6.7 *Transition and Related Control Structures*

Transition contains a state field named **usage** that contains a **UsageDescriptor** value. The value allows the definition of input and/or output statements (refer **UsageDescriptor**) during a collaborative process execution as a consequence of changes in the collaborative state. A second state field named **transitional** contains a single valuetype derived from the abstract **Transitional** valuetype.

Four types of **Transitional** valuetypes are defined:

- **Initialization**, declares a possible initial active-state target.
- **SimpleTransition**, declares a potential a state transition.
- **LocalTransition**, declares a potential transition from the current state to the current state, during which side effects such as timeout resetting and **Usage** references may be modified.
- **TerminalTransition**, signals termination of the running state of the processor and declares a successful or failure result.

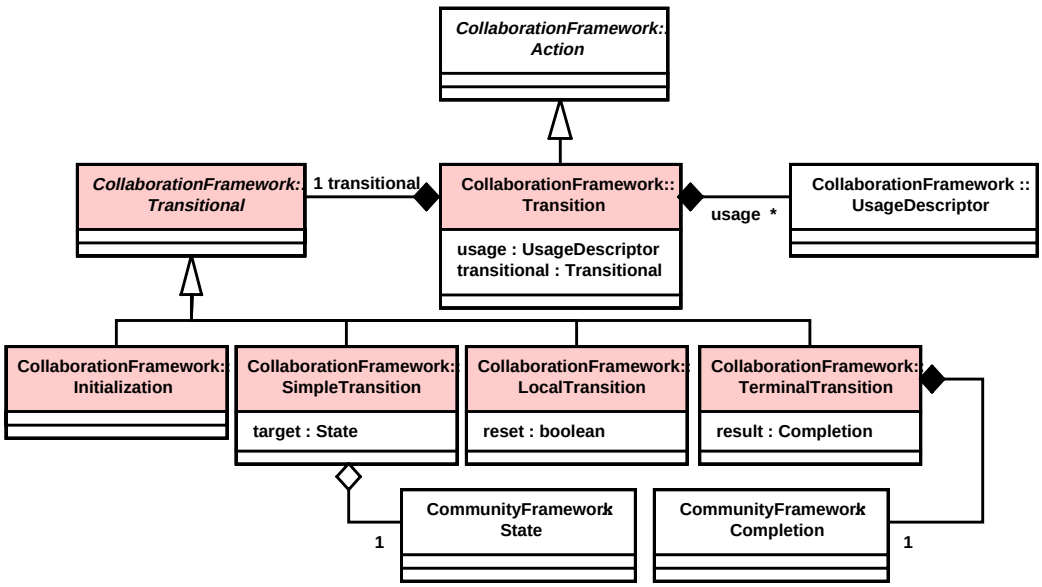


Figure 2-14 Transition and the Transitional family of valuetypes

2.6.7.1 *IDL Specification*

```
abstract valuetype Transitional { };

valuetype Transition :
    Action
    {
    public CollaborationFramework::Transitional transitional;
```

```
    public UsageDescriptors usage;
};

valuetype Initialization :
    Transitional
    {
};

valuetype SimpleTransition :
    Transitional
    {
    public State target;
};

valuetype LocalTransition :
    Transitional
    {
    public boolean reset;
};

valuetype TerminalTransition :
    Transitional
    {
    public Completion result;
};
```

Table 2-40 Transition State Table

Name	Type	Properties	Purpose
usage	UsageDescriptors	public	Contains a sequence of UsageDescriptor instance (input and output declarations) that define required or operational arguments to the Collaboration apply operation when the state containing the usage declaration is active.
transitional	Transitional	public	Declaration of the transitional operator – one of Initialization, SimpleTransition, LocalTransition or TerminalTransition.

Table 2-41 SimpleTransition State Table

Name	Type	Properties	Purpose
target	State	public	The state to be established as the active state of the CollaborationProcessor (refer CollaborationProcessor active_state attribute).

Table 2-42 LocalTransition State Table

Name	Type	Properties	Purpose
reset	boolean	public	If true, any timeout conditions established through Triggers containing Clocks are reset.

Table 2-43 TerminalTransition State Table

Name	Type	Properties	Purpose
result	Completion	public	Declaration of processor termination – the hosting processor will expose the Completion result instance, indicating the success or failure of the process (refer CollaborationProcessor state attribute).

2.6.7.2 Initialization

Initialization is a type of **Transitional** that declares the potential for establishment of the **active_state** as the **State** instance containing a **Trigger** that contains an **Action** that contains an **Initialization**. The containing **State** corresponds to the initalization target. The **Trigger** containing the **Initialization** may declare a priority value. The value of priority is considered in the event of implicit initialization arising from client invocation of the **Processor start** operation. When invoking start, the **Initialization** with the highest priority and non-conflicting constraints set is inferred. Alternatively, a **CollaborationProcessor** may be explicitly initialized by referencing the **Initialization**'s containing **Action** label under the **apply** operations.

SimpleTransition

SimpleTransition is **Transitional** that enables a state transition from the current active state to a **State** declared under by the **SimpleTransition target** value. A successful invocation of **apply** or **apply_arguments** on **CollaborationProcessor** will result in the change of the **CollaborationProcessor** active state to the state referenced by the **target** value.

LocalTransition

LocalTransition enables the possible modification of usage relationships (if the containing **Trigger** enables this), and the possibility to **reset** timeout constraints associated with the containing **Trigger**. **LocalTransition** can be considered as a transition from the current active state to the same state, where side effects concerning timeout and usage relationships can be declared.

Terminal Transition

Starting a **CollaborationProcessor** is enabled through the **start** or **initialize** operation. These actions cause the establishment of an initial active state and active-state path. Actions such as **SimpleTransition** enable modification of the active-state-path leading to the potential exposure of a **TerminalTransition** action. Once a **TerminalTransition** action has been fired, the hosting processor enters a closed and completed state (refer ProcessState). A **CollaborationProcessor** implementation signals this change though modification of the **state** attribute on the inherited **Processor** interface (and corresponding structured event). This attribute returns a **StateDescriptor** which itself contains the **Completion** valuetype declared under the **CollaborationModel TerminalTransition** (indicating **Success** or **Failure** of the process).

2.6.8 Compound Action Semantics

Two valuetypes define indirect action semantics. The first is a **Referral**, an action that references another **Action** instance. The second is **CompoundTransition** that introduces the notion of a transition where the target is defined by the result of the execution of another processor. An implementation of **Collaboration** on triggering a **CompoundTransition**, uses a factory **Criteria** instance defined under the **criteria** field to establish a new sub-processor to the current processor. The result of the sub-process execution is exposed by an instance of **Completion** (refer Completion valuetype). Completion contains a result identifier (refer **ResultClass** and **ResultID**). This identifier is used to establish the **Action** to apply based on a result to action **mapping**.

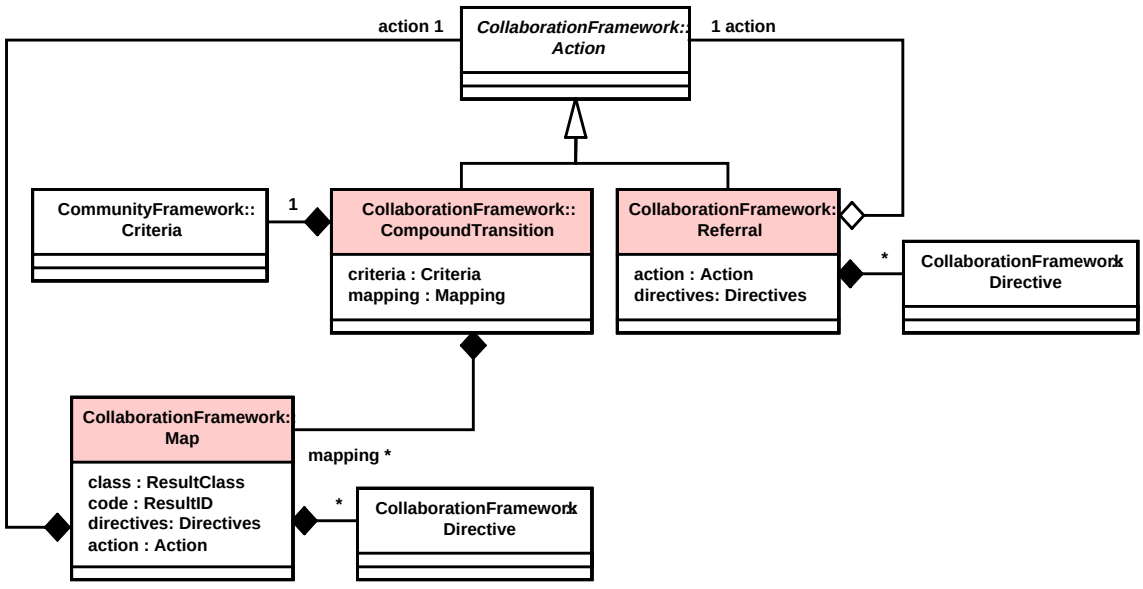


Figure 2-15 CompoundTransition, Referral and Map

Examples of the application of a **Compound** transition are shown in Chapter 1 “Collaboration Criteria.” The fulfillment transition of the promissory contract model is an example of a **CompoundTransition** that uses a bilateral negotiation sub-process between customer and supplier. The result of the negotiation sub-process raises a result state that is mapped by the fulfillment transition to one of two possible outcomes (fulfillment success or failure due to non-fulfillment). A similar use of compound transition is defined under the multilateral voting model in which an amendment is defined as a compound transition applying the same process model as the initial motion.

2.6.8.1 *IDL Specification*

```
valuetype Referral :
    Action
    {
    public CollaborationFramework::Action action;  // reference
    public CollaborationFramework::Directives directives;
};

valuetype Map
    {
    public ResultClass class;
    public ResultID code;
    public CollaborationFramework::Directives directives;
    public CollaborationFramework::Action action;
};

valuetype Mapping sequence <Map> ;

valuetype CompoundTransition :
    Action
    {
    public CommunityFramework::Criteria criteria;
    public CollaborationFramework::Mapping mapping;
};
```

Table 2-44 Referral State Table

Name	Type	Properties	Purpose
action	Action	public	A reference to the action to invoke (refer Action) where the action is an existing Action instance within the containing model.
directives	Directives	public	A sequence of Directive valuetypes that declare modifications (rename, remove, copy and move) to the associated Task usage associations that will be invoked before the action is handled by the Collaboration implementation.

Table 2-45 Map State Table

Name	Type	Properties	Purpose
class	ResultClass	public	One of the enumerated values of SUCCESS or FAILURE
code	ResultID	public	An optional Completion code that qualifies a success or failure class.
action	Action	public	The action to invoke (refer Action).
directives	Directives	public	A sequence of Directive valuetypes that declare modifications (rename, remove, copy and move) to the associated Task usage associations that will be invoked before the action is handled by the Collaboration implementation.

Table 2-46 CompoundTransition State Table

Name	Type	Properties	Purpose
criteria	Criteria	public	An instance of Criteria that is to be used as the criteria for sub-process establishment under a ResourceFactory.
mapping	Mapping	public	A sequence of Map instances defining the actions to be applied in the event of an identified result status. An implementation is responsible for ensuring a complete mapping of all possible sub-process result states to actions within the parent processor prior to initialization (refer verify operation on Collaboration interface).

2.6.9 Directive

Directive is a utility valuetype contained by **Trigger** and **Referral**. It is used to express an execution directive to an implementation of **Collaboration** concerning link associations on the coordinating **Task**. For example, a compound transition can contain a directive that declares that a link be modified before the transition is fired. Another link directive could be contained in a **Map** declaring that the result of the compound transition sub-process must be assigned as an input to the current process. Four concrete valuetypes support the abstract **Directive** interface - **Duplicate**, **Move**, **Remove**, and **Constructor**.

2.6.9.1 Duplicate

Instructs an implementation of **Collaboration** to create a new consumption link named **target** based on the state of a **source** link. If the value of **invert** is false, the type of link created is the same as the source link. If **invert** is true, then if the source link is a **Consumption** link, the created link will be a **Production** link and vice-versa. The resource associated to the new target link shall be the same as the resource declared under the source link.

2.6.9.2 Move

The **Move** directive is a directive to a **Collaboration** implementation to change a source **Consumption** link name to the value of target. If the invert value of the **Move** instance is true, the move directive implies replacement of the link with its inverse type; that is, if the source link is a type of **Consumption** link, then replace the link with a type of **Production** link. If the source link is a type of **Production** link, then replace the link with a type of **Consumption** link.

2.6.9.3 Remove

The **Remove** directive directs a **Collaboration** implementation to remove a tagged **Usage** link (with a tag value corresponding to **source**) from the coordinating **Task**.

2.6.9.4 Constructor

The **Constructor** directive directs a **Collaboration** implementation to create a new resource based on the supplied **criteria** and associate the resource under a new named **Consumption** link on the coordinating **Task** using the **target** value as the links tag value.

2.6.9.5 IDL Specification

```
abstract interface Directive {};
valuetype Directives sequence <Directive>;

valuetype Duplicate
    supports Directive
    {
    public Label source;
    public Label target;
    public boolean invert;
};

valuetype Move
    supports Directive
    {
    public Label source;
    public Label target;
```

```
    public boolean invert;
};

     valuetype Remove
    supports Directive
    {
    public Label source;
};

valuetype Constructor
    supports Directive
    {
    public Label target;
    public CommunityFramework::Criteria criteria;
};
```

Table 2-47 Duplicate State Table

Name	Type	Properties	Purpose
source	Label	public	The name (tag value) of an existing link held by the coordinating Task.
target	Label	public	The name (tag value) of a Usage Link to be created or replaced on the coordinating Task.
invert	boolean	public	If true, an implementation of Collaboration is required to create a new Usage link using the inverse type; that is, if source is Consumption, then target type is Production. If source is Production, then target type is Consumption. The new usage link is added to the coordinating Task.

Table 2-48 Move State Table

Name	Type	Properties	Purpose
source	Label	public	The name (tag value) of an existing link held by the coordinating Task.
target	Label	public	The name (tag value) of a Usage Link to be created or replaced on the coordinating Task.
invert	boolean	public	If true, an implementation of Collaboration is required to replace an existing Usage link with the inverse; that is, Consumption is replaced by Production, Production is replaced by Consumption.

Table 2-49 Remove State Table

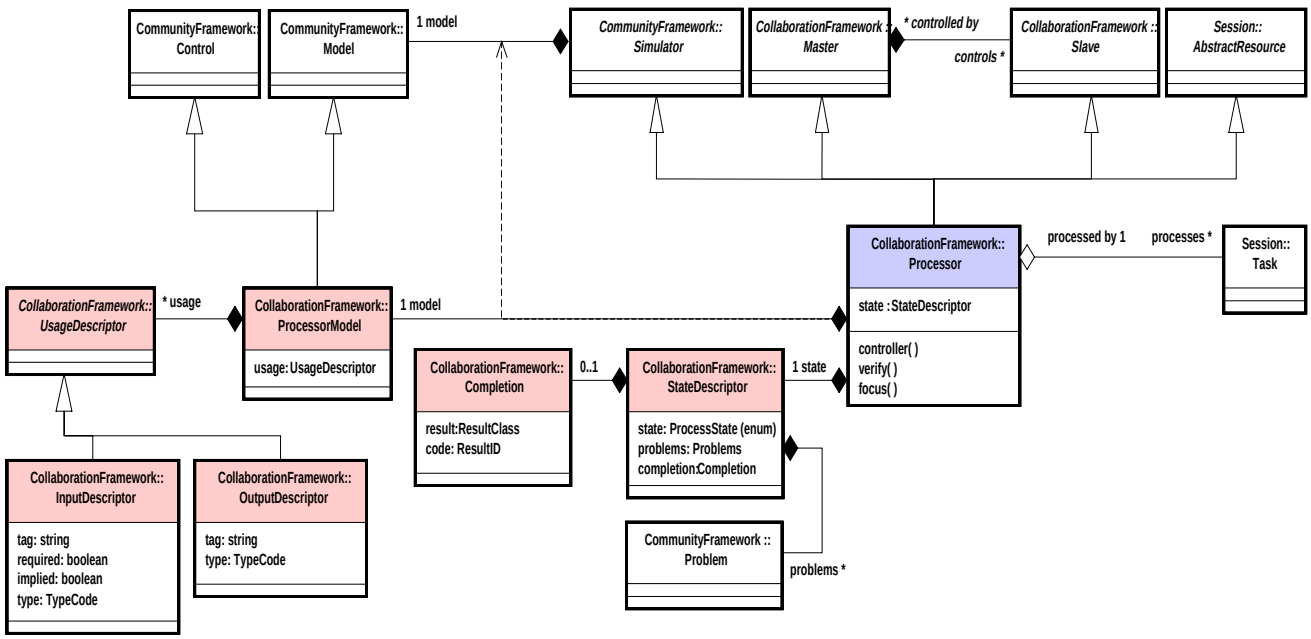
Name	Type	Properties	Purpose
source	Label	public	The name of a Usage Link to be removed from the coordinating Task.

Table 2-50 Constructor State Table

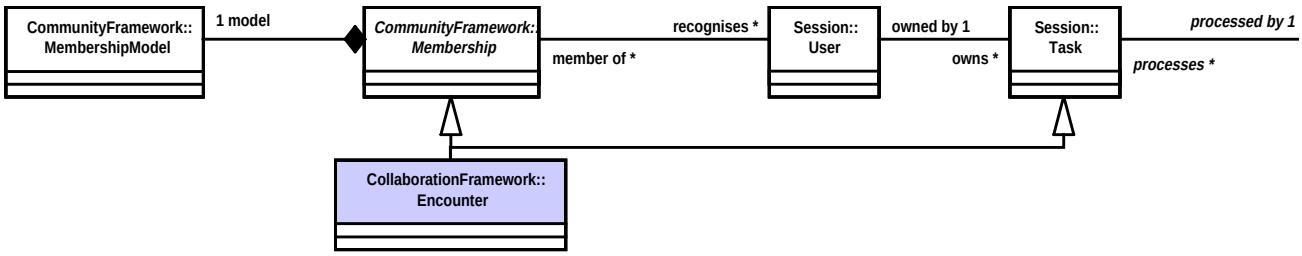
Name	Type	Properties	Purpose
target	Label	public	The name of a Usage Link to be created and added to the coordinating Task (replacing any existing usage link of the same name), using the supplied criteria.
criteria	Criteria	public	An instance of Criteria describing the resource to be created.

2.7 UML Overview

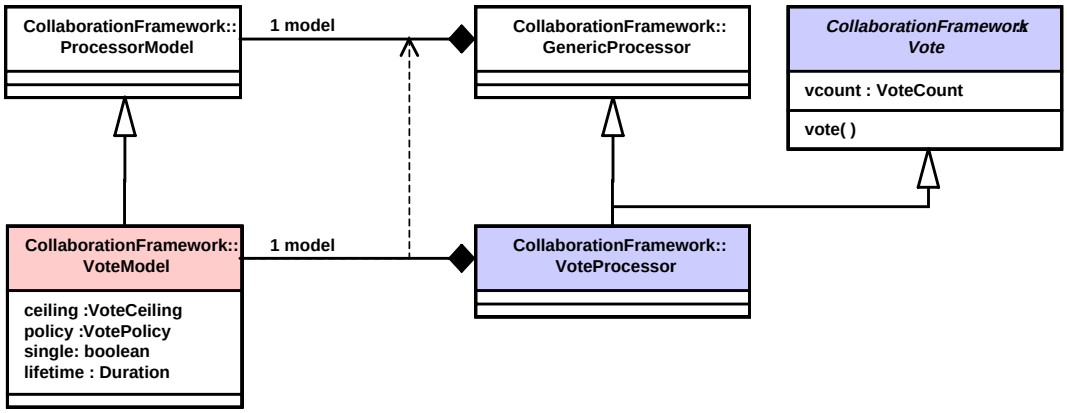
2.7.1 Processor and Related Valuetypes



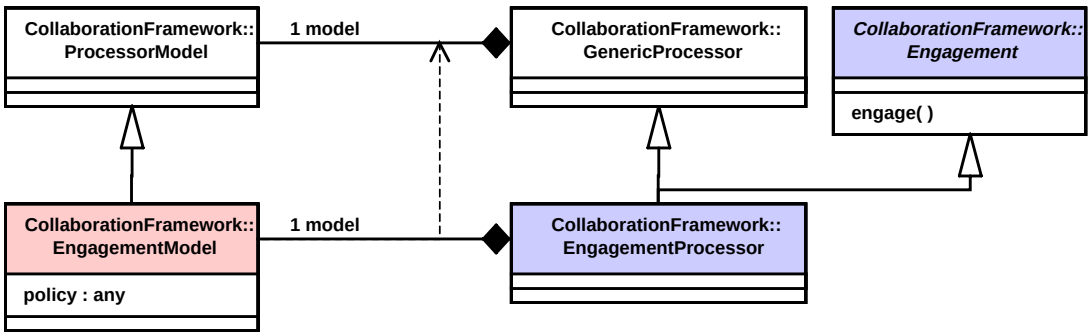
2.7.2 Encounter



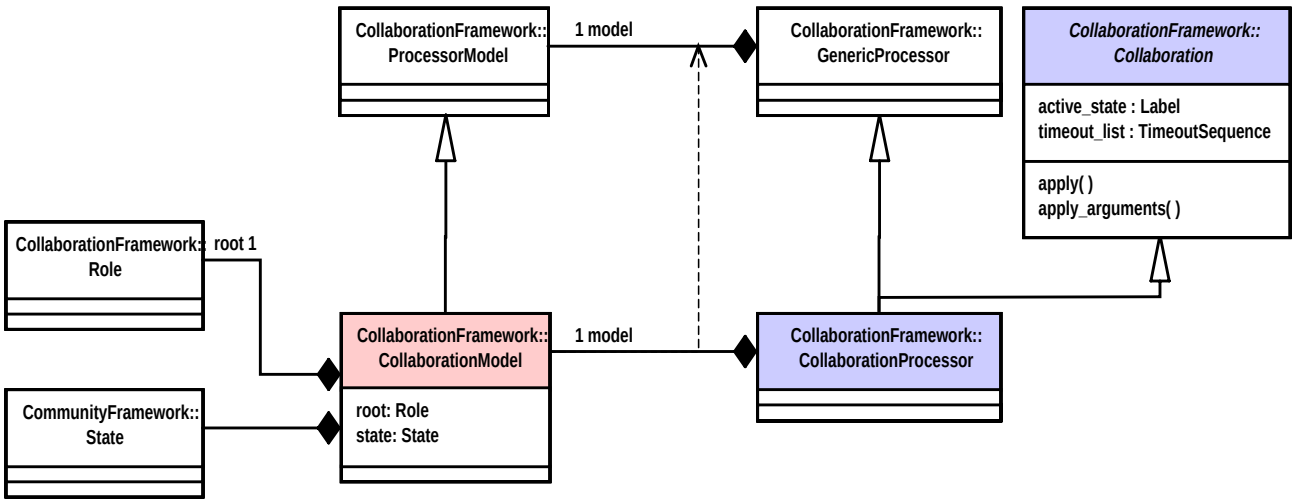
2.7.3 Voting



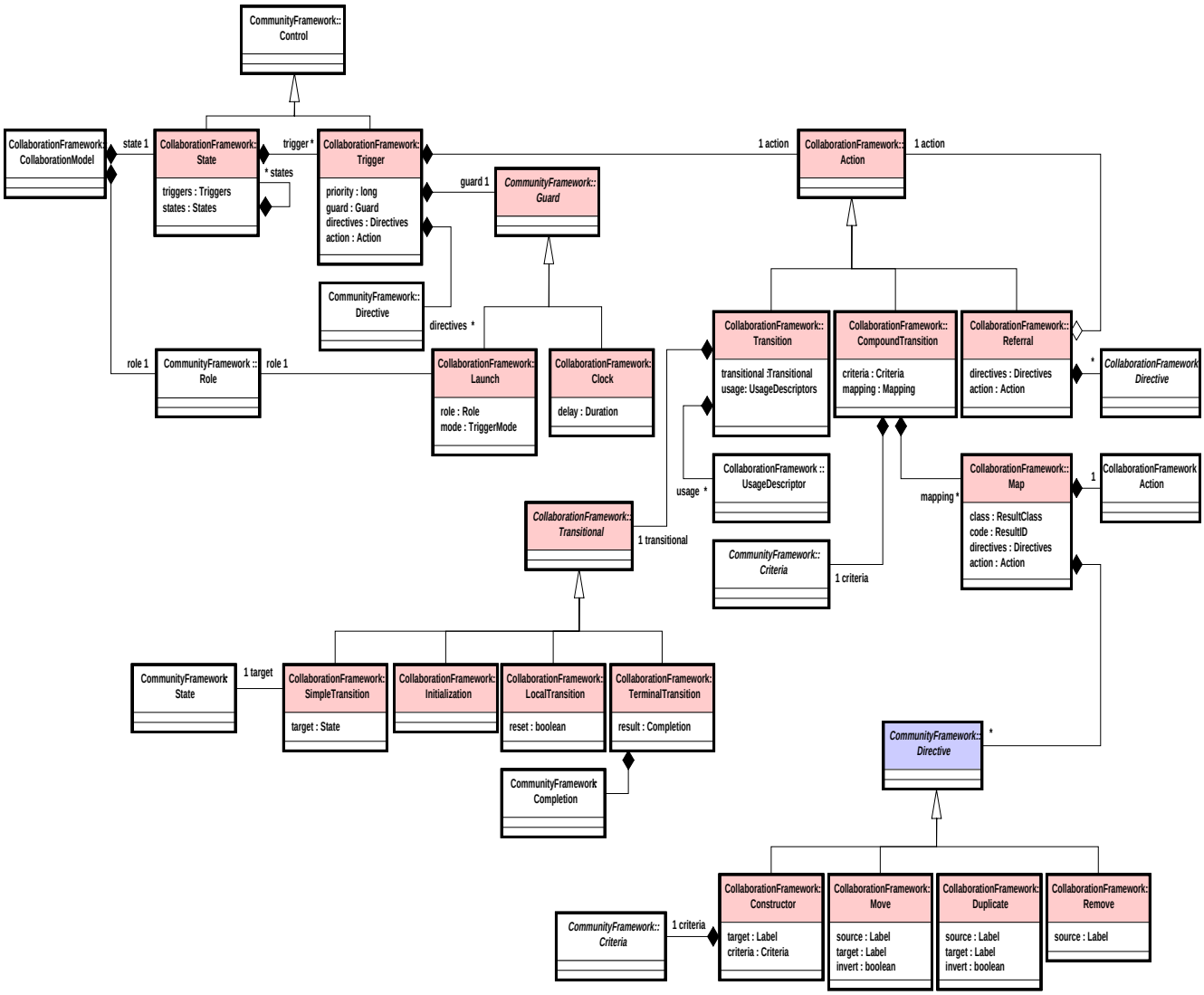
2.7.4 Engagement



2.7.5 Collaboration and CollaborationModel



2.7.6 Valuetypes Supporting CollaborationModel



2.8 CollaborationFramework Complete IDL

```
#ifndef _COLLABORATION_IDL_
#define _COLLABORATION_IDL_
#include <CommunityFramework.idl>
#pragma prefix "omg.org"

module CollaborationFramework{

#pragma version CollaborationFramework 2.0

//  forward declarations

abstract valuetype Action;
abstract valuetype Transitional;
abstract valuetype Guard;
abstract valuetype Proof;
abstract valuetype Evidence;
abstract valuetype UsageDescriptor;

valuetype State;
valuetype Initialization;
valuetype Trigger;
valuetype Transition;
valuetype SimpleTransition;
valuetype LocalTransition;
valuetype TerminalTransition;
valuetype CompoundTransition;
valuetype Referral;

abstract interface Slave;
abstract interface Master;
abstract interface Collaboration;
abstract interface Engagement;
abstract interface Vote;
abstract interface Directive;

interface Encounter;
interface Processor;
interface VoteProcessor;
interface EngagementProcessor;
interface CollaborationProcessor;

// typedefs

valuetype States sequence <State> ;
valuetype Triggers sequence <Trigger> ;
valuetype Initializations sequence <Initialization> ;
valuetype UsageDescriptors sequence <UsageDescriptor> ;
valuetype Slaves sequence <Slave> ;
```

```
valuetype Directives sequence <Directive>;
valuetype Label CommunityFramework::Label;
valuetype ProcessorState Session::task_state;
valuetype ResultID unsigned long ;
valuetype TypeCode CORBA::TypeCode;
valuetype ResultClass boolean;

// structures

valuetype Duration {
    public TimeBase::TimeT value;
};

struct VoteCeiling{
    short numerator;
    short denominator;
};

enum VotePolicy{
    AFFERMATIVE_MAJORITY,
    NON_ABSTAINING_MAJORITY
};

abstract valuetype Proof {};
abstract valuetype Evidence {};

enum VoteDescriptor{
    NO,
    YES,
    ABSTAIN
};

valuetype VoteStatement :
    Evidence
    {
    public VoteDescriptor vote;
};

valuetype VoteReceipt :
    Proof
    {
    public Session::Timestamp timestamp;
    public VoteStatement statement;
};

valuetype VoteCount :
    Proof
    {
    public Session::Timestamp timestamp;
    public long yes;
    public long no;
```

```
        public long abstain;
};

valuetype Timeout{
        public Label identifier;
        public Session::Timestamp timestamp;
};

valuetype TimeoutSequence sequence <Timeout> ;

enum TriggerMode{
        INITIATOR,
        RESPONDENT,
        PARTICIPANT
};

valuetype Completion
        {
        public ResultClass result;
        public ResultID code;
};

valuetype StateDescriptor
        {
        public ProcessorState state;
        public CollaborationFramework::Completion completion;
        public CommunityFramework::Problems problems;
};

// exceptions

exception InvalidTrigger{
        CommunityFramework::Problem problem;
        Label identifier;
};

exception ApplyFailure{
        CommunityFramework::Problem problem;
        Label identifier;
};

exception InitializationFailure{
        CommunityFramework::Problem problem;
        Label identifier;
};

exception EngagementProblem{
        CollaborationFramework::Evidence evidence;
        CommunityFramework::Problem problem;
};
```

```
interface SlaveIterator : CosCollection :: Iterator { };

// coordination link

abstract valuetype Coordination : Session::Execution{ };

valuetype Monitors : Coordination {
    public Processor resource;
};

valuetype Coordinates : Monitors {};

valuetype CoordinatedBy : Coordination {
    public Session::Task resource;
};

// management link

abstract valuetype Management : Session::Link{ };

valuetype Controls : Management {
    public Slave resource;
};

valuetype ControlledBy : Management {
    public Master resource;
};

/**
Encounter
*/

interface Encounter :
    Session::Task,
    CommunityFramework::Membership
    {
};

valuetype EncounterCriteria :
    CommunityFramework::Criteria
    {
    public CommunityFramework::MembershipModel model;
};

/*
ProcessorModel
*/

abstract valuetype UsageDescriptor { };

valuetype InputDescriptor :
```

```
        UsageDescriptor
        {
        public string tag;
        public boolean required;
        public TypeCode type;
};

valuetype OutputDescriptor :
        UsageDescriptor
        {
        public string tag;
        public TypeCode type;
};

valuetype ProcessorModel :
        CommunityFramework::Control
        supports CommunityFramework::Model
        {
        public UsageDescriptors usage;
};

/**
Master, Slave and Processor.
*/

abstract interface Master {
        SlaveIterator slaves (
                in long max_number,
                out Slaves slaves
        );
};

abstract interface Slave {
        readonly attribute CollaborationFramework::Master master;
};

abstract interface Processor :
        Session::AbstractResource,
        CommunityFramework::Simulator,
        Master, Slave
        {

        readonly attribute StateDescriptor state;

        Session::Task coordinator(
        ) raises (
                Session::ResourceUnavailable
        );

        CommunityFramework::Problems verify( );
```

```
        void start (
        ) raises (
            Session::CannotStart,
            Session::AlreadyRunning
        );
        void suspend (
        ) raises (
            Session::CannotSuspend,
            Session::CurrentlySuspended
        );
        void stop (
        ) raises (
            Session::CannotStop,
            Session::NotRunning
        );
};

valuetype ProcessorCriteria :
        CommunityFramework::Criteria
        {
        public ProcessorModel model;
};

/**
Engagement
*/

abstract interface Engagement
        {
        Proof engage(
            in CollaborationFramework::Evidence evidence
        ) raises (
            EngagementProblem
        );
};

interface EngagementProcessor :
        Engagement,
        Processor
        {
};

 valuetype EngagementModel :
        ProcessorModel
        {
        public CommunityFramework::Role role;
        public Duration lifetime;
        public boolean unilateral;
};

/**
```

```
Vote.
*/

abstract interface Vote
     {
     readonly attribute VoteCount vcount;

     VoteReceipt vote(
          in VoteDescriptor value
     );
};

interface VoteProcessor :
     Vote,
     Processor
     {
};

     valuetype VoteModel :
     ProcessorModel
     {
     public VoteCeiling ceiling;
     public VotePolicy policy;
     public boolean single;
     public Duration lifetime;
};

/**
Collaboration
*/

// directive

abstract interface Directive {};

valuetype Duplicate
     supports Directive
     {
     public Label source;
     public Label target;
     public boolean invert;
};

valuetype Move
     supports Directive
     {
     public Label source;
     public Label target;
     public boolean invert;
};
```

```
        valuetype Remove
    supports Directive
    {
    public Label source;
};

valuetype Constructor
    supports Directive
    {
    public Label target;
    public CommunityFramework::Criteria criteria;
};

// apply arguments

valuetype ApplyArgument
    {
    public CollaborationFramework::Label label;
    public Session::AbstractResource value;
};

valuetype ApplyArguments sequence <ApplyArgument> ;

// collaboration

abstract interface Collaboration
    {

    readonly attribute Label active_state;
    readonly attribute TimeoutSequence timeout_list;

    void apply(
        in Label identifier
    ) raises (
        InvalidTrigger,
        ApplyFailure
    );

    void apply_arguments(
        in Label identifier,
        in ApplyArguments args
    ) raises (
        InvalidTrigger,
        ApplyFailure
    );
 };

interface CollaborationProcessor :
    Collaboration,
    Processor
    {
```

```
};

/**
Collaboration controls
*/

valuetype State :
    CommunityFramework::Control
    {
    public CollaborationFramework::Triggers triggers;
    public CollaborationFramework::States states;
};

abstract valuetype Guard {};

valuetype Clock :
    Guard
    {
    public Duration timeout;
};

valuetype Launch :
    Guard
    {
    public TriggerMode mode;
    public CommunityFramework::Role role;
};

valuetype Trigger :
    CommunityFramework::Control
    {
    public long priority;
    public CollaborationFramework::Guard guard;
    public CollaborationFramework::Directives directives; // precondition
    public CollaborationFramework::Action action;
};

abstract valuetype Action { };

abstract valuetype Transitional { };

valuetype Transition :
    Action
    {
    public CollaborationFramework::Transitional transitional;
    public UsageDescriptors usage;
};

valuetype Initialization :
    Transitional
    {
```

```
};

valuetype SimpleTransition :
    Transitional
    {
    public State target;
};

valuetype LocalTransition :
    Transitional
    {
    public boolean reset;
};

valuetype TerminalTransition :
    Transitional
    {
    public Completion result;
};

valuetype Referral :
    Action
    {
    public CollaborationFramework::Action action;
    public CollaborationFramework::Directives directives;
};

valuetype Map
    {
    public ResultClass class;
    public ResultID code;
    public CollaborationFramework::Directives directives;
    public CollaborationFramework::Action action;
};

valuetype Mapping sequence <Map> ;

valuetype CompoundTransition :
    Action
    {
    public CommunityFramework::Criteria criteria;
    public CollaborationFramework::Mapping mapping;
};

valuetype CollaborationModel :
    ProcessorModel
    {
    public CommunityFramework::Role role;
```

```
        public CollaborationFramework::State state;
    };
};

#endif // _COLLABORATION_IDL_
```

Community Framework 3

Contents

This chapter contains the following sections.

3.1 Overview

The **CommunityFramework** defines a specialization of the Task and Session Workspace called **Community** and a specialization of **Community** called **Agency**. **Community** is defined as a specialization of **Workspace** and an abstract interface called **Membership**. **Agency** is a specialization of a **Community** that introduces the abstract interface **LegalEntity**.

Table 3-1 Principle Interfaces - Summary Table

Interface	Description
Community	The Community type combines the definition of Workspace from the Task and Session framework. Community is derived from the abstract interfaces Membership and Simulator.
Agency	Agency extends Community through the addition of the abstract interface named LegalEntity.
GenericResource	A type of AbstractResource used to wrap another object.

Table 3-2 Abstract Interfaces and Supporting Valuetypes - Summary Table

Interface	Description
Simulator	An abstract interface used to expose a valuetype supporting the Model valuetype.
Model	An abstract interface supported by valuetypes used for models that declares execution policy.
Control	A valuetype with identity, a label and human readable description.
Role	A valuetype derived from Control that defines a hierarchy of business roles and associated role policies.
RolePolicy	A valuetype defining policy of a business role.
MembershipModel	An extension of Control supporting the abstract Membership interface that exposes Membership policy and a role hierarchy.
MembershipPolicy	A valuetype used to define the policy applicable to a Membership. Contained by MembershipModel.
Membership	Membership is an abstract interface that enables association, qualification and retraction of instances of the type User with a concrete type derived from Membership (such as Community and Agency). Users are associated to a Membership through a type of Link called Member.
Member	A valuetype used to describe the association of a User to a Membership (inverse of Recognizes).
Recognizes	A valuetype used to describe the association of a Membership to a User (inverse of Member)

Table 3-3 Factory Related Interfaces and Valuetypes - Summary Table

Interface	Description
ResourceFactory	An abstract interface defining factory operations based on supplied Criteria valuetypes.
Problem	A valuetype used to describe issues relating or contributing to an exception condition.
Criteria	Abstract interface supported by ExternalCriteria, CommunityCriteria, AgencyCriteria and GenericCriteria.
ExternalCriteria	A criteria valuetype used as a container of an XML public and system identifier of criteria related information resources.
CommunityCriteria	A valuetype used as an argument to a resource factory. It contains a MembershipModel that defines the business semantics of the community to be created.
AgencyCriteria	A valuetype used as an argument to a resource factory. It contains a MembershipModel that defines the business semantics of the agency to be created.
GenericCriteria	A valuetype used as a criteria argument to a resource factory.

3.2 Model, Simulator, and Supporting Valuetypes

The interfaces defined under the **CommunityFramework** separate the notion of service object managed by a particular domain, (typically reference objects derived from the *Task and Session* specification) from valuetype used to describe policy or state. An abstract interface named **Simulator** defines the **model** attribute that returns a valuetype supporting the abstract **Model** interface. From a computational point of view, a by-reference object such as **Community** or **Agency** is a manager and container of a related model valuetype.

3.2.1 Model

A **Model** is an abstract interface supported by valuetypes exposed by the **Simulator** model attribute. An example of a valuetype that supports **Model** is **MembershipModel** (additional types supporting the **Model** abstract interface are defined under the **CollaborationFramework**).

3.2.1.1 IDL Specification

```
abstract interface Model
    {
};
```

3.2.2 Simulator

A **Simulator** is an abstract interface that defines a single attribute through which a client can access a related **Model**. A model valuetype defines constraints and operational semantics. Implementations of concrete simulators (such as **Community** and **Agency**) are responsible for ensuring that the appropriate type of model is returned through to the client. For example, a **Community** implementation of the **model** operation will return an instance of **MembershipModel**.

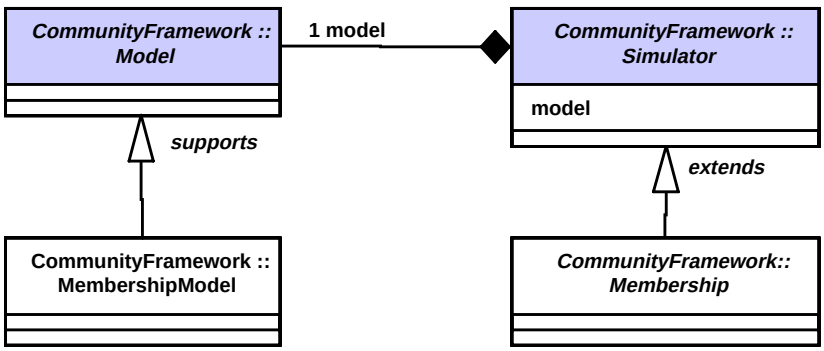


Figure 3-1 Model and Simulator

3.2.2.1 IDL Specification

```
abstract interface Simulator
    {
    readonly attribute CommunityFramework::Model model;
};
```

Table 3-4 Simulator Attribute Table

Name	Type	Properties	Purpose
model	Model	readonly	Access to a valuetype supporting the abstract Model interface.

3.2.3 Control

Control is an identifiable valuetype used in definition of valuetypes defining complex models. **Control** contains a human readable label and descriptive note. **Control** is used as a utility state container by several valuetypes defined within the **Community** and **Collaboration** frameworks.

3.2.3.1 IDL Specification

```
valuetype Label CORBA::StringValue;
valuetype Note CORBA::StringValue;
```

```
valuetype Control :
    {
    public CommunityFramework::Label  label;
    public CommunityFramework::Note  note;
};
```

Table 3-5 Control State Table

Name	Type	Properties	Purpose
label	Label	public	Name of the control.
note	Note	public	Descriptive text.

3.3 Membership, MembershipPolicy, and Member Link

The abstract **Membership** interface declares a set of operations supporting the association of **Users** (refer *Task and Session* specification) under a single policy domain. **Operations** provide support for the addition, modification and removal of a **User** association, access to the quorum status of a membership, and access to information about the set of associated **Users**. Membership to User association is through a link named **Member** (derived from the Task and Session Link).

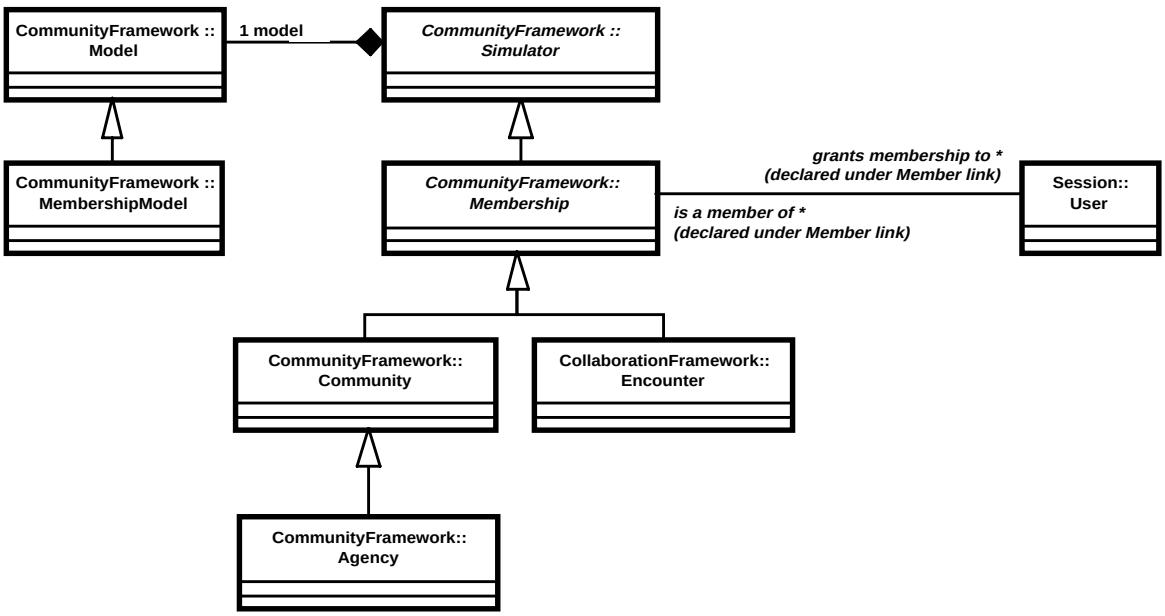


Figure 3-2 Membership Object Model

3.3.1 Membership

Membership is an abstract interface inherited by **Community** that defines operations supporting association and retraction of users under **Member** links, the qualification of members in terms of business roles, and operations supporting access to information about associated **Users**. A **MembershipModel** qualifies membership behavior. The **MembershipModel** defines a hierarchy of business roles that qualify the association between a **User** and the **Membership**. In addition, **MembershipModel** declares policy concerning privacy of **Member** relationship information, **User** to role association, and exclusivity of the membership.

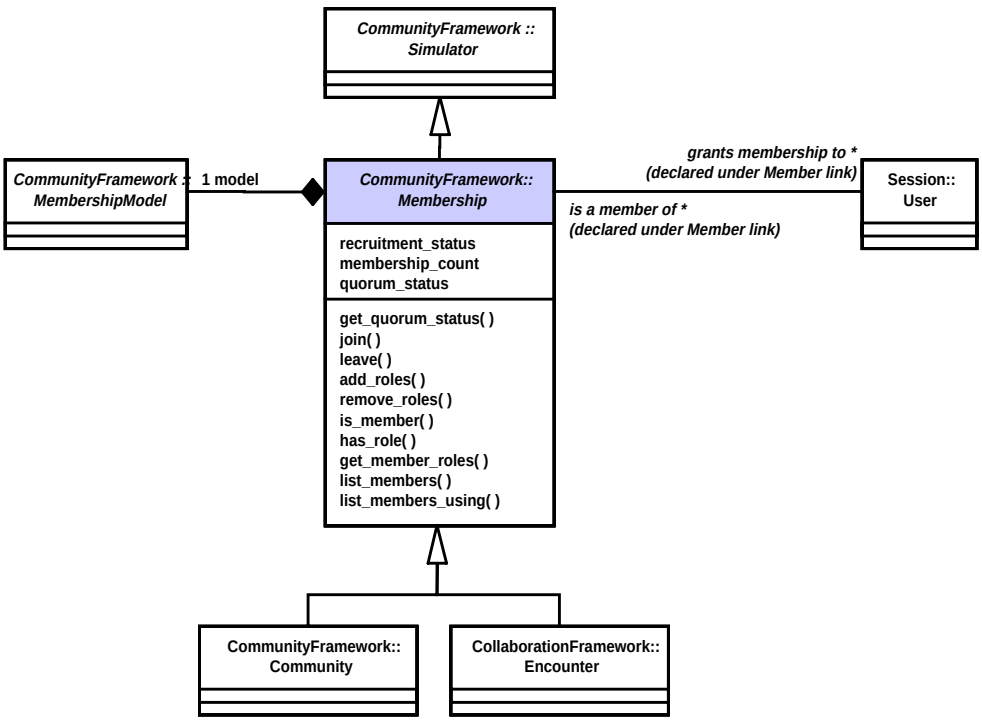


Figure 3-3 Membership Abstract Interface Object Model

3.3.1.1 IDL Specification

```
abstract interface Membership :
    Simulator
    {

    readonly attribute RecruitmentStatus recruitment_status;
    readonly attribute MembershipCount membership_count;
    readonly attribute boolean quorum_status;

    RoleStatus get_quorum_status(
```

```
        in Label identifier
);

Member join(
        in Session::User user,
        in Labels roles
) raises (
        AttemptedCeilingViolation,
        AttemptedExclusivityViolation,
        RecruitmentConflict,
        RoleAssociationConflict,
        MembershipRejected,
        UnknownRole
);

void leave(
        in CommunityFramework::Member member
) raises (
        RecruitmentConflict,
        UnknownMember
);

void add_roles(
        in CommunityFramework::Member member,
        in Labels roles
) raises (
        UnknownMember,
        RoleAssociationConflict,
        UnknownRole
);

void remove_roles(
        in CommunityFramework::Member member,
        in Labels roles
) raises (
        UnknownRole,
        UnknownMember,
        CannotRemoveRole
);

boolean is_member(
        in Session::User user
) raises (
        PrivacyConflict
);

boolean has_role(
        in Session::User user,
        in Label role
) raises (
        PrivacyConflict
```

```
        );

        Labels get_member_roles(
            in Session::User user
        ) raises (
            PrivacyConflict
        );

        Session::UserIterator list_members(
            in long max_number,
            out Session::Users list
        ) raises (
            PrivacyConflict
        );

        Session::UserIterator list_members_using(
            in Label role,
            in long max_number,
            out Session::Users list
        ) raises (
            PrivacyConflict
        );

};

exception PrivacyConflict
        {
        PrivacyPolicyValue reason;
};

exception AttemptedCeilingViolation{
        Membership source;
};

exception AttemptedExclusivityViolation{
        Membership source;
};

exception UnknownRole{
        Membership source;
};

exception UnknownMember{
        Membership source;
        Member link;
};

exception UnknownIdentifier{
        Membership source;
        Label identifier;
};
```

```
exception MembershipRejected{
    Membership source;
    string reason;
};

exception RoleAssociationConflict{
    Membership source;
    string reason;
    Label role;
};

exception CannotRemoveRole{
    Membership source;
    string reason;
    Label role;
};

exception RecruitmentConflict{
    Membership source;
    RecruitmentStatus reason;
};
```

3.3.1.2 Operations supporting association and retraction of Users

The **join** operation allows a client to associate a **User** reference with a **Membership** under a set of declared business roles (refer MembershipPolicy). The **join** operation returns a **Member** instance to be maintained by the **User** instance.

```
Member join(
        in Session::User user,
        in Lables roles
    ) raises (
        AttemptedCeilingViolation,
        AttemptedExclusivityViolation,
        RecruitmentConflict,
        RoleAssociationConflict,
        MembershipRejected,
        UnknownRole
    );
```

The **leave** operation disassociates a **Member** from a **Membership**.

```
void leave(
        in CommunityFramework::Member member
    ) raises (
        RecruitmentConflict,
        UnknownMember
    );
```

Table 3-6 Exceptions Related to the Join and Leave Operations

Exception	Reason
AttemptedCeilingViolation	An attempt is made to add a member association to a Membership where the number of Members is equal to or greater than the ceiling state field value exposed by the associated MemberPolicy instance.
AttemptedExclusivityViolation	If the associated MemberPolicy declares exclusive as true, then for any identifiable principal (CORBA Current Principal) there may be only one Member association for that principal.
RecruitmentConflict	May be raised at the discretion of an implementation when an attempt is made to join or leave a Membership when the recruitment status is CLOSED.
RoleAssociationConflict	Raised when an attempt is made to associate a Member to an abstract role.
MembershipRejected	Implementation specific decision to disallow a join request.
UnknownRole	Raised when an attempt is made to association a Member under an unknown role.
UnknownMember	May be raised at the discretion of an implementation following an attempt to disassociate a Member from a Membership.

3.3.1.3 Operations supporting modification of business roles assigned to Members

The **add_roles** operation enables the addition of business roles attributed to a **Member**.

```
void add_roles(
        in CommunityFramework::Member member,
        in Labels roles
    ) raises (
        UnknownMember,
        RoleAssociationConflict,
        UnknownRole
    );
```

The **remove_roles** operation enables the retraction of business roles attributed to a **Member**.

```
void remove_roles(
        in CommunityFramework::Member member,
        in Labels roles
    ) raises (
        UnknownRole,
        UnknownMember,
        CannotRemoveRole
    );
```

Table 3-7 Exceptions Related to the Role Association

Exception	Reason
UnknownRole	Raised following an attempt to associate or disassociate a Member when the supplied role identifier is unknown; that is, not defined within the associated MembershipPolicy.
UnknownMember	May be raised at the discretion of an implementation following an attempt to add or remove a role from/to a Member.
CannotRemoveRole	Raised if a role removal would result in no role association towards the Member.

3.3.1.4 Attributes and Operations supporting access to recruitment and quorum state

The following attribute returns the recruitment status of a **Membership**. The value returned is one of the enumeration values **OPEN_MEMBERSHIP** or **CLOSED_MEMBERSHIP**. Modification of the recruitment status of a **Membership** is implementation specific. When a **Membership** is under a **CLOSED_MEMBERSHIP**, an implementation may raise the RecruitmentConflict exception.

```
enum RecruitmentStatus{
    OPEN_MEMBERSHIP,
    CLOSED_MEMBERSHIP
};

// from Membership

    readonly attribute RecruitmentStatus recruitment_status;
```

The following attribute supports access to the number of associated **Member** instances. The valuetype **MembershipCount** contains two values, the number of **Member** instances associated to the **Membership** (static field), and the number of **Member** instances referencing connected **Users** at the time of invocation (refer Task and Session, User, Connected State).

```
valuetype MembershipCount{
    public long static;
    public long active;
};

// from Membership

    readonly attribute MembershipCount membership_count;
```

The following attribute returns true if all roles defined within the associated **MembershipPolicy** have met quorum – that is to say that for each role, the number of member instances associated with that role, equal or exceed the quorum value defined under the **RolePolicy** associated with the given role (refer RolePolicy).

```
// from Membership

    readonly attribute boolean quorum_status;
```

Quorum status relating to individual roles is available through the **get_quorum_status** operation. The identifier argument corresponds to identify of a role exposed within a **MembershipModel**.

```
// from Membership

    RoleStatus get_quorum_status(
        in Label identifier
    );
```

Possible **QuorumStatus** values correspond to **QUORUM_VALID**, indicating that all roles have reached quorum, **QUORUM_PENDING**, indicating that the role has not reached quorum, and the special case of **QUORUM_UNREACHABLE**, indicating that the maximum number of members required for a particular role is less than the minimum required.

```
enum QuorumStatus {
    QUORUM_VALID,
    QUORUM_PENDING,
    QUORUM_UNREACHABLE
};

 valuetype RoleStatus
    {
    public Label identifier;
    public MembershipCount count;
    public QuorumStatus status;
};
```

3.3.1.5 Operations supporting access to information about members

The **is_member** operation returns true if the supplied **User** is a member of the membership.

```
// from Membership

    boolean is_member(
        in Session::User user
    ) raises (
        PrivacyConflict
    );
```

The **has_role** operation returns true if the supplied **User** is associated to the **Membership** under a role corresponding to the supplied identifier.

```
// from Membership

    boolean has_role(
        in Session::User user,
        in Label role
    ) raises (
        PrivacyConflict
    );
```

The **get_member_roles** operation returns the sequence of all role identifiers associated with the supplied user.

```
// from Membership

    Labels get_member_roles(
        in Session::User user
    ) raises (
        PrivacyConflict
    );
```

The **list_members** operation returns an iterator of all **User** instances associated with the **Membership**. The **max_number** argument constrains the maximum number of **User** instances to include in the returned list sequence.

```
// from Membership

    Session::UserIterator list_members(
        in long max_number,
        out Session::Users list
    ) raises (
        PrivacyConflict
    );
```

The **list_members_using** operation returns an iterator of all **User** instances associated with the **Membership** under a supplied role. The **max_number** argument constrains the maximum number of **Member** instances to include in the returned list sequence.

```
// from Membership

    Session::UserIterator list_members_using (
        in Label role,
        in long max_number,
        out Session::Users list
    ) raises (
        PrivacyConflict
    );
```

Table 3-8 Exceptions Related to Information About Members

Exception	Reason
PrivacyConflict	Raised in the case of a conflict between the invocation and the privacy policy defined under the Membership's MemberPolicy instance (refer MembershipPolicy, Privacy Constraints).

3.3.2 MembershipModel

MembershipModel is a valuetype that extends the **Model** valuetype through addition of fields containing a **MembershipPolicy** and a **Role** representing the root business role of a role hierarchy.

3.3.2.1 IDL Specification

```
valuetype MembershipModel :
    Control
supports Model
    {
    public MembershipPolicy policy;
    public CommunityFramework::Role role;
};
```

Table 3-9 MembershipModel State Table

Name	Type	Properties	Purpose
policy	MembershipPolicy	public	Defines privacy and exclusivity policy of the containing Membership.
role	Role	public	The root Role instance establishing a business role hierarchy.

3.3.3 MembershipPolicy

The **MembershipPolicy** valuetype is contained within the **CommunityModel** valuetype (and other valuetypes defined under the **CollaborationFramework**). **MembershipPolicy** defines privacy and exclusivity policy of the containing **Membership**.

3.3.3.1 IDL Specification

```
enum PrivacyPolicyValue
    {
    PUBLIC_DISCLOSURE,
    RESTRICTED_DISCLOSURE,
    PRIVATE_DISCLOSURE
```

```
};

valuetype MembershipPolicy
    {
    public PrivacyPolicyValue privacy;
    public boolean exclusive;
};
```

Table 3-10 Membership Policy State Table

Name	Type	Properties	Purpose
privacy	PrivacyPolicyValue	public	Qualification of the extent of information to be made available to clients (refer Privacy Constraints).
exclusive	boolean	public	Restricts the number of Member instances associated to a Membership to 1 for a given principal identity (refer CORBA::Current).

3.3.3.2 *Privacy Constraints*

The **MembershipPolicy** privacy attribute exposes an enumeration of privacy qualifiers. Each qualifier defines a level of information access concerning members and the roles they have. Privacy constraints refer to structural information (the association of members to a membership) and member role attribution.

Table 3-11 PrivacyPolicyValue Enumeration Table

Value	Description
PUBLIC_DISCLOSURE	Operations may return structural and member role associations to non-members.
RESTRICTED_DISCLOSURE	Operations may return structural and member role associations to members that share a common root Membership (where a root membership is derived from navigation of collection relationships to higher-level membership instances).
PRIVATE_DISCLOSURE	Operations may return structural and member role associations to members of the same Membership.

3.3.4 *Member and Recognizes Link*

Member is a type of **Privilege** link (refer Task and Session) that defines relationship between a **Membership** and a **User**. **Recognizes** is the inverse association of **Member** that associates a **Membership** with a **User**. A **Member** instance when held by a **Membership** implementation references the participating **User**. The inverse relationship, held by an implementation of **User**, contains a reference to the target **Membership**.

3.3.4.1 IDL Specification

```
valuetype Member : Session::Privilege {
    public Membership resource;
};

valuetype Recognizes : Session::Privilege {
    public Session::User resource;
    public Labels roles;
};
```

Table 3-12 Member State Table

Name	Type	Properties	Purpose
resource	Membership	public	The reference to a Membership that the User, holding this link is a member of.

Table 3-13 Recognizes State Table

Name	Type	Properties	Purpose
resource	User	public	The reference to a User that is a recognized member of the Membership holding this link.
roles	Labels	public	A sequence of role identifies managed by the Membership implementation that the membership has granted to the Member.

3.4 Roles and Role Related Policy

A business role hierarchy is defined with the **Role** valuetype. The hierarchy declares a set role instances against which members can be implicitly or explicitly associated.

3.4.1 Role

Role is a valuetype that declares the notion of a “business role” of a **User**. The state fields **label** and **note** inherited from **Control** are used to associate a role name and role description. **Role** supplements this information with an additional three state fields, **policy**, **is_abstract**, and **roles**. The **roles** field contains a sequence of role instances through which role hierarchies can be constructed. The policy field value is **RolePolicy** valuetype that qualifies the quorum, ceiling, quorum assessment and quorum policy applicable to the containing role. A **Role** can be declared as an abstract role by setting the **is_abstract** state field value to true. Declaring the role as abstract disables direct association of a **User** to the **Role** under a **Membership**. Instead, members can associate lower-level roles, thereby implicitly associating themselves with the containing roles.

Examples of business role hierarchies include the logical association of “customer” and “supplier” as roles under a parent named “signatories.” In this example, both “customer” and “supplier” would be modeled as **Role** instances with **is_abstract** set to false, and contained within a single **Role** named “signatories.” By setting the “signatories” role **is_abstract** value to true, **Members** cannot directly associate to this role. Instead, **Members** associating to either “customer” or “supplier” are implicitly granted “signatory” association.

An implementation is responsible for ensuring the consistency of quorum and ceiling values across a role hierarchy.

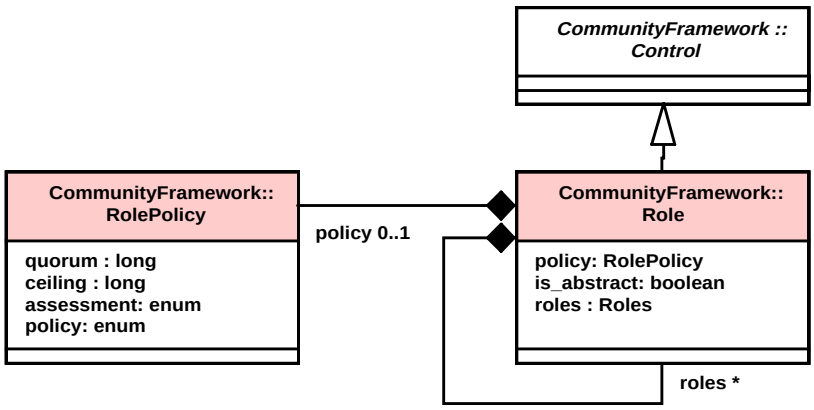


Figure 3-4 Role and Role Policy Object Model

3.4.1.1 IDL Specification

```
valuetype Role :
Control
    {
    public RolePolicy policy;
    public CommunityFramework::Roles roles;
    public boolean is_abstract;
};
```

Table 3-14 Role State Table

Name	Type	Properties	Purpose
policy	RolePolicy	public	Defines policy associated with an instance of RoleContainer or RoleElement. If null, no direct policy constraint is implied.

Table 3-14 Role State Table

roles	Roles	public	A sequence of Role instances that are considered as children relative to the containing role. Association of a Member to a child role implicitly associates the Member with all parent roles.
is_abstract	boolean	public	If true, Member instances may not be directly associated with the role under a Membership. Members may be associated implicitly through association to a non-abstract sibling.

3.4.2 RolePolicy

RolePolicy is a valuetype that defines ceiling limits and quorum policy for a particular role. The value of the quorum filed defines the minimum number of **Members** that must be associated with the role that the policy is associated with before the role can be considered to have reached quorum. The ceiling field defines the maximum number of **Members** that may be associated under the role. The policy field exposes a **RolePolicy** value that details the mechanism to quorum calculations. In the case of a null value for policy or assessment, the value shall be inferred by the parent policy. In the case of no parent policy declaration, quorum policy shall be **SIMPLE** and assessment policy shall be **LAZY** (representing the least restrictive case). The absence of a ceiling value shall indicate no limit on the number of associated members. The absence of a quorum value shall imply a quorum of 0.

3.4.2.1 IDL Specification

```
enum QuorumPolicy
    {
    SIMPLE, // default
    CONNECTED
};

enum QuorumAssessmentPolicy
    {
    STRICT,
    LAZY // default
};

valuetype RolePolicy
    {
    public long quorum;
    public long ceiling;
    public QuorumPolicy policy;
    public QuorumAssessmentPolicy assessment;
```

};

Table 3-15 RolePolicy State Table

Name	Type	Properties	Purpose
quorum	long	public	The minimum number of Members that must be associated with the role before the role can be considered to have achieved quorum.
ceiling	long	public	The maximum number of Member instances that may be associated to this role.
assessment	QuorumAssessmentPolicy	public	An enumeration used to determine the mechanism to be applied to quorum assessment. The enumeration describes STRICT and LAZY assessment policies. Under STRICT assessment, the establishment of a quorum is required before the membership is considered valid. Under LAZY assessment, the determination of quorum is based on the accumulative count of members during the lifetime of the membership. LAZY assessment introduces the possibility for the execution of optimistic processes that depend on valid quorums for finalization and commitment of results.
policy	QuorumPolicy	public	An emanation of SIMPLE or CONNECTED. When the value is SIMPLE, quorum calculation is based on number of Member instances. When the quorum policy is CONNECTED, the quorum calculation is based on the number of Member instances that reference a User that is in a connected state.

3.5 Community, Agency, LegalEntity, and Related Valuetypes

3.5.1 Community

A **Community** is a type combining a formal model of membership with the Task and Session Workspace. As a **Workspace**, a **Community** is a container of **AbstractResource** instances. As a **Membership**, a **Community** exposes a **MembershipModel** detailing the allowable business roles and group constraints applicable to associated **Users**. A new instance of **Community** may be created by passing an instance of **CommunityCriteria** to the **create** operation on **ResourceFactory**.

3.5.1.1 IDL Specification

```
interface Community :
    Session::Workspace,
    Membership
```

```
    {
};

valuetype CommunityCriteria :
    Criteria
    {
    public MembershipModel model;
};
```

Table 3-16 CommunityCriteria State Table

Name	Type	Properties	Purpose
model	MembershipModel	public	The model to associate to the Community on creation.

3.5.2 *Agency and LegalEntity*

Agency is a specialization of **Community** and **LegalEntity** that introduces the notion of organized community such as a company. As a **LegalEntity**, an **Agency** may be associated to a number of users representing roles relative to a resource derived from **LegalEntity**. **LegalEntity** is an abstract interface that defines access to implementation specific criteria such as security policy, public company information and so forth. A new instance of **Agency** may be created by passing an instance of **AgencyCriteria** to the create operation on **ResourceFactory**.

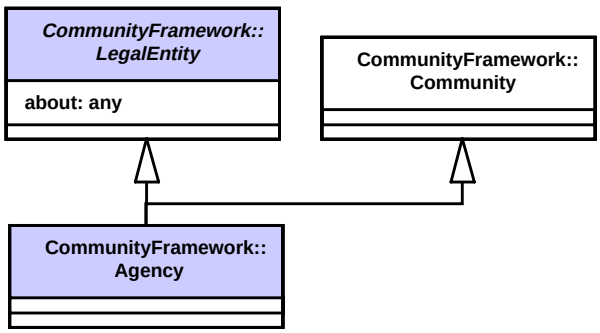


Figure 3-5 LegalEntity Object Model

3.5.2.1 *IDL Specification*

```
abstract interface LegalEntity {
    readonly attribute any about;
};

interface Agency : Community, LegalEntity { };
```

```
valuetype AgencyCriteria :
    CommunityCriteria
    {
};
```

Table 3-17 LegalEntity Attribute Table

Name	Type	Properties	Purpose
about	any	readonly	A value that may be used in an implementation specific way to expose security and other credentials towards clients.

3.6 General Utility Interfaces

3.6.1 GenericResource

GenericResource is a type of **AbstractResource** that exposes operations through which values (in the form of an any) can be attributed to the resource in an interoperable manner. Instances of **GenericResource** are created through a **ResourceFactory** using an instance of **GenericCriteria** as the criteria argument.

3.6.1.1 IDL Specification

```
exception LockedResource{
    Generic source;
};

abstract interface Generic

{
    readonly attribute any value;
    attribute boolean locked;
    attribute boolean template;
    void set_value(
        in any value
    ) raises (
        LockedResource
    );
};

interface GenericResource :
    Session::AbstractResource,
    Generic
    {
};

valuetype GenericCriteria : Criteria { };
```

3.6.2 Criteria

Concrete instances of **Criteria** may be passed as arguments to the **ResourceFactory create** operation. **Criteria** is an abstract interface supported by valuetypes that define factory creation criteria for concrete resource types defined within **Community** and **Collaboration** frameworks. A **Criteria** specialization is defined for each concrete resource type (refer ResourceFactory Required Criteria Support). **ExternalCriteria** is a special case of **Criteria** used to describe a reference to an external artifact (such as an XML document) that can be resolved in an implementation specific manner.

3.6.2.1 IDL Specification

```
valuetype Arguments CosLifeCycle::Criteria;

valuetype Criteria:
Control
{
    public Arguments values;
};

valuetype ExternalCriteria :
    Criteria
    {
    public CORBA::StringValue common;
    public CORBA::StringValue system;
};
```

Table 3-18 Criteria State Table

Name	Type	Properties	Purpose
values	Arguments	readonly	Implementation specific criteria used as supplementary information by a ResourceFactory implementation.

Table 3-19 ExternalCriteria State Table

Name	Type	Properties	Purpose
common	StringValue	public	XML public identifier.
system	StringValue	public	XML system identifier.

3.6.3 ResourceFactory

ResourceFactory is a general utility exposable by **FactoryFinder** interfaces on **Session::Workspace** and **Session::User** interfaces. **ResourceFactory** creates new instances of **AbstractResource** and derived types based on a supplied name and

Criteria. The **supporting** operation exposes a sequence of default **Criteria** instances supported by the factory. The **Criteria** types that a resource factory is required to expose and support are detailed in the following table.

Table 3-20 ResourceFactory Required Criteria Support

Module	Criteria type	Created Resource Type
CommunityFramework	CommunityCriteria	Community
	AgencyCriteria	Agency
	GenericCriteria	GenericResource
CollaborationFramework	ProcessorCriteria	Processor
		EngagementProcessor
		VoteProcessor
		CollaborationProcessor

3.6.3.1 *IDL Specification*

```
exception ResourceFactoryProblem{
    ResourceFactory source;
    CommunityFramework::Problem problem;
};

abstract interface ResourceFactory
    {

    readonly attribute CriteriaSequence supporting;

    Session::AbstractResource create(
        in CORBA::StringValue name,
        in CommunityFramework::Criteria criteria
    ) raises (
        ResourceFactoryProblem
    );
};
```

3.6.4 *Problem*

Problem is a utility valuetype that is exposed under the ResourceFactoryProblem exception within the **CommunityFramework** module, and is used to describe configuration and runtime problems within the **CollaborationFramework** that are not readily exposed as formal exceptions. Examples of the application of **Problem** instances include the description of the cause of a failure arising during a factory creation operation. Other examples from the **CollaborationFramework** include description of non-fulfillment of a constraints and documentation of non-critical problem encountered during the execution of a collaborative process.

The **Problem** valuetype contains a timestamp, a problem identifier, message and description, and a possibly empty sequence of contributing **Problem** declarations.

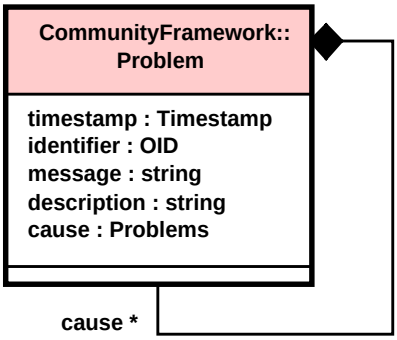


Figure 3-6 Problem Valuetype Object Model

3.6.4.1 IDL Specification

```
valuetype Problem
    {
    public Session::Timestamp timestamp;
    public Label identifier;
    public CORBA::StringValue message;
    public CORBA::StringValue description;
    public Problems cause;
};
```

Table 3-21 Problem State Table

Name	Type	Properties	Purpose
timestamp	Timestamp	public	Date and time that the problem identification occurred.
identifier	Label	public	Identifier of a labeled control.
message	StringValue	public	Short human readable message describing the problem.
description	StringValue	public	Descriptive text detailing the problem, suitable for presentation under a human interface.
cause	Problems	public	A sequence of Problem instances representing the problem cause.

3.7 UML Overview

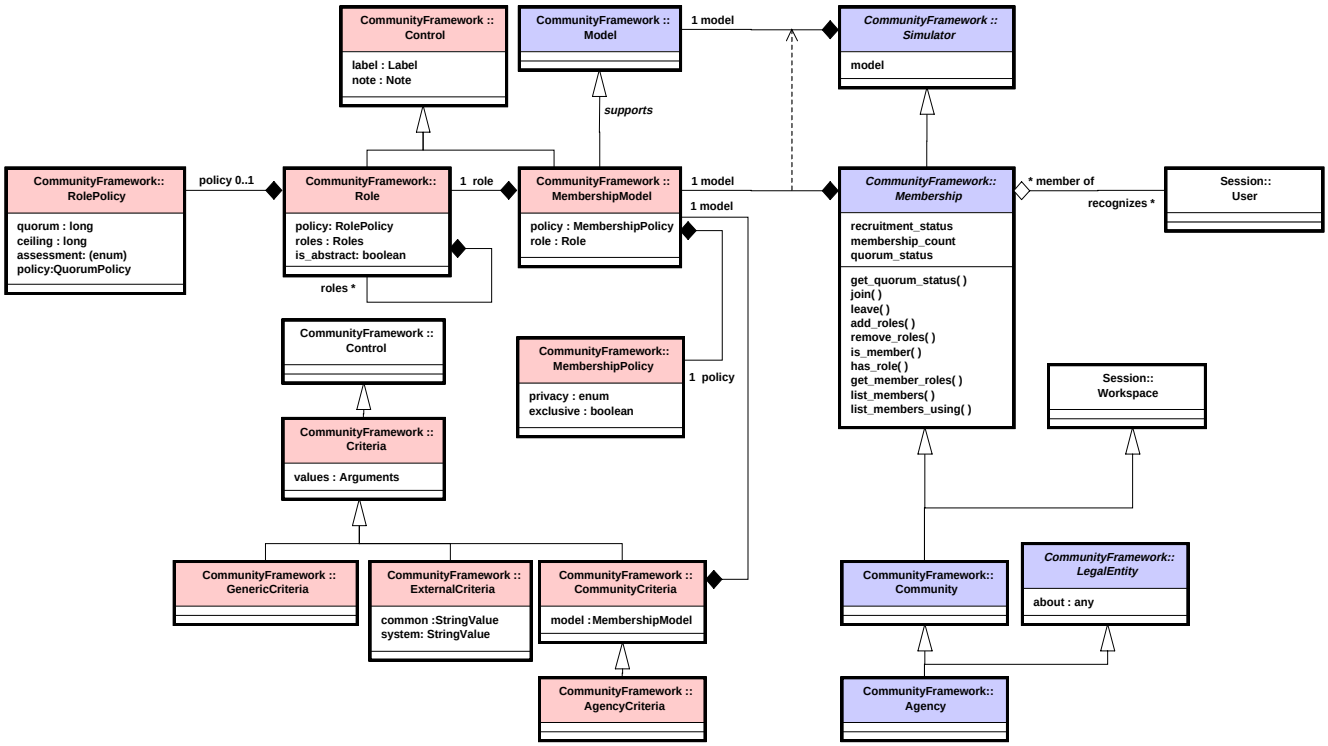


Figure 3-7 Principal Interfaces Only - does not include enumeration types, GenericResource, or ResourceFactory

3.8 CommunityFramework Complete IDL

```
#ifndef _COMMUNITY_IDL_
#define _COMMUNITY_IDL_
#include <Session.idl>
#pragma prefix "omg.org"

module CommunityFramework{

#pragma version CommunityFramework 2.0

// forward declarations

interface Agency;
interface Community;

abstract interface LegalEntity;
abstract interface Model;
abstract interface Simulator;
```

```
abstract interface Membership;
abstract interface Generic;
abstract interface ResourceFactory;

valuetype Criteria;
valuetype Control;
valuetype Role;
valuetype MembershipPolicy;
valuetype MembershipModel;
valuetype Problem;

// typedefs

valuetype Roles sequence <Role>;
valuetype Models sequence <Model>;
valuetype CriteriaSequence sequence <Criteria>;
valuetype Problems sequence <Problem>;
valuetype Note CORBA::StringValue;
valuetype Label CORBA::StringValue;
valuetype Labels sequence <Label>;

// links

valuetype Member : Session::Privilege {
    public Membership resource;
};

valuetype Recognizes : Session::Privilege {
    public Session::User resource;
    public Labels roles;
};

// structures

enum QuorumAssessmentPolicy
    {
    STRICT,
    LAZY // default
};

enum PrivacyPolicyValue
    {
    PUBLIC_DISCLOSURE,
    RESTRICTED_DISCLOSURE,
    PRIVATE_DISCLOSURE
};

enum RecruitmentStatus{
    OPEN_MEMBERSHIP, // default
    CLOSED_MEMBERSHIP
};
```

```
valuetype MembershipCount{
    public long static;
    public long active;
};

enum QuorumPolicy
    {
    SIMPLE, // default
    CONNECTED
};

enum QuorumStatus {
    QUORUM_VALID,
    QUORUM_PENDING,
    QUORUM_UNREACHABLE
};

 valuetype RoleStatus
    {
    public Label identifier;
    public MembershipCount count;
    public QuorumStatus status;
};

valuetype Problem
    {
    public Session::Timestamp timestamp;
    public Label identifier;
    public CORBA::StringValue message;
    public CORBA::StringValue description;
    public Problems cause;
};

// exceptions

exception PrivacyConflict
    {
    PrivacyPolicyValue reason;
};

exception AttemptedCeilingViolation{
    Membership source;
};

exception AttemptedExclusivityViolation{
    Membership source;
};

exception UnknownRole{
    Membership source;
```

```
};

exception UnknownMember{
    Membership source;
    Member link;
};

exception UnknownIdentifier{
    Membership source;
    Label identifier;
};

exception MembershipRejected{
    Membership source;
    string reason;
};

exception RoleAssociationConflict{
    Membership source;
    string reason;
    Label role;
};

exception CannotRemoveRole{
    Membership source;
    string reason;
    Label role;
};

exception RecruitmentConflict{
    Membership source;
    RecruitmentStatus reason;
};

exception LockedResource{
    Generic source;
};

exception ResourceFactoryProblem{
    ResourceFactory source;
    CommunityFramework::Problem problem;
};


// interfaces

abstract interface Model
    {
};

abstract interface Simulator
```

```
    {
    readonly attribute CommunityFramework::Model model;
};

valuetype MembershipPolicy
    {
    public PrivacyPolicyValue privacy;
    public boolean exclusive;
};

valuetype RolePolicy
    {
    public long quorum;
    public long ceiling;
    public QuorumPolicy policy;
    public QuorumAssessmentPolicy assessment;
};

valuetype Control
    {
    public CommunityFramework::Label label;
    public CommunityFramework::Note note;
};

valuetype Role :
        Control
    {
    public RolePolicy policy;
    public CommunityFramework::Roles roles;
    public boolean is_abstract;
};

abstract interface Membership :
    Simulator
    {

    readonly attribute RecruitmentStatus recruitment_status;
    readonly attribute MembershipCount membership_count;
    readonly attribute boolean quorum_status;

    RoleStatus get_quorum_status(
        in Label identifier // role identifier
    );

    Member join(
        in Session::User user,
        in Labels roles
    ) raises (
        AttemptedCeilingViolation,
        AttemptedExclusivityViolation,
        RecruitmentConflict,
```

```
        RoleAssociationConflict,
        MembershipRejected,
        UnknownRole
    );

    void leave(
        in CommunityFramework::Member member
    ) raises (
        RecruitmentConflict,
        UnknownMember
    );

    void add_roles(
        in CommunityFramework::Member member,
        in Labels roles
    ) raises (
        UnknownMember,
        RoleAssociationConflict,
        UnknownRole
    );

    void remove_roles(
        in CommunityFramework::Member member,
        in Labels roles
    ) raises (
        UnknownRole,
        UnknownMember,
        CannotRemoveRole
    );

    boolean is_member(
        in Session::User user
    ) raises (
        PrivacyConflict
    );

    boolean has_role(
        in Session::User user,
        in Label role
    ) raises (
        PrivacyConflict
    );

    Labels get_member_roles(
        in Session::User user
    ) raises (
        PrivacyConflict
    );

    Session::UserIterator list_members(
        in long max_number,
```

```
            out Session::Users list
        ) raises (
            PrivacyConflict
        );

        Session::UserIterator list_members_using(
            in Label role,
            in long max_number,
            out Session::Users list
        ) raises (
            PrivacyConflict
        );

};

valuetype MembershipModel :
        Control supports Model
        {
        public MembershipPolicy policy;
        public CommunityFramework::Role role;
};

valuetype Criteria :
        Control
        {
        public CosLifeCycle::Criteria values;
};

valuetype ExternalCriteria :
        Criteria
        {
        public CORBA::StringValue common;
        public CORBA::StringValue system;
};

interface Community :
        Session::Workspace,
        Membership
        {
};

valuetype CommunityCriteria  :
        Criteria
        {
        public MembershipModel model;
};

abstract interface LegalEntity {
        readonly attribute any about;
};
```

```
interface Agency : Community, LegalEntity { };

valuetype AgencyCriteria :
    CommunityCriteria
    {
};

abstract interface Generic {

    readonly attribute any value;
    attribute boolean locked;
    attribute boolean template;

    void set_value(
        in any value
    ) raises (
        LockedResource
    );
};

interface GenericResource :
    Session::AbstractResource,
    Generic
    {
};

valuetype GenericCriteria : Criteria { };

abstract interface ResourceFactory
    {

    readonly attribute CriteriaSequence supporting;

    Session::AbstractResource create(
        in CORBA::StringValue name,
        in CommunityFramework::Criteria criteria
    ) raises (
        ResourceFactoryProblem
    );
};

};

#endif // _COMMUNITY_IDL_
```

Changes to the Task and Session Specification (formal/00-05-03) *A*

A.1 BaseBusinessObject

A.1.1 BaseBusinessObject Revision

The *Task and Session Specification's* (formal/00-05-03) definition of **BaseBusinessObject** includes inheritance of the **CosNotifyComm**, **StructuredPushConsumer**, and **StructuredPushSupplier** interfaces. The semantics of **StructuredPushSupplier** implies association to a single **StructuredProxyPushConsumer**, however, the **BaseBusinessObject** interface is intended to support multiple concurrent consumers from potentially different business domains without mandating nor excluding the use of **Notification** channels as an implementation mechanism. To enable the documented behavior an explicit factory operation is required through which a **StructuredPushSupplier** reference can be exposed for a given consumer. This behavior is required to support association of multiple consumers under the **Community** and **Collaboration** interfaces.

The **CommunityFramework** requires that the definition of **BaseBusinessObject** under formal/00-05-03 be replaced with the following definition.

BaseBusinessObject

BaseBusinessObject is the abstract base class for all principal Task and Session objects. It has identity, is transactional, has a lifecycle, and is a notification supplier.

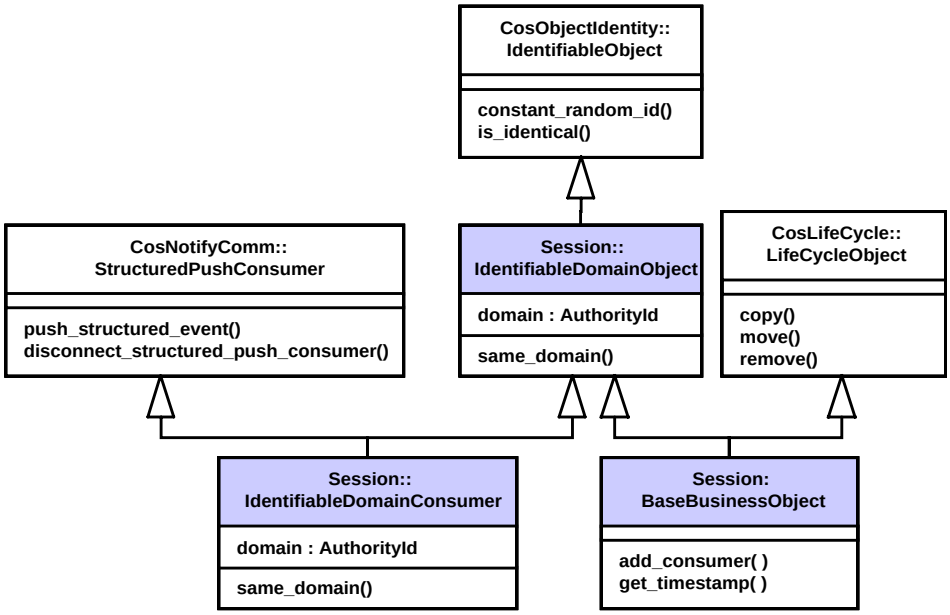


Figure A-1

IDL Specification

```
interface IdentifiableDomainConsumer :
    Session::IdentifiableDomainObject,
    CosNotifyComm::StructuredPushConsumer
    {
};

valuetype Timestamp TimeBase::UtcT ;

interface BaseBusinessObject :
    Session::IdentifiableDomainObject,
    CosLifeCycle::LifeCycleObject
    {
    CosNotifyComm::StructuredPushSupplier add_consumer(
        in IdentifiableDomainConsumer consumer
    );
    Timestamp creation( );
    Timestamp modification( );
    Timestamp access( );
};
```

The **CosNotification** service defines a **StructuredEvent** that provide a framework for the naming of an event and the association of specific properties to that event. All events specified within this facility conform to the **StructuredEvent** interface. This specification requires specific event types to provide the following properties as a part of

the **filterable_data** of the structured event header. Under the **CosNotification** specification all events are associated with a unique domain name space. This specification establishes the domain namespace "**org.omg.session**" for structured events associated with **AbstractResource** and its sub-types.

Association of an Event Consumer

IdentifiableDomainConsumer defines a **StructuredPushConsumer** callback object that can be passed to an implementation of **BaseBusinessObject** under the **add_consumer** operation. An implementation of this operation is required to establish the association of the consumer with an instance of **StructuredPushSupplier** before returning the supplier to the invoking client.

Accessing Creation, Modification, and Last Event timestamps

The operations, **creation**, **modification**, and **access** return a **Timestamp** value. The **creation** operation returns the date and time of the creation. The **modification** operation returns the last modification date and time (where modification refers to a modification of the state of a concrete derived type). The **access** operation returns the date and time a derived type was accessed.

Link

The definition of a **Link** (an association declaration) under the *Task and Session Specification* (formal/00-05-03) is in the form of a struct containing an object reference and relationship type identifier. These identifiers are declared as constants within the **Session** module. Task and Session specification of **Link** does not allow extension of associations required by the **Community** and **Collaboration** Framework specifications. Restoration of module independent extension of **Links** is possible if the **Link** struct declaration is replaced with a valuetype definition.

The **CommunityFramework** introduces the following changes to the definition of Link under Chapter 2, Section 2.5 of formal/00-05-03.

A.1.2 Links

The **Link** type is used within the Task and Session framework as an argument to operations that establish relationship dependencies between resources such as usage and containment. The **Link** type is used as an argument to the **bind**, **replace** and **release** operations of an **AbstractResource** and as a type exposed under the **expand** operation.

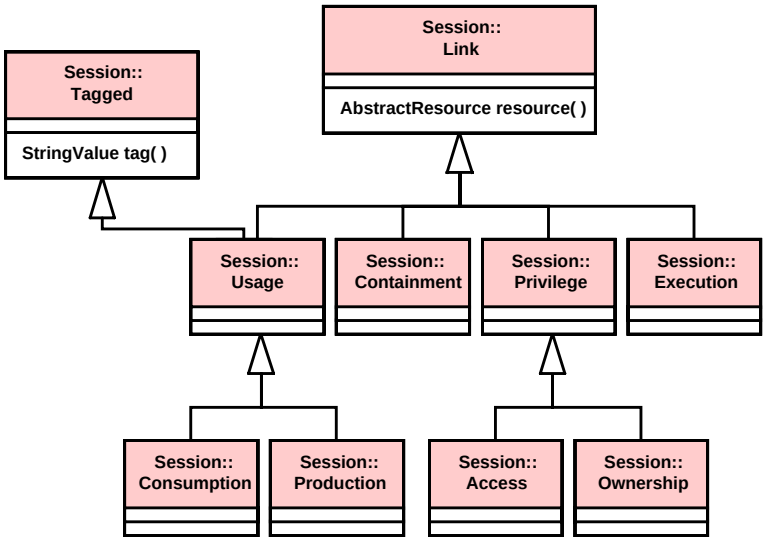


Figure A-2 Abstract Link Definitions (link families)

IDL Specification

```
abstract valuetype Link {
    AbstractResource resource( );
};

abstract interface Tagged {
    CORBA::StringValue tag( );
};

abstract valuetype Containment : Link{ };
abstract valuetype Privilege  : Link{ };
abstract valuetype Access : Privilege { };
abstract valuetype Ownership : Privilege { };
abstract valuetype Usage : Link supports Tagged { };
abstract valuetype Consumption : Usage{ };
abstract valuetype Production : Usage{ };
abstract valuetype Execution : Link{ };

valuetype Consumes : Consumption {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ConsumedBy : Consumption {
    public Task task;
    public CORBA::StringValue tag;
```

```
};

valuetype Produces : Production  {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ProducedBy : Production {
    public Task task;
    public CORBA::StringValue tag;
};

valuetype Collects : Containment {
    public AbstractResource resource;
};
valuetype CollectedBy : Containment {
    public Workspace resource;
};

valuetype ComposedOf : Collects { };
valuetype IsPartOf : CollectedBy { };

valuetype Accesses : Access {
    public Workspace resource;
};
valuetype AccessedBy  : Access {
    public User resource;
};

valuetype Administers : Accesses { };
valuetype AdministeredBy : AccessedBy { };

valuetype Owns : Ownership {
    public Task resource;
};

valuetype OwnedBy : Ownership {
    public User resource;
};
```

Link

Link represents an abstract association of one resource towards another. **Link** contains a single operation named **resource** that returns a reference to an **AbstractResource**. **Link** serves as an abstract base to a series of other abstract relationship families – Containment, Privilege, Usage, and Execution. Unless otherwise stated, a link represents a weak aggregation relationship.

```
abstract valuetype Link {
    AbstractResource resource( );
};
```

Containment

Containment is an abstract **Link** that represents the set of concrete **Link** definitions dealing with a **Collects** of **AbstractResource** by a **Workspace**, and the inverse notion of an **AbstractResource** being **CollectedBy** a **Workspace**. An instance of **Workspace** maintains a set of n **Collects** link instances, each holding a reference to exactly one collected **AbstractResource**. For every instance of **Collects**, there is an opposite **CollectedBy Link** instance maintained by an **AbstractResource** that references the collecting **Workspace**. A specialization of both **Collects** and **CollectedBy** is defined to represent a **Workspace** containing an **AbstractResource**, where an implementation wishes to express strong aggregation from the containing **Workspace** to the contained **AbstractResource**. This is defined under the **ComposedOf** and **IsPartOf** links where **ComposedOf** is a type of **Collects** and **IsPartOf** is a type of **CollectedBy**.

```
abstract valuetype Containment : Link{ };

valuetype Collects : Containment {
    public AbstractResource resource;
};
valuetype CollectedBy : Containment {
    public Workspace resource;
};

valuetype ComposedOf : Collects { };
valuetype IsPartOf : CollectedBy { };
```

Table A-1 Collects State Table

Name	Type	Properties	Purpose
resource	AbstractResource	public	A weak reference to a single AbstractResource contained by a Workspace managing this Link instance. In the case of the derived ComposedOf link, the relationship is one of strong aggregation.

Table A-2 CollectedBy State Table

Name	Type	Properties	Purpose
resource	Workspace	public	A weak reference to a single Workspace that contains the AbstractResource managing by this link instance. In the case of the derived CollectedBy link, the Workspace is a Workspace that strongly aggregates the AbstractResource that holds the Link.

Privilege

Privilege is a type of abstract link, representing a family of abstract relationships dealing with **Access** and **Ownership**. **Access** is an abstract **Link** that serves as the abstract base type for **Accesses** and **AccessedBy**. **Accesses** is a **Link** held by a **User** that references a **Workspace** – similar to a bookmark. **AccessedBy** is a **Link** held by a **Workspace** referencing a **User** that has attached a bookmark to it. The specialization of **Accesses** and **AccessedBy** named **Administers** and **AdministeredBy** provide a qualification of the access relationship whereby external clients can establish the identity of an administrating user identity. **Ownership** is an abstract link used to reflect the bi-directional relationship between a **User** and a **Task**. Every **Task** is owned by exactly one user, reflected under the **OwnerBy** link. A **User Owns** between zero and many **Tasks**.

```
abstract valuetype Privilege  : Link{ };
abstract valuetype Access : Privilege { };
abstract valuetype Ownership : Privilege { };

valuetype Accesses : Access {
    public Workspace resource;
};
valuetype AccessedBy  : Access {
    public User resource;
};

valuetype Administers : Accesses { };
valuetype AdministeredBy : AccessedBy { };

valuetype Owns : Ownership {
    public Task resource;
};

valuetype OwnedBy : Ownership {
    public User resource;
};
```

Table A-3 Accesses State Table

Name	Type	Properties	Purpose
resource	Workspace	public	A weak reference to a single Workspace held by a User, representing a bookmark of a Workspace by a User. A specialization of Access named Administers qualifies the Workspace as a Workspace that the holding user has administrative responsibility for.

Table A-4 AccessedBy State Table

Name	Type	Properties	Purpose
resource	User	public	A weak reference to a single User that is maintaining a bookmark reference to the Workspace holding this link, A specialization of AccessedBy named AdministeredBy qualifies the User as an administrator of the Workspace.

Table A-5 Owns State Table

Name	Type	Properties	Purpose
resource	Task	public	A strong aggregation reference to a single Task held by a User, representing a user's unit of Work.

Table A-6 OwnedBy State Table

Name	Type	Properties	Purpose
resource	User	public	A weak reference to a single User that is the owner of the Task holding this link.

Usage

Usage is an abstract **Link** that captures the notions of the bi-directional relationships between a **Task** and the **AbstractResource** references that are associated through consumption and production relationships. **Usage** is an abstract base type for **Consumption** and **Production** that extends the notion of **Link** through the introduction of the tag operation. Any concrete valuetype supporting usage is required to expose a state field named **tag**. The tag value is equivalent to an argument name, facilitating the establishment of naming conventions on the resources consumed by and produced by a **Task**. **Consumption** is the abstract base for the **Link** valuetypes **Consumes** and **ConsumedBy**. **Production** is the abstract base for the **Link** valuetypes **Produces** and **ProducedBy**. **Consumes** is a **Link** held by a **Task** that references an **AbstractResource** it is consuming. The inverse of this association is the **Link ConsumedBy**, held by the consumed **AbstractResource**, referencing the **Task** that is consuming it. **Produces** is a **Link** held by a **Task** that references an **AbstractResource** it is producing. The inverse of this association is the link **ProducedBy**, held by the produced **AbstractResource**, referencing the **Task** that is producing it.

```
abstract interface Tagged {
    CORBA::StringValue tag( );
```

```
};

abstract valuetype Usage : Link supports Tagged { };
abstract valuetype Consumption : Usage{ };
abstract valuetype Production : Usage{ };

valuetype Consumes : Consumption {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ConsumedBy : Consumption {
    public Task task;
    public CORBA::StringValue tag;
};

valuetype Produces : Production  {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ProducedBy : Production {
    public Task task;
    public CORBA::StringValue tag;
};
```

Table A-7 Consumes State Table

Name	Type	Properties	Purpose
resource	AbstractResource	public	A weak aggregation reference to a single AbstractResource consumed by the Task holding this link.
tag	StringValue	public	An application specific name attributed to the association.

Table A-8 ConsumedBy State Table

Name	Type	Properties	Purpose
resource	Task	public	A weak reference to a single Task that is consuming the AbstractResource holding this link.
tag	StringValue	public	An application specific name attributed to the association.

Table A-9 Produces State Table

Name	Type	Properties	Purpose
resource	AbstractResource	public	A weak aggregation reference to a single AbstractResource produced by the Task holding this link.
tag	StringValue	public	An application specific name attributed to the association.

Table A-10 ProducedBy State Table

Name	Type	Properties	Purpose
resource	Task	public	A weak reference to a single Task that is producing the AbstractResource holding this link.
tag	StringValue	public	An application specific name attributed to the association.

Execution

The abstract link **Execution** is defined under the **Session** module. It represents the abstract family of relationships between a processor and **Task**. The definition of concrete associations between a **Task** and the processing source is implementation dependent.

```
abstract valuetype Execution : Link{ };
```

General Comments

The **Link** type is a generalized utility that enables an **AbstractResource**, **User**, **Task**, or **Workspace** to declare a dependency which is exposed directly under the expand operation on **AbstractResource**, and indirectly through related list operations.

The **Link** type is provided as a means through which the type and subject resource of a dependency may be declared by the resource raising the dependency to the target. Declaration of dependency between resources enables referential integrity between resources irrespective of technology or administrative domain boundaries. Declaration, modification and retraction of dependencies are achieved through invocation of the **bind**, **release** and **replace** operations on **AbstractResource**.

A.2 AbstractResource

Modification of the **AbstractResource** interface is required by the Community Framework in relation to the management of exposed **Link** instances. Section 2.2.6 AbstractResource of formal/00-05-03 – subsection “Get Resource Tree by Link Kind” shall be replaced with the following sections “Get Resource Tree by Link Kind” and “Count Operation.”

A.2.1 Get Resource Tree by Link Kind

This operation asks an **AbstractResource** to return a set of resources linked to it by a specific relationship. Objects returned are, or are created as, **AbstractResource** instances. This operation may be used by desktop managers to present object relationship graphs.

```
LinkIterator expand (
        in CORBA::TypeCode type,
        in long max_number,
        out Links seq
    );
```

Table A-11 Expand Argument List

Argument	Description
type	The CORBA::TypeCode referencing a type derived from Link, passed under the type argument qualifies the link selection constraint relative to its most derived type. Any link that is derived from the type identified by the **type** argument is a candidate to include in the returned set of links.
max_number	The maximum number of elements to be included in the seq of exposed Link instances.
seq	A sequence of Link instances.
iterator	An iterator of Link instances.

Count Operation

This operation returns the number of **Links** held by an **AbstractResource** corresponding to a given **TypeCode** filter. **Filter** arguments are based on the same filtering model as applied under the **expand** operation.

```
short count(
        in CORBA::TypeCode type
    );
```

A.3 Session Module Revisions

There are several occurrences within the **Task and Session Specification** of exception, enumeration and struct declarations that are defined with the scope of object interfaces. This approach complicates access to these type declarations by external modules. Resolution of the problem can be readily achieved by moving the respective declarations from interface to module level as recommended under the following IDL updates.

EDITORIAL CHANGE: Section 2.2.6 of formal/00-05-03 – move following exception declarations within AbstractResource interface IDL to module level.

```
exception ResourceUnavailable{ };
exception ProcessorConflict{ };
exception SemanticConflict{ };
```

EDITORIAL CHANGE: Section 2.2.8 of formal/00-05-03 – move following declarations within User interface IDL to module level.

```
enum connect_state {connected, disconnected};
exception AlreadyConnected {};
exception NotConnected {};
```

EDITORIAL CHANGE: Section 2.2.12 of formal/00-05-03 – move following declarations within Task interface IDL to module level.

```
exception CannotStart {};
exception AlreadyRunning {};
exception CannotSuspend {};
exception CurrentlySuspended {};
exception CannotStop {};
exception NotRunning {};
enum task_state {
    open, not_running, notstarted, running,
    suspended, terminated, completed, closed };
```

The formal/00-05-03 Task and Session IDL does not contain a pragma version declaration. In order to distinguish version modification based on the changes proposed here, a pragma version of 2.0 is recommended. In addition, the non-IDL statement **#pragma javaPackage "org.omg"** shall be removed.

EDITORIAL CHANGE: Replace section 2.5 of formal/00-05-02 with the following IDL.

```
// Task and Session - Session.idl
#ifndef _SESSION_
#define _SESSION_

#include <CosLifeCycle.idl>
#include <CosObjectIdentity.idl>
#include <CosCollection.idl>
#include <NamingAuthority.idl>
#include <CosNotifyComm.idl>
```

```
#include <CosPropertyService.idl>
#include <TimeBase.idl>
#include <orb.idl>

#pragma prefix "omg.org"

module Session {

#pragma version Session 2.0

interface AbstractResource;
interface Task;
interface Workspace;
interface AbstractPerson;
interface User;
interface Message;
interface Desktop;

abstract valuetype Link;

// sequence definitions

typedef sequence<Session::AbstractResource>AbstractResources;
typedef sequence<Session::Task>Tasks;
typedef sequence<Session::Message>Messages;
typedef sequence<Session::User>Users;
typedef sequence<Session::Workspace>Workspaces;
typedef sequence<Session::Link>Links;

// iterator definitions

interface AbstractResourceIterator : CosCollection :: Iterator { };
interface TaskIterator : CosCollection :: Iterator { };
interface MessageIterator : CosCollection :: Iterator { };
interface WorkspaceIterator : CosCollection :: Iterator { };
interface UserIterator : CosCollection :: Iterator { };
interface LinkIterator : CosCollection :: Iterator { };

abstract interface Tagged {
    CORBA::StringValue tag( );
};

abstract valuetype Link {
    AbstractResource resource( );
};

abstract valuetype Containment : Link{ };
abstract valuetype Privilege  : Link{ };
abstract valuetype Access : Privilege { };
abstract valuetype Ownership : Privilege { };
abstract valuetype Usage : Link supports Tagged { };
```

```
abstract valuetype Consumption : Usage{ };
abstract valuetype Production : Usage{ };
abstract valuetype Execution : Link{ };

// concrete links

valuetype Consumes : Consumption {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ConsumedBy : Consumption {
    public Task resource;
    public CORBA::StringValue tag;
};

valuetype Produces : Production  {
    public AbstractResource resource;
    public CORBA::StringValue tag;
};
valuetype ProducedBy : Production {
    public Task resource;
    public CORBA::StringValue tag;
};

valuetype Collects : Containment {
    public AbstractResource resource;
};
valuetype CollectedBy : Containment {
    public Workspace resource;
};

valuetype ComposedOf : Collects { };
valuetype IsPartOf : CollectedBy { };

valuetype Accesses : Access {
    public Workspace resource;
};
valuetype AccessedBy  : Access {
    public User resource;
};

valuetype Administers : Accesses { };
valuetype AdministeredBy : AccessedBy { };

valuetype Owns : Ownership {
    public Task resource;
};

valuetype OwnedBy : Ownership {
    public User resource;
};
```

```
// interfaces

interface IdentifiableDomainObject :
    CosObjectIdentity::IdentifiableObject
    {
    readonly attribute NamingAuthority::AuthorityId domain;
    boolean same_domain(
        in IdentifiableDomainObject other_object
    );
};


interface IdentifiableDomainConsumer :
    Session::IdentifiableDomainObject,
    CosNotifyComm::StructuredPushConsumer
    {
};

valuetype Timestamp TimeBase::UtcT ;

interface BaseBusinessObject :
    IdentifiableDomainObject,
    CosLifeCycle::LifeCycleObject
    {
    CosNotifyComm::StructuredPushSupplier add_consumer(
        in IdentifiableDomainConsumer consumer
    );
    Timestamp creation( );
    Timestamp modification( );
    Timestamp access( );
};

exception ResourceUnavailable{ };
exception ProcessorConflict{ };
exception SemanticConflict{ };

interface AbstractResource :
    BaseBusinessObject {

    attribute string name;
    readonly attribute TypeCode resourceKind;

    void bind(
        in Link link
    ) raises (
        ResourceUnavailable,
        ProcessorConflict,
        SemanticConflict
    );
```

```
        void replace(
            in Link old,
            in Link new
        ) raises (
            ResourceUnavailable,
            ProcessorConflict,
            SemanticConflict
        );

        void release(
            in Link link
        );

        void list_contained (
            in long max_number,
            out Session::Workspaces workspaces,
            out WorkspaceIterator wsit
        );

        void list_consumers (
            in long max_number,
            out Tasks tasks,
            out TaskIterator taskit
        );

        Task get_producer( );

        short count(
            in CORBA::TypeCode type
        );

        LinkIterator expand (
            in CORBA::TypeCode type,
            in long max_number,
            out Links seq
        );

    };

    interface AbstractPerson :
        CosPropertyService::PropertySetDef
        {
    };

    enum connect_state {
        connected,
        disconnected
    };

    exception AlreadyConnected {};
    exception NotConnected {};
```

```
interface User  :
    AbstractResource,
    AbstractPerson,
    CosLifeCycle::FactoryFinder
    {

    readonly attribute connect_state connectstate;

    void connect(
    ) raises (
        AlreadyConnected
    );

    void disconnect(
    ) raises (
        NotConnected
    );

    void enqueue_message (
        in Message new_message
    );

    void dequeue_message (
        in Message message
    );

    void list_messages(
        in long max_number,
        out Messages messages,
        out MessageIterator messageit
       );

    Task create_task (
        in string name,
        in AbstractResource process,
        in AbstractResource data
    );

    void list_tasks (
        in long max_number,
        out Tasks tasks,
        out TaskIterator taskit
    );

    Desktop get_desktop ( );

    Workspace create_workspace (
        in string name,
        in Users accesslist
    );
```

```
        void list_workspaces (
            in long max_number,
            out Session::Workspaces workspaces,
            out WorkspaceIterator wsit
        );
};

interface Message : AbstractResource {
        attribute any message_id;
        attribute any message;
};

interface MessageFactory{
        Message create(
            in any message_id,
            in any message
        );
};

interface Workspace :
        AbstractResource,
        CosLifeCycle::FactoryFinder
        {
        void add_contains_resource(
            in AbstractResource resource
        );

        void remove_contains_resource(
            in AbstractResource resource
        );

        Workspace create_subworkspace (
            in string name,
            in Users accesslist
        );

        void list_resources_by_type(
            in TypeCode resourcetype,
            in long max_number,
            out AbstractResources resources,
            out AbstractResourceIterator resourceit
        );
};

interface Desktop:Workspace {

        void set_belongs_to(
            in User user
        );
```

```
    User belongs_to();
};

exception CannotStart {};
exception AlreadyRunning {};
exception CannotSuspend {};
exception CurrentlySuspended {};
exception CannotStop {};
exception NotRunning {};

enum task_state {
   open, not_running, notstarted, running,
   suspended, terminated, completed, closed
};

interface Task :
    AbstractResource
    {

    attribute string description;

    task_state get_state( );

    User owned_by();
    void set_owned_by (
        in User new_task_owner
    );

    void add_consumed(
        in AbstractResource resource,
        in string tag
    );
    void remove_consumed(
        in AbstractResource resource
    );
    void list_consumed (
        in long max_number,
        out AbstractResources resources,
        out AbstractResourceIterator resourceit,
        out LinkIterator linkit
    );

    void add_produced(
        in AbstractResource resource,
        in string tag
    );
    void remove_produced(
        in AbstractResource resource
    );
    void list_produced (
        in long max_number,
```

```
            out AbstractResources resources,
            out AbstractResourceIterator resourceit,
            out LinkIterator linkit
        );

        void set_processor(
            in Session::AbstractResource processor
        ) raises (
            ProcessorConflict
        );
        AbstractResource get_processor( );

        void start (
        ) raises (
            CannotStart,
            AlreadyRunning
        );
        void suspend (
        ) raises (
            CannotSuspend,
            CurrentlySuspended
        );
        void stop (
        ) raises (
            CannotStop,
            NotRunning
        );
};
};

#endif /* _SESSION_ */
```

Complete OMG IDL B

B.1 CollaborationFramework Complete IDL

```
#ifndef _COLLABORATION_IDL_
#define _COLLABORATION_IDL_
#include <CommunityFramework.idl>
#pragma prefix "omg.org"

module CollaborationFramework{

#pragma version CollaborationFramework 2.0

//  forward declarations

abstract valuetype Action;
abstract valuetype Transitional;
abstract valuetype Guard;
abstract valuetype Proof;
abstract valuetype Evidence;
abstract valuetype UsageDescriptor;

valuetype State;
valuetype Initialization;
valuetype Trigger;
valuetype Transition;
valuetype SimpleTransition;
valuetype LocalTransition;
valuetype TerminalTransition;
valuetype CompoundTransition;
valuetype Referral;

abstract interface Slave;
abstract interface Master;
```

```
abstract interface Collaboration;
abstract interface Engagement;
abstract interface Vote;
abstract interface Directive;

interface Encounter;
interface Processor;
interface VoteProcessor;
interface EngagementProcessor;
interface CollaborationProcessor;

// typedefs

valuetype States sequence <State> ;
valuetype Triggers sequence <Trigger> ;
valuetype Initializations sequence <Initialization> ;
valuetype UsageDescriptors sequence <UsageDescriptor> ;
valuetype Slaves sequence <Slave> ;
valuetype Directives sequence <Directive>;
valuetype Label CommunityFramework::Label;
valuetype ProcessorState Session::task_state;
valuetype ResultID unsigned long ;
valuetype TypeCode CORBA::TypeCode;
valuetype ResultClass boolean;

// structures

valuetype Duration {
    public TimeBase::TimeT value;
};

struct VoteCeiling{
    short numerator;
    short denominator;
};

enum VotePolicy{
    AFFERMATIVE_MAJORITY,
    NON_ABSTAINING_MAJORITY
};

abstract valuetype Proof {};
abstract valuetype Evidence {};

enum VoteDescriptor{
    NO,
    YES,
    ABSTAIN
};

valuetype VoteStatement :
```

```
    Evidence
    {
    public VoteDescriptor vote;
};

valuetype VoteReceipt :
    Proof
    {
    public Session::Timestamp timestamp;
    public VoteStatement statement;
};

valuetype VoteCount :
    Proof
    {
    public Session::Timestamp timestamp;
    public long yes;
    public long no;
    public long abstain;
};

valuetype Timeout{
    public Label identifier;
    public Session::Timestamp timestamp;
};

valuetype TimeoutSequence sequence <Timeout> ;

enum TriggerMode{
    INITIATOR,
    RESPONDENT,
    PARTICIPANT
};

valuetype Completion
    {
    public ResultClass result;
    public ResultID code;
};

valuetype StateDescriptor
    {
    public ProcessorState state;
    public CollaborationFramework::Completion completion;
    public CommunityFramework::Problems problems;
};

// exceptions

exception InvalidTrigger{
    CommunityFramework::Problem problem;
```

```
    Label identifier;
};

exception ApplyFailure{
    CommunityFramework::Problem problem;
    Label identifier;
};

exception InitializationFailure{
    CommunityFramework::Problem problem;
    Label identifier;
};

exception EngagementProblem{
    CollaborationFramework::Evidence evidence;
    CommunityFramework::Problem problem;
};

interface SlaveIterator : CosCollection :: Iterator { };

// coordination link

abstract valuetype Coordination : Session::Execution{ };

valuetype Monitors : Coordination {
    public Processor resource;
};

valuetype Coordinates : Monitors {};

valuetype CoordinatedBy : Coordination {
    public Session::Task resource;
};

// management link

abstract valuetype Management : Session::Link{ };

valuetype Controls : Management {
    public Slave resource;
};

valuetype ControlledBy : Management {
    public Master resource;
};

/**
Encounter
*/

interface Encounter :
```

```
    Session::Task,
    CommunityFramework::Membership
    {
};

valuetype EncounterCriteria :
    CommunityFramework::Criteria
    {
    public CommunityFramework::MembershipModel model;
};

/*
ProcessorModel
*/

abstract valuetype UsageDescriptor { };

valuetype InputDescriptor :
    UsageDescriptor
    {
    public string tag;
    public boolean required;
    public TypeCode type;
};

valuetype OutputDescriptor :
    UsageDescriptor
    {
    public string tag;
    public TypeCode type;
};

valuetype ProcessorModel :
    CommunityFramework::Control
    supports CommunityFramework::Model
    {
    public UsageDescriptors usage;
};

/**
Master, Slave and Processor.
*/

abstract interface Master {
    SlaveIterator slaves (
        in long max_number,
        out Slaves slaves
    );
};

abstract interface Slave {
```

```
        readonly attribute CollaborationFramework::Master master;
};

abstract interface Processor :
        Session::AbstractResource,
        CommunityFramework::Simulator,
        Master, Slave
        {

        readonly attribute StateDescriptor state;

        Session::Task coordinator(
        ) raises (
                Session::ResourceUnavailable
        );

        CommunityFramework::Problems verify( );

        void start (
        ) raises (
                Session::CannotStart,
                Session::AlreadyRunning
        );
        void suspend (
        ) raises (
                Session::CannotSuspend,
                Session::CurrentlySuspended
        );
        void stop (
        ) raises (
                Session::CannotStop,
                Session::NotRunning
        );
};

valuetype ProcessorCriteria :
        CommunityFramework::Criteria
        {
        public ProcessorModel model;
};

/**
Engagement
*/

abstract interface Engagement
        {
        Proof engage(
                in CollaborationFramework::Evidence evidence
        ) raises (
                EngagementProblem
```

```
    );
};

interface EngagementProcessor :
    Engagement,
    Processor
    {
};

 valuetype EngagementModel :
    ProcessorModel
    {
    public CommunityFramework::Role role;
    public Duration lifetime;
    public boolean unilateral;
};

/**
Vote.
*/

abstract interface Vote
    {
    readonly attribute VoteCount vcount;

    VoteReceipt vote(
        in VoteDescriptor value
    );
};

interface VoteProcessor :
    Vote,
    Processor
    {
};

    valuetype VoteModel :
    ProcessorModel
    {
    public VoteCeiling ceiling;
    public VotePolicy policy;
    public boolean single;
    public Duration lifetime;
};

/**
Collaboration
*/

// directive
```

```
abstract interface Directive {};

valuetype Duplicate
    supports Directive
    {
    public Label source;
    public Label target;
    public boolean invert;
};

valuetype Move
    supports Directive
    {
    public Label source;
    public Label target;
    public boolean invert;
};

     valuetype Remove
    supports Directive
    {
    public Label source;
};

valuetype Constructor
    supports Directive
    {
    public Label target;
    public CommunityFramework::Criteria criteria;
};

// apply arguments

valuetype ApplyArgument
    {
    public CollaborationFramework::Label label;
    public Session::AbstractResource value;
};

valuetype ApplyArguments sequence <ApplyArgument> ;

// collaboration

abstract interface Collaboration
    {

    readonly attribute Label active_state;
    readonly attribute TimeoutSequence timeout_list;

    void apply(
        in Label identifier
```

```
        ) raises (
            InvalidTrigger,
            ApplyFailure
        );

        void apply_arguments(
            in Label identifier,
            in ApplyArguments args
        ) raises (
            InvalidTrigger,
            ApplyFailure
        );
    };

interface CollaborationProcessor :
    Collaboration,
    Processor
    {
};

/**
Collaboration controls
*/

valuetype State :
    CommunityFramework::Control
    {
    public CollaborationFramework::Triggers triggers;
    public CollaborationFramework::States states;
};

abstract valuetype Guard {};

valuetype Clock :
    Guard
    {
    public Duration timeout;
};

valuetype Launch :
    Guard
    {
    public TriggerMode mode;
    public CommunityFramework::Role role;
};

valuetype Trigger :
    CommunityFramework::Control
    {
    public long priority;
    public CollaborationFramework::Guard guard;
```

```
        public CollaborationFramework::Directives directives; // precondition
        public CollaborationFramework::Action action;
};

abstract valuetype Action { };

abstract valuetype Transitional { };

valuetype Transition :
        Action
        {
        public CollaborationFramework::Transitional transitional;
        public UsageDescriptors usage;
};

valuetype Initialization :
        Transitional
        {
};

valuetype SimpleTransition :
        Transitional
        {
        public State target;
};

valuetype LocalTransition :
        Transitional
        {
        public boolean reset;
};

valuetype TerminalTransition :
        Transitional
        {
        public Completion result;
};

valuetype Referral :
        Action
        {
        public CollaborationFramework::Action action;
        public CollaborationFramework::Directives directives;
};

valuetype Map
        {
        public ResultClass class;
        public ResultID code;
        public CollaborationFramework::Directives directives;
        public CollaborationFramework::Action action;
```

```
};

valuetype Mapping sequence <Map> ;

valuetype CompoundTransition :
    Action
    {
    public CommunityFramework::Criteria criteria;
    public CollaborationFramework::Mapping mapping;
};

valuetype CollaborationModel :
    ProcessorModel
    {
    public CommunityFramework::Role role;
    public CollaborationFramework::State state;
 };
};

#endif // _COLLABORATION_IDL_
```

B.2 CommunityFramework Complete IDL

```
#ifndef _COMMUNITY_IDL_
#define _COMMUNITY_IDL_
#include <Session.idl>
#pragma prefix "omg.org"

module CommunityFramework{

#pragma version CommunityFramework 2.0

// forward declarations

interface Agency;
interface Community;

abstract interface LegalEntity;
abstract interface Model;
abstract interface Simulator;
abstract interface Membership;
abstract interface Generic;
abstract interface ResourceFactory;

valuetype Criteria;
valuetype Control;
valuetype Role;
valuetype MembershipPolicy;
valuetype MembershipModel;
valuetype Problem;
```

```
// typedefs

valuetype Roles sequence <Role>;
valuetype Models sequence <Model>;
valuetype CriteriaSequence sequence <Criteria>;
valuetype Problems sequence <Problem>;
valuetype Note CORBA::StringValue;
valuetype Label CORBA::StringValue;
valuetype Labels sequence <Label>;

// links

valuetype Member : Session::Privilege {
    public Membership resource;
};

valuetype Recognizes : Session::Privilege {
    public Session::User resource;
    public Labels roles;
};

// structures

enum QuorumAssessmentPolicy
    {
    STRICT,
    LAZY // default
};

enum PrivacyPolicyValue
    {
    PUBLIC_DISCLOSURE,
    RESTRICTED_DISCLOSURE,
    PRIVATE_DISCLOSURE
};

enum RecruitmentStatus{
    OPEN_MEMBERSHIP, // default
    CLOSED_MEMBERSHIP
};

valuetype MembershipCount{
    public long static;
    public long active;
};

enum QuorumPolicy
    {
    SIMPLE, // default
    CONNECTED
```

```
};

enum QuorumStatus {
    QUORUM_VALID,
    QUORUM_PENDING,
    QUORUM_UNREACHABLE
};

 valuetype RoleStatus
    {
    public Label identifier;
    public MembershipCount count;
    public QuorumStatus status;
};

valuetype Problem
    {
    public Session::Timestamp timestamp;
    public Label identifier;
    public CORBA::StringValue message;
    public CORBA::StringValue description;
    public Problems cause;
};

// exceptions

exception PrivacyConflict
    {
    PrivacyPolicyValue reason;
};

exception AttemptedCeilingViolation{
    Membership source;
};

exception AttemptedExclusivityViolation{
    Membership source;
};

exception UnknownRole{
    Membership source;
};

exception UnknownMember{
    Membership source;
    Member link;
};

exception UnknownIdentifier{
    Membership source;
    Label identifier;
```

```
};

exception MembershipRejected{
    Membership source;
    string reason;
};

exception RoleAssociationConflict{
    Membership source;
    string reason;
    Label role;
};

exception CannotRemoveRole{
    Membership source;
    string reason;
    Label role;
};

exception RecruitmentConflict{
    Membership source;
    RecruitmentStatus reason;
};

exception LockedResource{
    Generic source;
};

exception ResourceFactoryProblem{
    ResourceFactory source;
    CommunityFramework::Problem problem;
};


// interfaces

abstract interface Model
    {
};

abstract interface Simulator
    {
    readonly attribute CommunityFramework::Model model;
};

valuetype MembershipPolicy
    {
    public PrivacyPolicyValue privacy;
    public boolean exclusive;
};
```

```
valuetype RolePolicy
    {
    public long quorum;
    public long ceiling;
    public QuorumPolicy policy;
    public QuorumAssessmentPolicy assessment;
};

valuetype Control
    {
    public CommunityFramework::Label label;
    public CommunityFramework::Note note;
};

valuetype Role :
        Control
    {
    public RolePolicy policy;
    public CommunityFramework::Roles roles;
    public boolean is_abstract;
};

abstract interface Membership :
    Simulator
    {

    readonly attribute RecruitmentStatus recruitment_status;
    readonly attribute MembershipCount membership_count;
    readonly attribute boolean quorum_status;

    RoleStatus get_quorum_status(
        in Label identifier // role identifier
    );

    Member join(
        in Session::User user,
        in Labels roles
    ) raises (
        AttemptedCeilingViolation,
        AttemptedExclusivityViolation,
        RecruitmentConflict,
        RoleAssociationConflict,
        MembershipRejected,
        UnknownRole
    );

    void leave(
        in CommunityFramework::Member member
    ) raises (
        RecruitmentConflict,
        UnknownMember
```

```
);

void add_roles(
    in CommunityFramework::Member member,
    in Labels roles
) raises (
    UnknownMember,
    RoleAssociationConflict,
    UnknownRole
);

void remove_roles(
    in CommunityFramework::Member member,
    in Labels roles
) raises (
    UnknownRole,
    UnknownMember,
    CannotRemoveRole
);

boolean is_member(
    in Session::User user
) raises (
    PrivacyConflict
);

boolean has_role(
    in Session::User user,
    in Label role
) raises (
    PrivacyConflict
);

Labels get_member_roles(
    in Session::User user
) raises (
    PrivacyConflict
);

Session::UserIterator list_members(
    in long max_number,
    out Session::Users list
) raises (
    PrivacyConflict
);

Session::UserIterator list_members_using(
    in Label role,
    in long max_number,
    out Session::Users list
) raises (
```

```
            PrivacyConflict
        );

};

valuetype MembershipModel :
        Control supports Model
        {
        public MembershipPolicy policy;
        public CommunityFramework::Role role;
};

valuetype Criteria :
        Control
        {
        public CosLifeCycle::Criteria values;
};

valuetype ExternalCriteria :
        Criteria
        {
        public CORBA::StringValue common;
        public CORBA::StringValue system;
};

interface Community :
        Session::Workspace,
        Membership
        {
};

valuetype CommunityCriteria  :
        Criteria
        {
        public MembershipModel model;
};

abstract interface LegalEntity {
        readonly attribute any about;
};

interface Agency : Community, LegalEntity { };

valuetype AgencyCriteria :
        CommunityCriteria
        {
};

abstract interface Generic {

        readonly attribute any value;
```

```
        attribute boolean locked;
        attribute boolean template;

        void set_value(
            in any value
        ) raises (
            LockedResource
        );
};

interface GenericResource :
        Session::AbstractResource,
        Generic
        {
};

valuetype GenericCriteria : Criteria { };

abstract interface ResourceFactory
        {

        readonly attribute CriteriaSequence supporting;

        Session::AbstractResource create(
            in CORBA::StringValue name,
            in CommunityFramework::Criteria criteria
        ) raises (
            ResourceFactoryProblem
        );
};

};

#endif // _COMMUNITY_IDL_
```

Index

Index