OOI CYBERINFRASTRUCTURE
NSF OCEAN OBSERVATORIES INITIATIVE

OOI – CyberInfrastructure

Architecture & Design

Logical Data Model

OV-7 PDR CANDIDATE

November 2007

NSF JOI Joint Oceanographic Institutions UCSD

Document owner: OOI CI System Architecture Team

Document History

Date	Version	By	Description of Changes
2007-07-30	2.0	CI ADT	Conceptual Architecture Update Initial Draft
2007-10-05	3.0	CI ADT	Early working draft with new models
2007-11-12	3.1	CI ADT	Aligned with OV-2

Preamble

The set of documents named AV*, OV*, SV*, TV* are all part of the OOI CyberInfrastructure Architecture & Design (CIAD), in the structure prescribed by the DoDAF (Department of Defense Architecture Framework). Each document has a designated title, an identifier (such as AV-1) and covers a specific topic in a self-contained way. Document AV-1 provides further explanations and a summary. A glossary of the terms used in these documents and their context can be found in AV-2.

The figure below suggests an intuitive reading flow through the provided documents. Other documents will be added to the figure as they emerge during the design of the CI (for the complete set of documents see AV-1). The thick arrow suggests a reading order through the core documents (AV-1, OV-1, OV-5 and SV-1). The red rectangle highlights the current document.

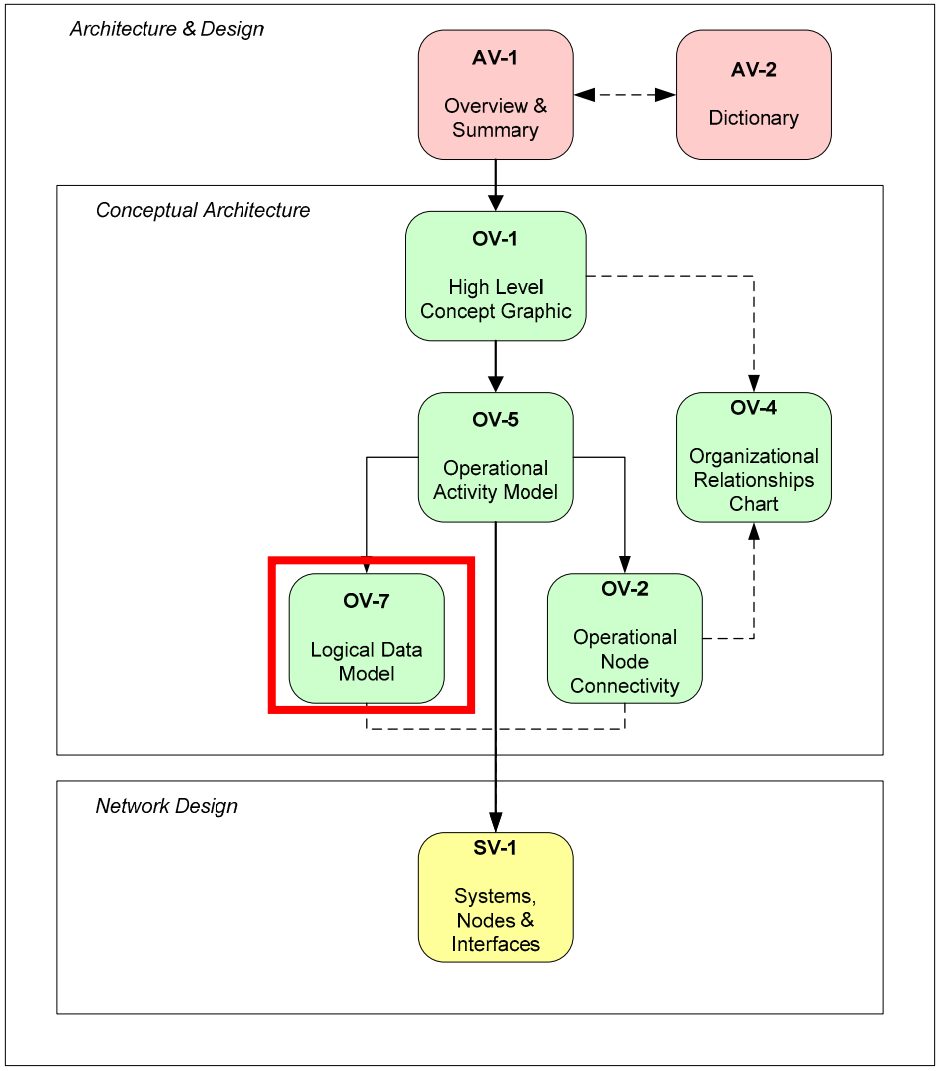

Table of Contents

OOI - CyberInfrastructure
Architecture & Design
Logical Data Model (OV-7)

1 Introduction

1.1 Product Overview

This document describes the structure of an architecture domain's system data types and the structural business process rules (defined in the architecture's OV) that govern the system data. It provides a definition of architecture domain data types, their attributes or characteristics, and their interrelationships (adapted from [DoDAF-vII 2007]).

1.2 Product Purpose and Description

The document, including the domain's system data types or entity definitions, is a key element in supporting interoperability between architectures, since these definitions may be used by other organizations to determine system data compatibility. Often, different organizations may use the same entity name to mean very different kinds of system data with different internal structure. This situation will pose significant interoperability risks, as the system data models may appear to be compatible, each having a Target Track data entity but having different and incompatible interpretations of what Target Track means.

An OV-7 may be necessary for interoperability when shared system data syntax and semantics form the basis for greater degrees of information systems interoperability, or when a shared database is the basis for integration and interoperability among business processes and, at a lower level, among systems (adapted from [DoDAF-vII 2007]).

2 Rich Service View of OOI

Rich Service[1] is an architectural pattern to describe hierarchical service-oriented systems. It allows the architect to decompose the system according to different major concerns.

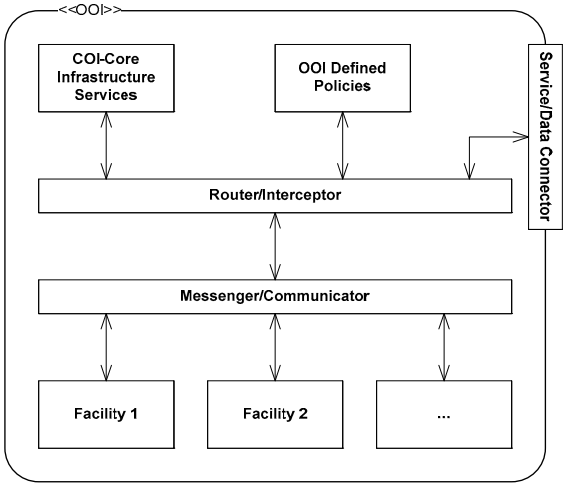

Figure 1: OOI Rich Service View

[1] An introduction to the Rich Services architectural pattern is presented across the different products to ensure self-containedness.

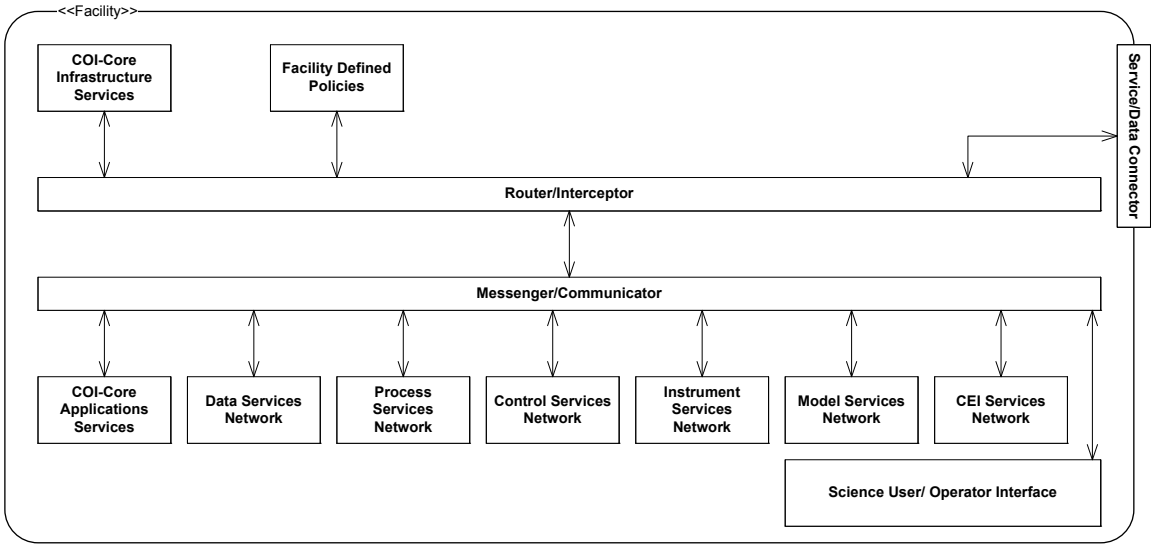

Figure 2: Facility Rich Service View

Figure 1 presents the high-level view of the OOI system based on the *Rich Service* pattern. The *Messenger/Communicator* and the *Router/Interceptor* represent a generic ESB (Enterprise Service Bus) infrastructure that allows flexible communication between the various components connected to it. The figure shows that the OOI system will be composed, at the highest level, of a set of *Facilities* that interact according to a *Policy* defined by OOI. COI-Core Infrastructure Services plus the ESB provide the infrastructure needed to enable the interaction of *Facilities* and the enforcement of the *Policies* defined.

The *Rich Service* pattern leverages features provided by typical ESB infrastructures and distinguishes between two types of components: *Rich Application Services* (RASs) and *Rich Infrastructure Services* (RISs). The first type is connected to the Messenger/Communicator layer and provides business domain services (functionalities required to deliver the results needed by the users of the system). In Figure 1, for example, *Facilities* are captured as RASs. In fact, the interactions between the various facilities that form the OOI system produce information, needed by the scientists (users of the system) to answer their scientific questions.

The *Service/Data Connector* is the interface between a *Rich Service* and its environment. It encapsulates the interactions and behavior of the components of the *Rich Service* and presents a well-defined interface to other systems.

Figure 2 shows the *Facility* RAS in more details. Because *Rich Service* is a hierarchical architectural pattern, the *Facility* has the same structure of the *OOI* Rich Service. Here, the *Policies* that are applied to

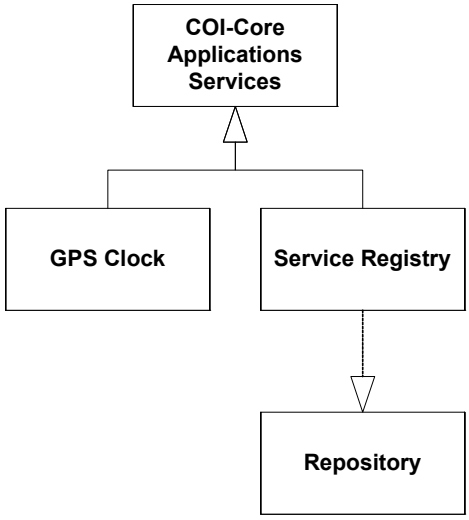

Figure 3: Example Rich Application Services

the interactions between components of the facility can be locally defined and enforced.

Each Facility provides its functionality by leveraging the services of the *Resource Services Networks* introduced in OV2. These networks are captured as RAS. Therefore, each *Network* provides a coherent set of functionalities and interacts with other *Networks* according to the specified *Policies*.

In particular, the *Facility* RAS uses the *COI-Core Application Services* and the *COI-Core infrastructure Services*. Figure 3: Example Rich Application Services presents two important instances of the COI-Core services. *Service Registry* is instantiated in the technical infrastructure by *Repositories* provided by the *Data Network*. Logically, however, this functionality is in the *COI-Core*. *COI-Core* is in charge of exposing this interface and forwarding the request to the pertinent services.

Figure 4 shows the RISs provided by *COI-Core*. The *Service/Data Connector* is represented here as a *COI-Core* service because it mediates all communications between the components of the system according to policies defined by OOI. This is a main responsibility of the COI-Core subsystem.

Infrastructure services such as *Identity Management* and *Authentication* are policies that need to be defined and applied across the whole OOI system. In general, the mechanisms for enforcing policies (*Policy Enforcement*) are provided by the *COI-Core*. *State Management* and *Logging* are provided in the generic *Interaction* framework provided by the COI.

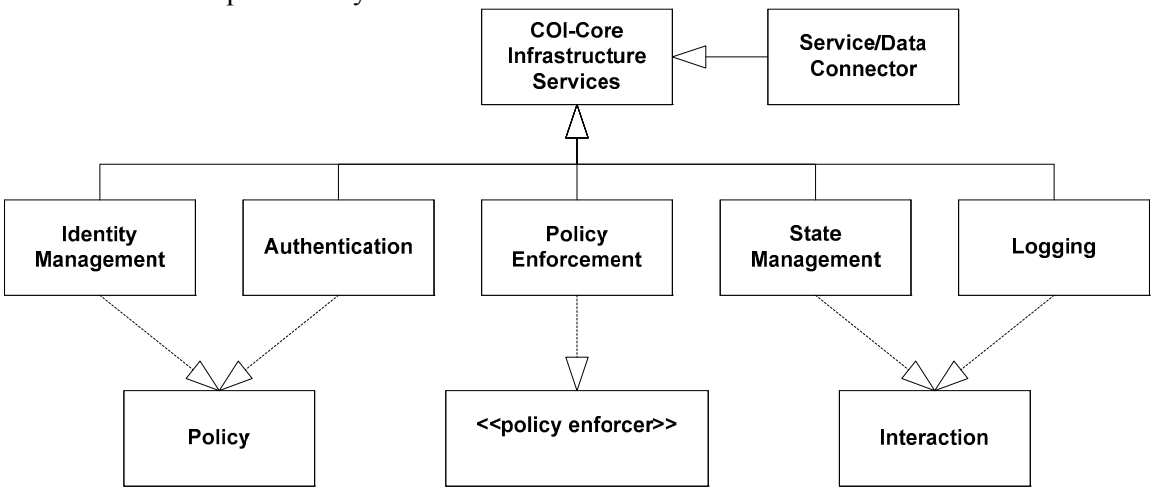

Figure 4: Rich Infrastructure Services

3 Detailed Views of the CyberInfrastructure

3.1 Overview

Exhibit 5 shows an **overview** of the main elements of the OOI CyberInfrastructure and their relationships; detailed models for these main elements are presented in the subsequent sections. Dotted boxes identify the subsystems responsible for the functionality provided by the contained elements and relations; these subsystems trace back to the operational nodes called out in OV-2. As relations can cross the boundaries between different subsystems, the figure captures the dependencies between them. The main entities in the Process Services Network are the *Computation Scheduler*, the *Process Controller*, the *Process Definition*, the *Process Plan*, and the *Process Status*.

The *Computation Scheduler* is in charge of negotiating the computation resources participating in the *Resource Setup Protocol*. From an agreed-upon *Observation Plan* the *Computation Scheduler* produces a *Process Plan* containing only the part of the *Observation Plan* concerning computational resources. A *Process Controller* uses the *Process Plan* to determine what *Process Definitions* must be provided to the *Dispatcher* in the CEI Services Network to be executed in some *Execution Engine*. During execution, the *Execution Engine* provides feedback as *Process Status* that is consumed by the *Process Controller*. If some problem in the execution is discovered, the *Process Controller* can resolve it locally, by negotiating

with the *Dispatcher*, or can involve the *Computation Scheduler* and negotiate new resources or report failure trough the *Resource Setup Protocol*.

The CEI Services Network is responsible for the execution of *Process Definitions* on *Execution Engines* that are instantiated on real *Computation Nodes*. The *Dispatcher* is the element of CEI that instantiates *Execution Engines* and *Communication Facilities*. *Execution Engines* run on *Computation Nodes*; when they are instantiated they read *Process Definitions* and execute them creating *Process Instances*. Each *Execution Engine* understands one *Definition Language* in which *Process Definitions* are encoded. *Communication Facilities* are created by the *Dispatcher* on the *Communication Infrastructure* and enable the communication of different *Process Instances*.

The *Instrument Services Network* is responsible for deploying and providing access to the various scientific *Instruments* that are used to perform measurements and experiments. Each instrument will be connected to an *Instrument Proxy*. This proxy is a Process Instance in the cyber-infrastructure and is in charge of making the data collected by the instrument available to the rest of OOI and of relaying commands received by other parts of the system to the instrument. Because it is the mediator between the physical network the instrument is connected to and the rest of the cyber-infrastructure of OOI, the *Instrument Proxy* has full control over the instrument and the enforcement of policies defined by the instrument owner. The Instrument Planner participates in OOI system-wide interactions to enforce policies over the usage of the various instruments. For example, it participates together with the Computation Scheduler and the Resource Planner in the Resource Setup Protocol, which enables scientists to obtain resources that allow them to perform observations and experiments leveraging the OOI system.

The main elements of the *Control Services Network* are the *Resource Planner* and the *Observation Plan*. The *Resource Planner* receives an *Observation Request* from the *Modeling Network* and reaches a *Service Agreement* with the *Process*, *Data* and *Instrument Services Networks* to use the resources needed to perform such observation. Once the agreement is reached (exchanging *Service Agreement Proposals* with the other participants in the *Resource Setup Protocol*), an *Observation Plan* is created. This plan describes an interaction of technical entities in the OOI that allows execution of the observation goal of the scientist. The differences between an *Observation Request* and an *Observation Plan* are twofold. First, the *Observation Request* is expressed in the language of the scientists, using the vocabulary provided by a *Science Ontology,* whereas the *Observation Plan* is expressed in terms of Processes and Communication Facilities that need to be instantiated and interconnected. Second, the *Observation Plan* is bound to physical resources that have been negotiated and obtained according to the policies defined by OOI.

The *Data Services Network* is responsible for providing the physical data storage services and the internal format of data. Various message type definitions for transporting information in the OOI system are a core element of the Data Services Network. The *Data Planner* is in charge of negotiating storage resources with the Control Services Network to ensure the availability of storage space during the observation.

Finally, the *COI-Core* is responsible for the communication, routing, and all mediation facilities of OOI. Exhibit 5 presents the main entities of this subsystem. An *Interaction Specification* captures the communication patterns between *Interaction Roles*. Roles communicate over *Communication Channels* via *Messages*. The *Interaction* is the realization of a specification and is defined by the sequence of messages over the various communication channels of the system.

3.2 The Rich Service Architecture Model

Exhibit 6 - Rich Service Model presents the details of the Rich Service architectural pattern in terms of the entities that form the OOI domain model. The main element is the *Rich Service*, which is, in turn, composed of a *Routing Interface*, a *Communication Interface*, and other *Rich Services*. There are two types of *Rich Services*: *Rich Infrastructure Services* (RISs) and *Rich Application Services* (RASs).

In the OOI CyberInfrastructure, the *Messenger/Communicator* block of the Rich Service pattern is realized by the *Communication Interface* exposing a set of *Local Queues,* which contain *Messages.* These

queues are used by *Routers* in the *Router/Interceptor* block to receive messages from and deliver messages to Rich Application Services.

The *Router/Interceptor* block exposes the *Routing Interface* to Rich Infrastructure Services. This interface is made of three entities: *Communication Setup Strategies, Communication Facilities,* and the *Communication Infrastructure*. *Communication Setup Strategies* are used to create different types of *Communication Facilities*. To perform this task, they read *Communication Channels* that specify the communication needs based on *Science Domain Properties*. The *Communication Infrastructure* is the entity that provides and governs all *Communication Facilities* in the system. It is the ultimate policy enforcer for the creation and management of communication. A *Communication Facility* is the active entity that transmits messages between *Routers* in OOI. To perform its functionality, the *Router* uses *Remote Queues* bound to *Communication Facilities*. The *Remote Queue* allows communication between routers using the *Communication Infrastructure.* The communication of each RAS is mediated by a *Router*. RISs can leverage the *Routing Interface* to apply policies to all communications of RASs. In fact, the *Router* component delegates policy decisions to RISs acting as *Policy Enforcers*.

Rich Application Services are realized by *Process Instances* that perform the computation tasks required by scientists for their observations. As mentioned before, Rich Service is a hierarchical pattern where one Rich Service can be composed of other Rich Services and a Communication Infrastructure. Therefore, *Policies* can be defined at each level of the hierarchy inside a Rich Service and are enforced by the internal RISs acting as *Policy Enforcers*.

3.3 Interaction

The models of this section capture a key contribution of the COI-Core: the notion of *Interaction* and its *Interaction Specification* data structure. Exhibit 7 shows how both an *Observation Plan* and a *Policy* are just two different ways of capturing *Interaction Specifications*. Interaction specifications are of two types: *Science Services* and *Policy*. The difference between the two is in the subjects of the *Interaction*. *Science Services* are captured in the Rich Service pattern by RIS. They capture interactions of *Process Instances* needed to perform tasks of interest for the scientist that use the system. In particular, an *Observation Plan* captures the interaction needed to perform observations that connects data streams from instruments, filtering of such data, modeling algorithms and visualization. Scientists can request the OOI system to perform an observation by creating and *Observation Request* and presenting it to the *Resource Planner* in the *Control Network*. An *Observation Request* is expressed in terms of a *Science Ontology* and has no concrete resources associated to it. The *Resource Planner* converts the request into an *Observation Plan* and by means of a *Resource Setup Protocol* obtains and binds concrete resources to it.

A *Policy* captures the *Interactions* between technical entities of the OOI system to enforce policies and manage resources. *Policies* are captured in the Rich Service pattern by RISs. In particular, we identify two types of *Policies*: *Local Policies* mandate the behavior of a single entity of the system and can be enforced by a single *Policy Enforcer*, *OOI Policies* are global and define the behavior of different entities according to the rules defined by OOI. The *Resource Setup Protocol* is an example of *OOI Policy*; it describes a protocol that *Resource Planner*, *Computation Scheduler*, and *Instrument Planner* have to execute to grant resources to *Observation Plans*. A *Service Agreement Proposal* is exchanged between the various entities until they reach an agreement.

Local Policies are associated to single resources and applied locally. In particular, they are associated to Process Instances and determine the concrete behavior of the *Interaction Roles* such instances are playing. In OOI all behaviors are defined by *Interaction Specifications*, therefore, *Interaction Roles* and the *Communication Channels* they exchange *Messages* on along with the corresponding *Message Specifications* are the structural and behavioral interfaces of each service provided by OOI. *Interaction Specifications* constrain the communication behavior of all *Interaction Roles* that appear in them; for this reason, great care must be taken in ensuring that, if *Local Policies* conflict with the communication pattern required by the *Interaction Specification* the failure of some component to perform as expected is managed consistently.

Exhibit 8 captures the main elements of a generic language to describe *Interaction Specifications*. The *Interaction Specification* is just one *Interaction Element*, therefore, it can be as simple as a *Local Action* (for example switching of a sensor in and instrument) or a complex *Composite Interaction* with *Operators* multiple *Roles*, *Communication Channel Specifications,* and *Messages*. This generic language for Interaction Specifications allows us, in particular, to use widely-used notations such as Message Sequence Chart (MSC) or the Unified Modeling Language (UML) Sequence Diagrams for writing down Interaction Specifications. Both of these languages support Operators for sequential and parallel composition, choice, and repetition; in addition, the generic language also supports a powerful operator (called join) for overlapping Interaction Specifications, i.e. Interaction Specifications that share at least one role and at least one Messages among shared roles.

Each *Interaction Element* can have *Science Domain Properties* associated to it. Such properties are expressed in a *Science Ontology* (described later in this document) and allow *Interaction Specifications* to be described in terms of properties of relevance for the scientists using the system. The *Science Domain Properties* are used to bind the abstract concept of interaction to the concrete problems of the scientist. As an example, we can consider a *Message Specification*. It describes information sent by a particular *Interaction Role*. A science domain property could allow the scientist to define that the message contains a temperature expressed in Celsius degrees. The architecture allows science properties to be expanded by plugging in new science ontologies; therefore, the interaction specification language does not contain those concepts natively but as properties expressed in a generic *Science Ontology*.

3.4 Governance

The governance of OOI is performed by applying *Policies* bound to *Resources*. Exhibit 9 shows the main elements of governance in OOI: a *Capability* in OOI is defined by the participation of a *Resource* in an *Interaction*. An *Interaction* is the enactment of an *Interaction specification*, therefore, a *Capability* implies the real execution of a task that involves a *Resource*. *Capabilities* are always made of two components: a *Domain Capability* part, emerging from a *Science Service* that *Defines* how the *Interaction* must be carried out, and a *Policy Capability* part, emerging from the *Constraints* imposed by *policy enforcers* on the *Interaction*.

Each *Resource* can have multiple *Policies* attached to it and delegates their application to *Policy Enforcers*. A *Policy Enforcer* must be in the position to constrain how a *Resource* participates in an *Interaction*; therefore, *Policy Enforcers* manage the *Resources* they apply *Policies* to.

Exhibit 9 also depicts the various *Policy Enforces*, *Resources*, and *Policy Types* of OOI. *Computation Scheduler* is one of the *Policy Enforcers* of OOI; it manages *taskable resources* (i.e. the various *Resources* that have the capability of running a *Process Instance*) and *Process Definitions*. Examples of *Policies* that can be applied by a *Computation Scheduler* are: a *Process Definition* to be loaded in an *Instrument* must be digitally signed by an authority trusted by the Institution that owns the *Instrument*, the execution of a particular *Process Instance* will be terminated if it lasts more than a specified duration. An *Instrument Proxy* is a mediator of all communication between the OOI CI and the *Instrument*, therefore, it can enforce all kind of policies on how an *Instrument* is used. For example, if the battery is low the Instrument Proxy can decrease the sample rate, or it can accept commands only if sent by an authenticated administrator defined by the institution that owns the instrument. The *Instrument Planner* is also a *Policy Enforcer* for the instrument; in fact, it is in charge of making the instrument available for the CI by instantiating the right *Instrument Proxy* and registering it in the *Instrument Process Repository*. Moreover, because the *Instrument Planner* instantiates the *Instrument Proxy*, it can also manage and apply policies to Instrument Proxies. It is also the responsibility of the *Instrument Planner* to interact with the *Resource Planner* and schedule *Instrument Resources*; it can also apply usage policies to *Instruments.*

Execution Engines manage *Process Instances*. Typical policies that can be applied are the amount of CPU cycles dedicated to a process, access to storage, memory or networking resources. A *Resource Planner* manages *Interaction Specifications* because it is in charge of transforming *Observation Requests* into *Observation Plans*. An *Observation Plan* has physical *Resources* attached to it and the corresponding

permissions to access them. Typically the *Resource Planner* can apply global *OOI Policies*, for example it initiates and participates in the *Resource Setup Protocol*. The *Communication Infrastructure* can apply policies to *Communication Facilities*. In fact, the *Communication Infrastructure* is responsible for enforcing the type of *Communication Facility* to create. *Messages* and the attached *Data* are managed by *RIS* in the Rich Service Architecture. In fact the *Router* delegates to RIS all the routing policies. *Messages* are transported over the *Communication Facility*, but all formats and characteristics of such messages are enforced by *RIS*. Finally the *Data* payload of a message can be used as a discriminator for performing the routing and special *RIS* routing policies can be configured to enforce the transformation, or any elaboration of any data payloads.

Exhibit 10 depicts how two particularly important *Policies*: *Authentication* and *Authorization* are implemented in OOI by defining the *Interaction Roles* that play together in the instantiation of those *Policies*.

3.5 Process

The Process Model (shown in Exhibit 11) captures the main elements of the OOI CI computation infrastructure; this model spans the Process Services Network and the CEI Services Network. *Process Definitions* specify, in various *Definition Languages*, the computation that *Process Instances* need to perform. Each *Process Definition* is read by one of the depicted *Execution Engines* and is executed in a *Process Instance*. Processes communicate by means of *Communication Facilities* (by the mediation of routers). The Computation Scheduler interacts with the rest of the OOI system via the Resource Setup Protocol. *Computation Scheduler* does not directly instantiate Processes or *Communication Infrastructures*. The task of configuring the computation is assigned to the *Dispatcher* in CEI. The *Computation Scheduler* produces a *Process Plan* that is interpreted by the *Process Controller*. A *Process Plan* contains the *Process Definitions* for all processes that need to be instantiated to run a particular observation plus all the information to create network connections between processes. The Plan is consumed by the *Process Controller*; it splits the *Process Plan* in the various *Process Definitions* and *Communication Channels* to be instantiated and uses various *Dispatcher*s to instantiate them. The *Dispatcher* sets up *Computation Nodes* with the appropriate *Execution Engines* that understand the given *Process Definitions* and start the *Process Instances*. The setting up of *Communication Facilities* is delegated by the *Dispatcher* to a *Communication Setup Strategy* that is provided by the *Communication Infrastructure*.

3.6 Instrument

Exhibit 12 shows the Instrument Model, focusing on the relationship between an *Instrument* – the physical device that conducts experiments and performs measurements - and its representation in the OOI CyberInfrastructure, the *Instrument Proxy*. The core concept expressed in the Instrument Model is that the communication with the *Instrument* is mediated by an *Instrument Strategy* that implements the appropriate *Instrument Dependent Communication Protocol*. The *Instrument Proxy* receives from the OOI CI *Command Messages* that are (after proper translation) relayed to the *Instrument*. Data produced by the *Instrument* is translated by the *Instrument Proxy* to three different types of messages: *Raw Data Messages* carry the scientific data gathered by the *Instrument*, *Instrument Status Messages* provide *Engineering Data* on the status of the *Instrument*, and *Processing Status Messages* provide *Engineering Data* on the status of computations that are running on the *Computation Nodes* inside the *Instrument*.
Exhibit 13 exposes the relationship between *Instruments*, the *Instrument Platform* they are attached to, and the *Translators* needed to facilitate the OOI CyberInfrastructure interaction with the specific data and commands of each *Instrument*.

3.7 Data

The Data Model of Exhibit 14 shows how all information needed by OOI is stored in *Repositories* and associated to *Metadata* expressed in terms of *Science Domain Properties* using a *Science Ontology* vocabulary. Moreover, it outlines two types of *Communication Facilities*: *Queue* and *Topic*. The *Data Plan-*

ner participates in the *Resource Setup Protocol* and receives *Service Agreement Proposals*. Using this protocol the *Data Planner* allocates resources in the *Data Product Repository* to carry out observations.

In the OOI CI, all observations requested from scientists, the descriptions of *Instruments*, and the scientific data obtained, are expressed in terms of a generic *Science Ontology*. Different *Instruments* and *Processes* may require different *Science Ontologies* (see Exhibit 15), which are, therefore, pluggable and identified by a *Unique Name*. An extensible ontology for describing *Observation Requests* to the OOI CI is depicted.

4 Exhibits

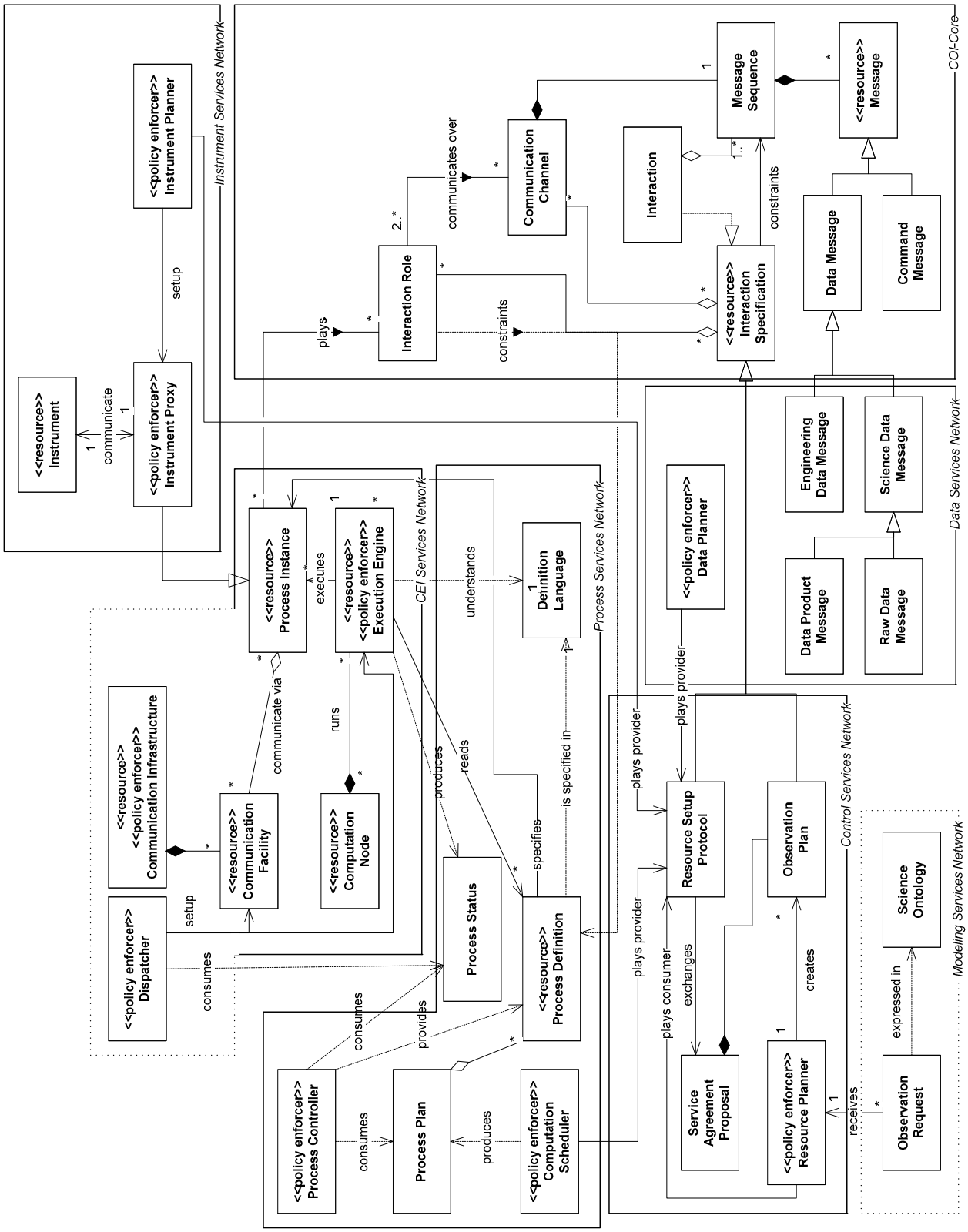

Exhibit 5 - Domain Model Overview

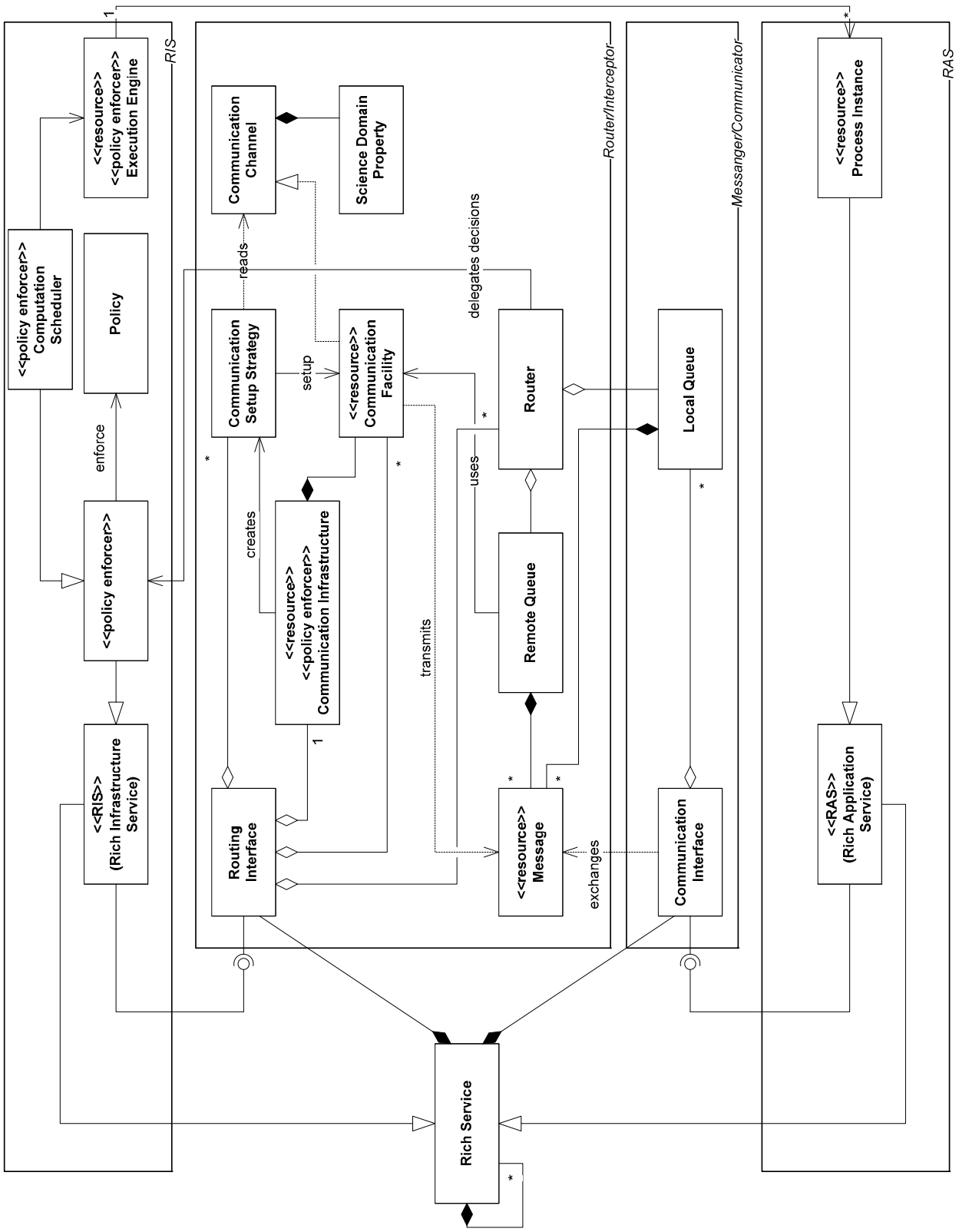

Exhibit 6 - Rich Service Model

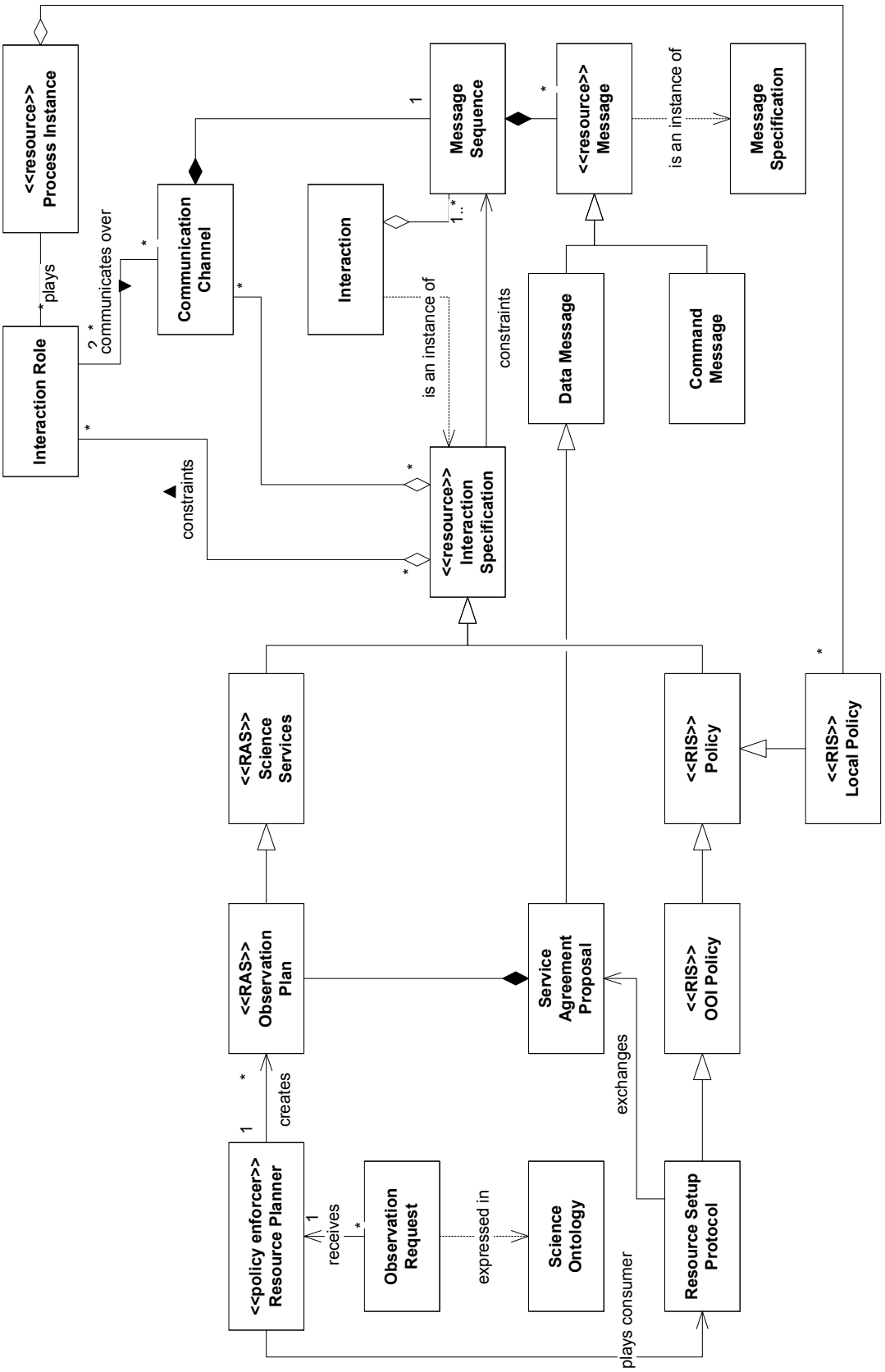

Exhibit 7 - Interaction Model

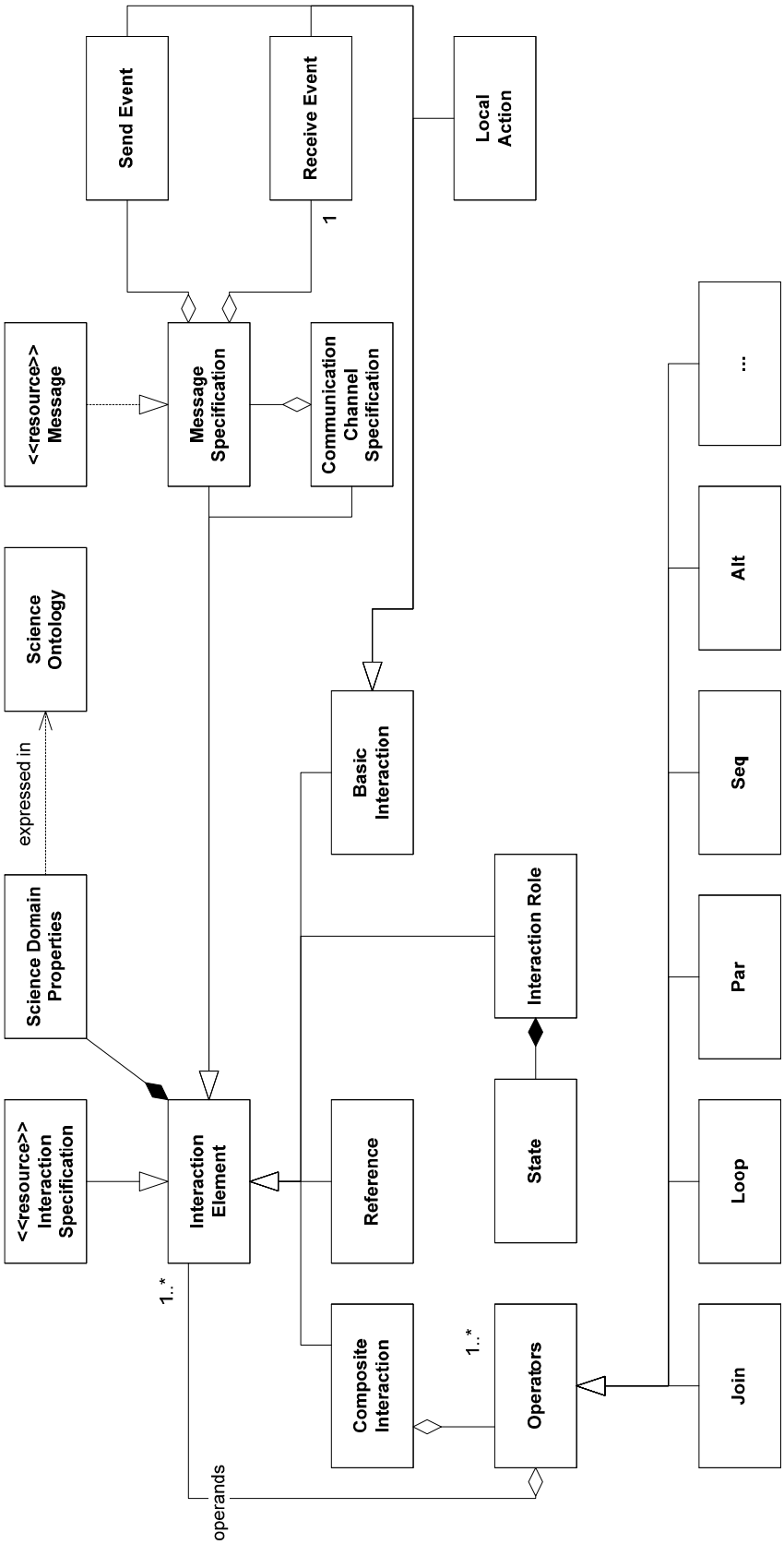

Exhibit 8 - Interaction Specification Model

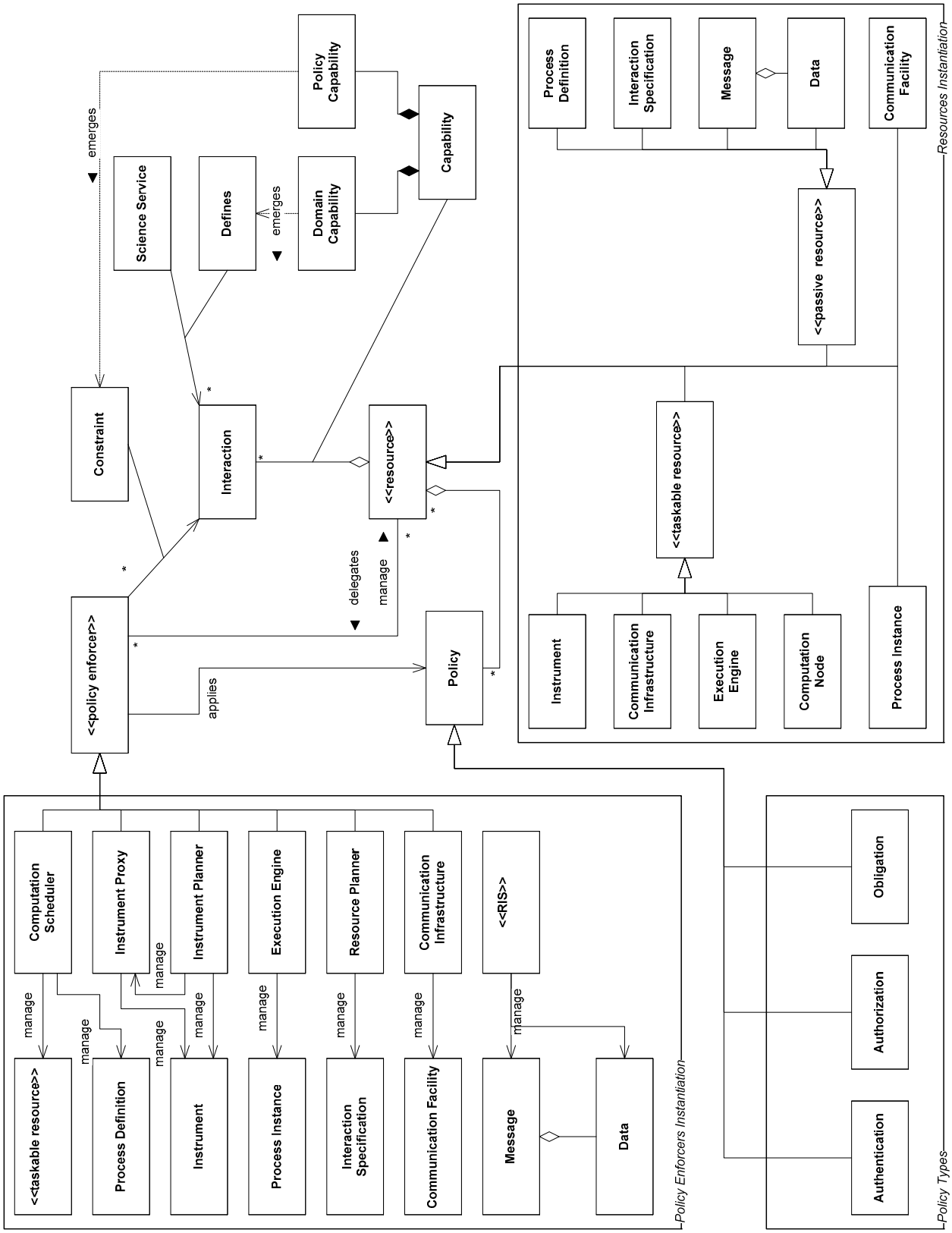

Exhibit 9 - Governance Model

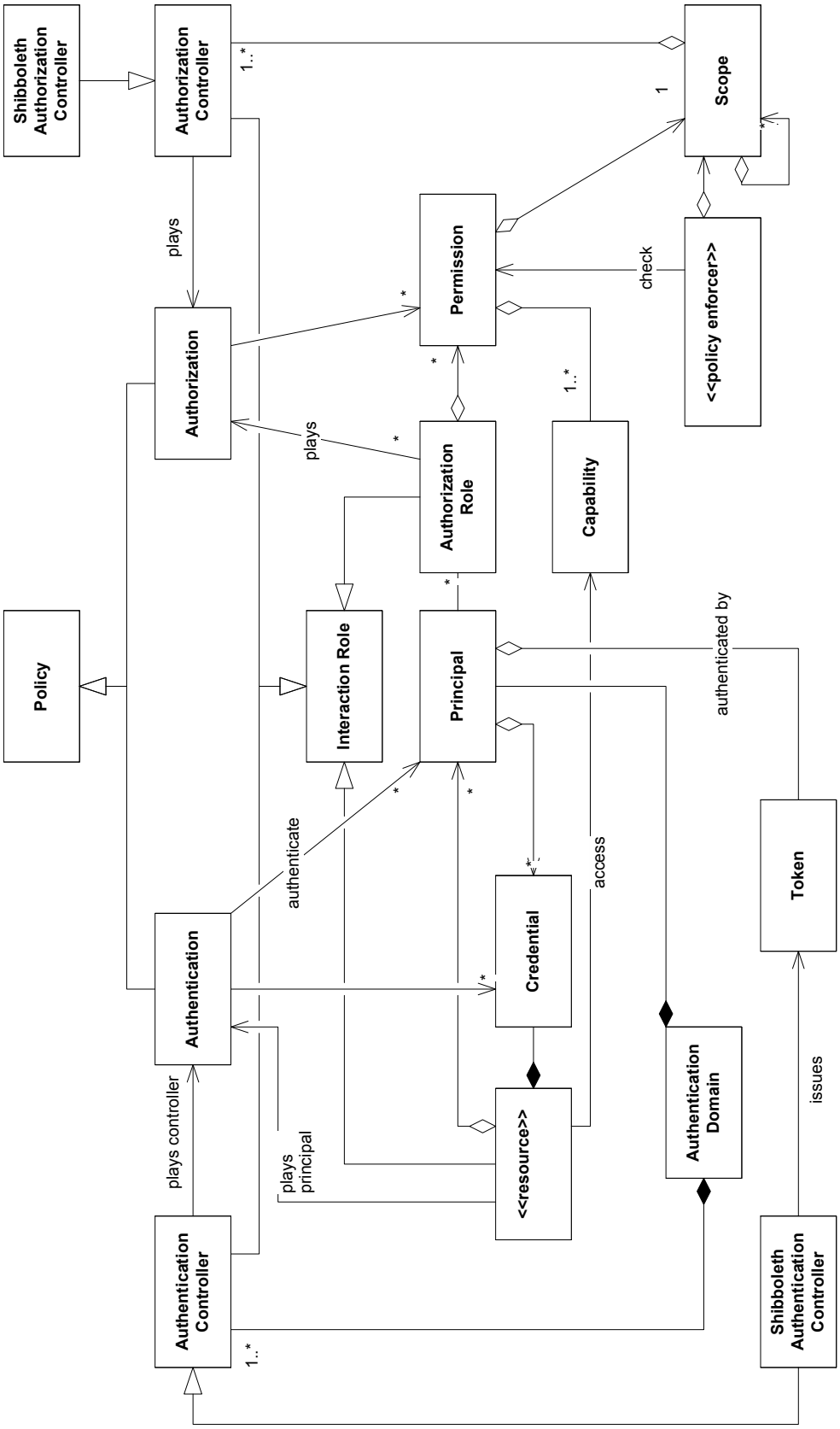

Exhibit 10 - Policy Model

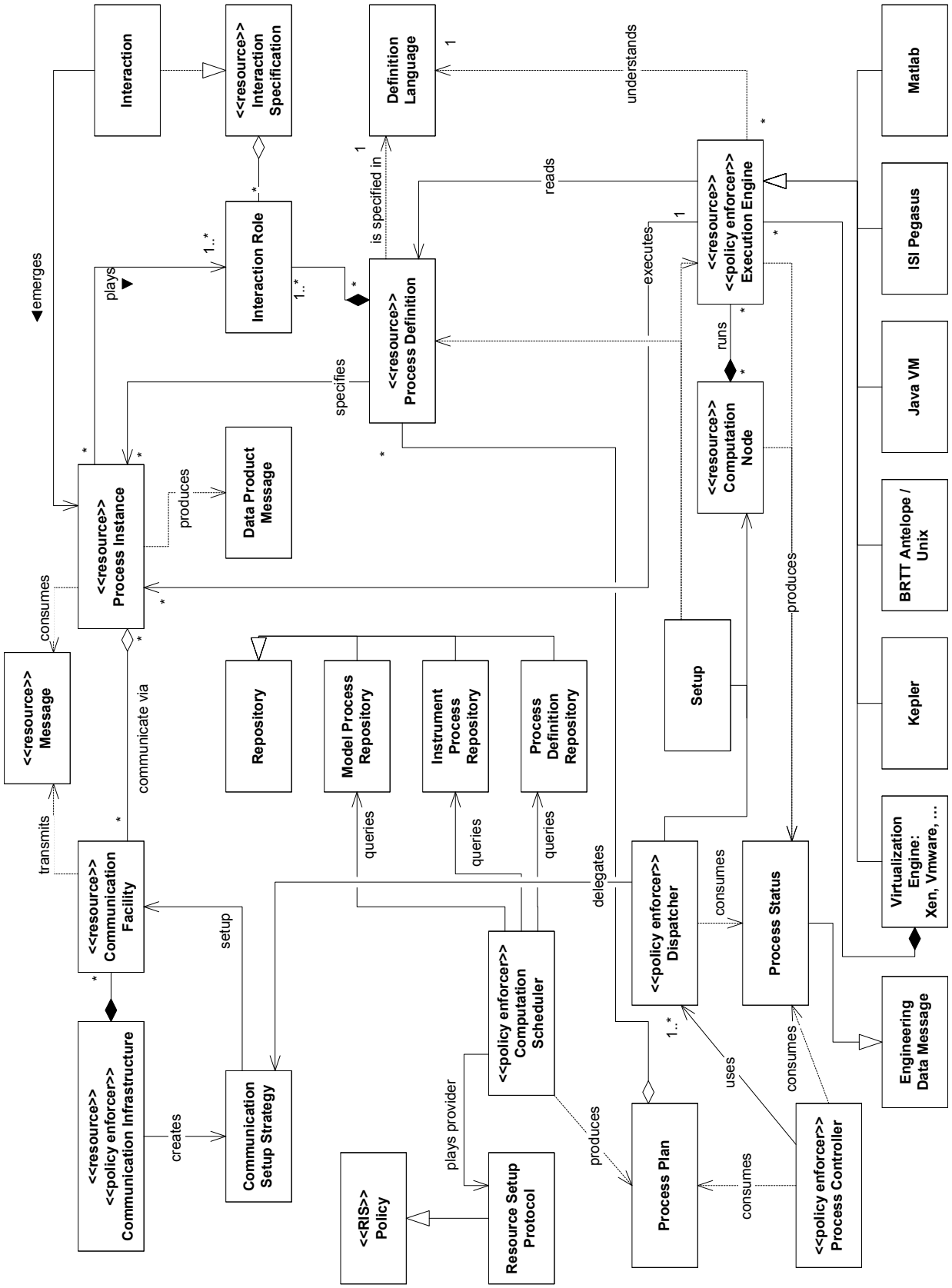

Exhibit 11 - Process Model

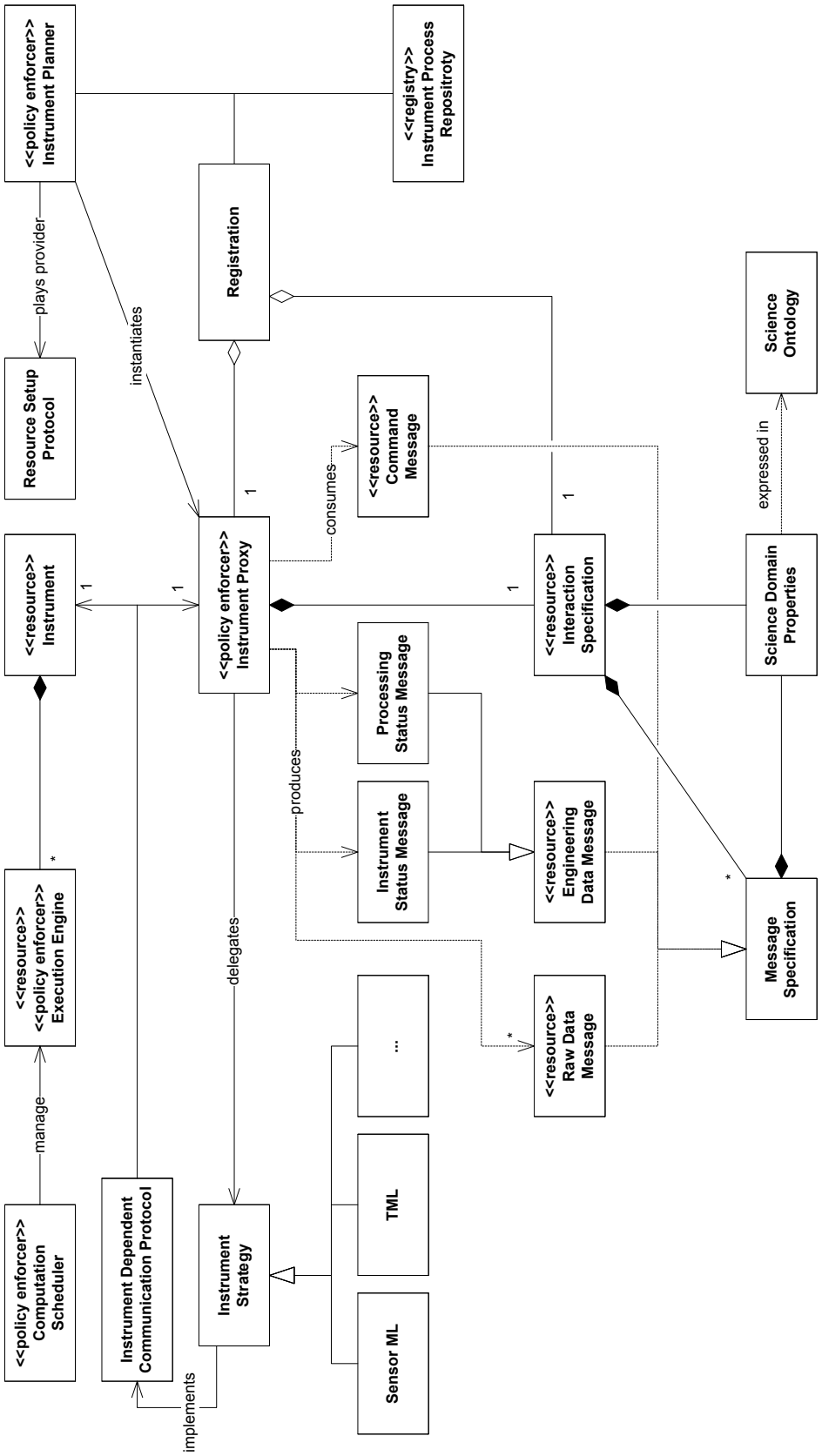

Exhibit 12 - Instrument Model

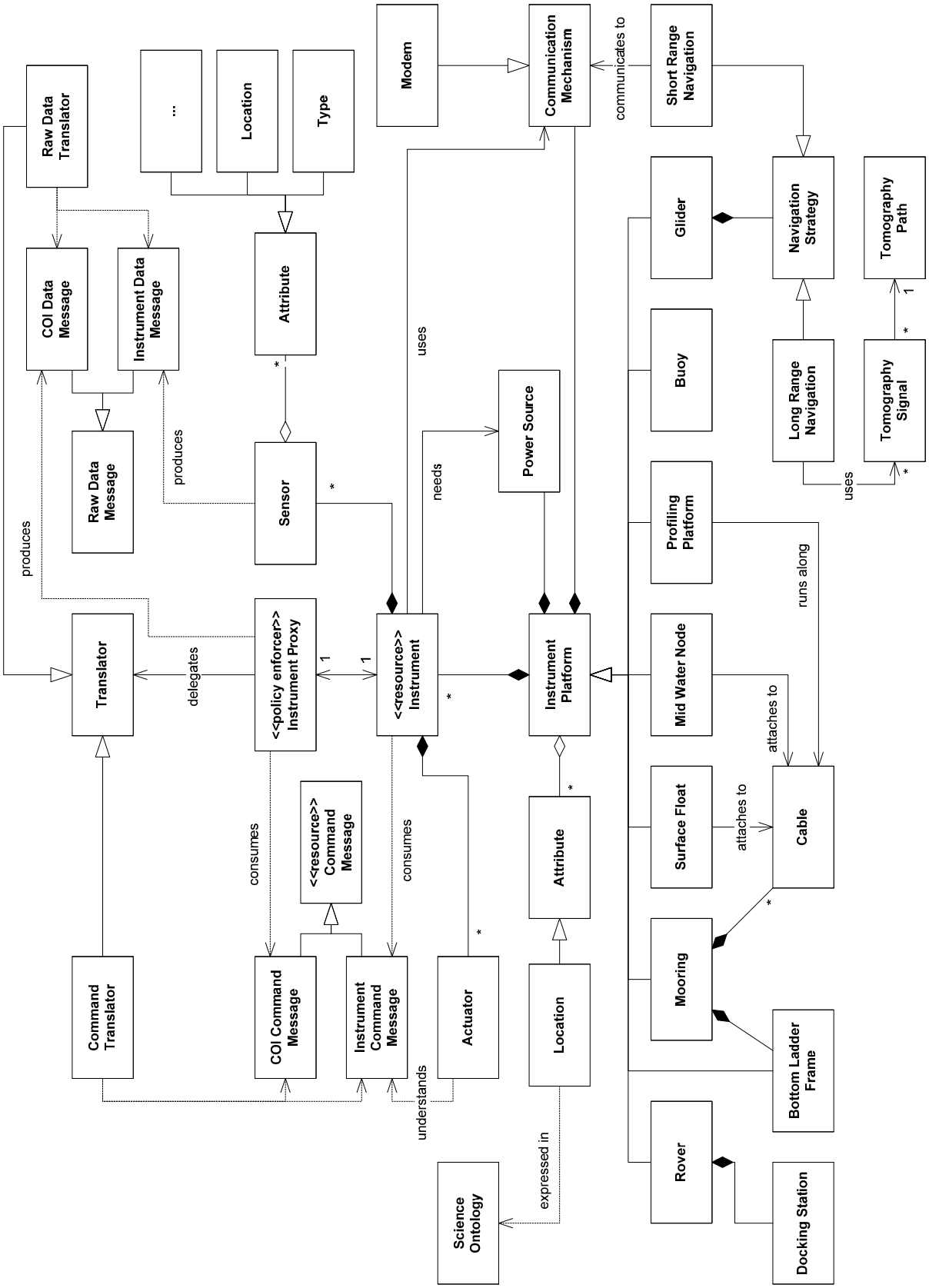

Exhibit 13 - Instrument Platform Model

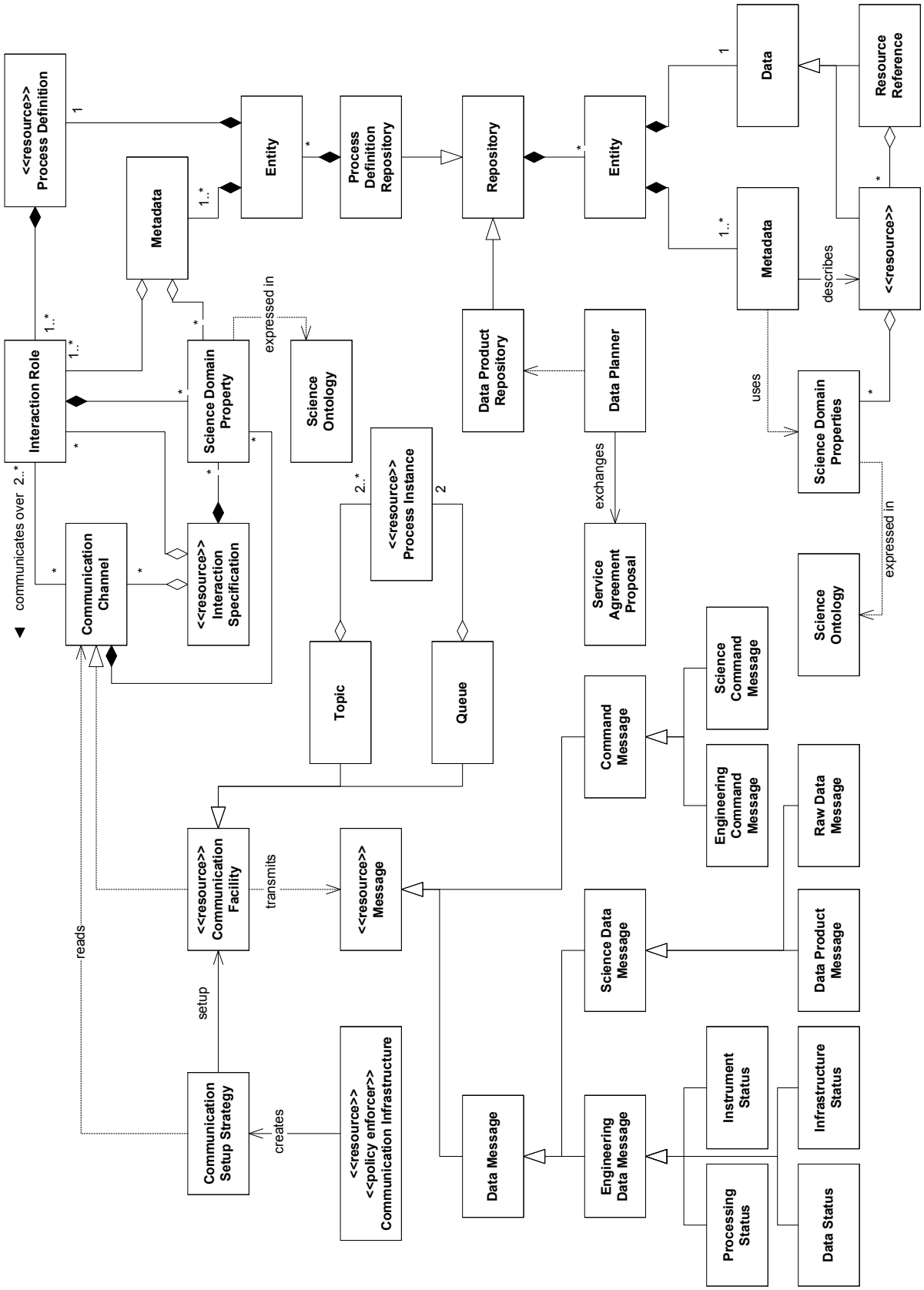

Exhibit 14 - Data Model

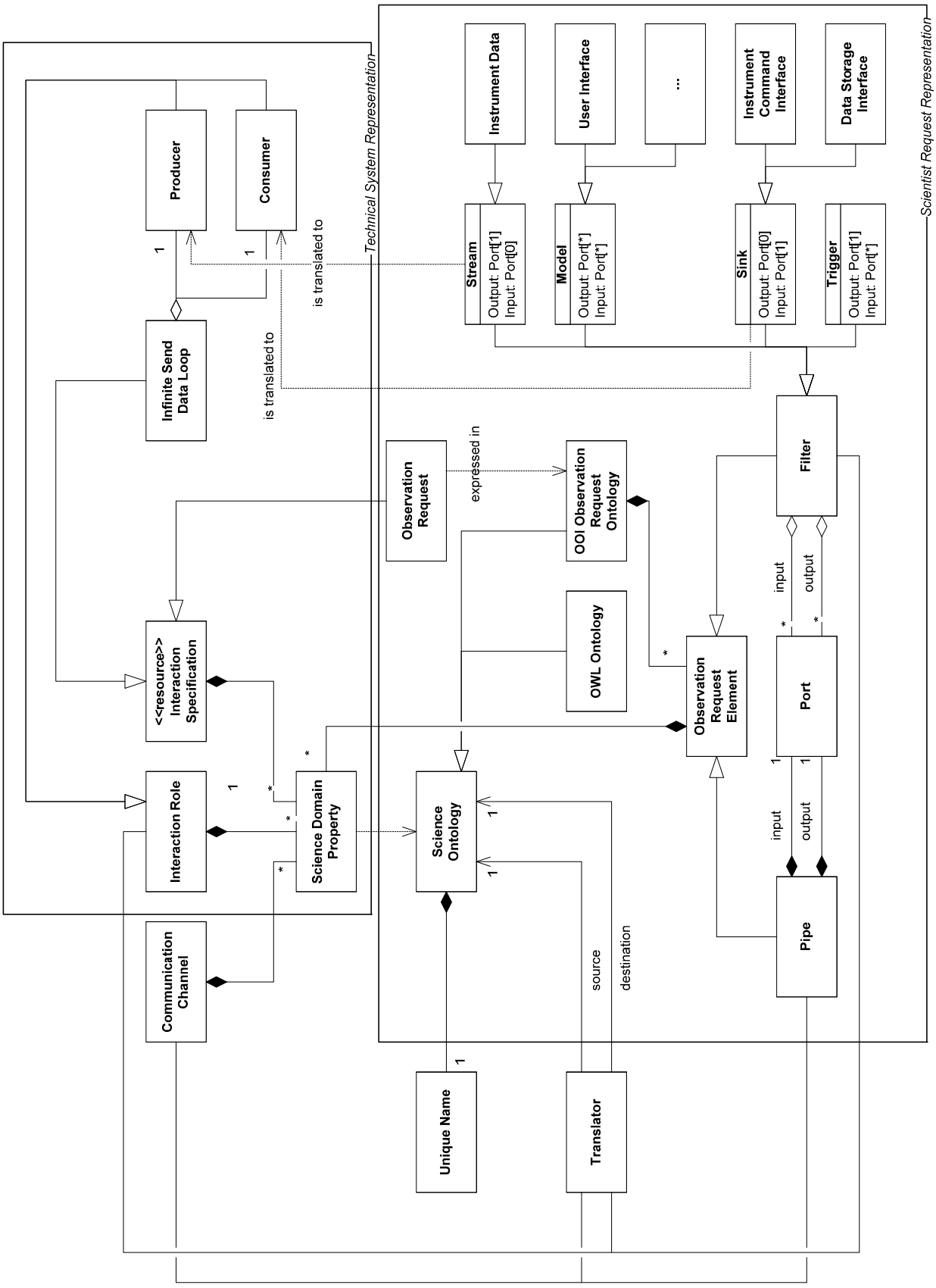

Exhibit 15 - Science Ontology Model