Emergency Resource Management and Coordination

Ilker Murat Karakas*, Dogan Kaya Berktas*, Eren Algan*
*Bilkent University Computer Engineering Department, Ankara Turkey
{karakas,berktas,algan}@cs.bilkent.edu.tr

***Abstract*-In this study, our aim is to come up with a domain-specific language that would be of use in the resource management sub-domain of the emergency management information systems space. We start by conducting a domain analysis. A meta-modeling activity follows the domain-analysis, where we design the abstract syntax, concrete syntax and static semantics for the language that we devise. The meta-model for the language is developed a) a based on MOF from scratch and also b) using UML profiling.**

Keywords-emergency response; coordination; resource management; meta-model; doman-specific language; edxl-rm; ws-bpel

I. INTRODUCTION

Emergencies are multi-causal, requiring complex response mechanisms depending on the nature of the disaster, and are usually manifested in several ways including natural disasters, man-made disasters, and combinations of natural and man-made disasters.

Emergency Management is in essence a set of activities comprising four phases [4]; namely Preparedness, Mitigation, Response and Recovery. The management of emergencies is an endeavor that is characterized by involvement from a multitude of stakeholders, including numerous government agencies, military groups, non-government and charitable organizations, private enterprise and community groups.

Coordination is a cornerstone in emergency-management operations. In order for an effective Emergency/Incident Management System to function, coordination must take place on several levels simultaneously. Coordination ties directly to communication both horizontally and vertically within the chain of command and is often dependent on interagency cooperation to be successful; again this is a frequent cause of failure in both exercises and real-world emergency events. Coordination is essential regardless of whether the response involves a single agency response or several agencies. Coordination of resources controls confusion, prevents freelancing, and strengthens the overall response. Coordination is at different levels. At the incident scene itself, when a mutual aid resource delivers equipment or personnel to an incident; those resources must be coordinated with the response efforts underway at that time. At the regional level, when a major incident occurs that requires a more robust response, some resources could be limited in availability to the Incident Commander.

There are lots of examples of lack of coordination resulting in being unable to effectively deal with the emergencies. At a humanitarian disaster level, significant coordination must occur at all levels of private, public, and government organizations and multiple types of resources and disaster management services and operations are required for assisting a significant population affected by the disaster.

There are currently no meta-models that effectively target the coordination of emergency management operations. There are very few initiatives, but those are either too simplistic, or target some very specific fields like simulation. The idea of a domain-specific language for emergency resource management is innovative in this respect. While building up this language and the meta-model, we base our work on existing industry standards; namely the WS-BPEL [2] meta-model and the existing EDXL-RM [3] resource management specifications.

Effective emergency response is possible only through efficient networking and collaborating of emergency response stakeholders. According to [5], the primary elements enabling effective collaboration are

- *Coordination* - The ability to coordinate activities based on operational response plans.
- *Dynamic Commitment* - The ability to form collaboration commitments with other agents.
- *Shared Knowledge* - The ability to share, understand and utilize domain and external knowledge relevant to the collaboration activity.
- *Agent Context* - The ability to share their specific situational context to guide and monitor coordination status.
- *Situational Knowledge* - The ability to sense, integrate, and process disaster situational knowledge.

- *Utilize Resources* - The ability to effectively utilize a wide variety of resources, including infrastructure.

- *Core services* - The ability to access rights management, agency locator, information discovery, and similar core shared services, and dynamically give them new information and policies.

In addition, under [1] it is stated that being able to coordinate resources at times of emergencies is one of the biggest obstacles in the way of effective emergency management. The Emergency Management operational domain clearly lacks such coordination standards and the necessary tooling that support efficient utilization and coordination of resources during times of emergencies. One reason behind this is the fact that there is a lack of models and meta-models that would allow emergency information management providers build the necessary tooling for supporting the coordination aspects of emergency response operations.

The rest of the paper is structured as follows: First domain analysis is conducted. Then a meta-modeling activity follows the domain- analysis, where the abstract syntax, concrete syntax and static semantics for the language are designed. Then the model-to-model and model to text transformations are performed.

II. Domain Analysis

As mentioned, for this study we have actually decided to work on the topic/domain of Emergency Management with particular focus on the Resource Management sub-domain.

Our research and literature survey indicated the existence of only one resource management standard (i.e. specification), which is the Organization for the Advancement of Structured Information Standards (OASIS) consortium produced/endorsed specification called the Emergency Management Data Exchange Language - Resource Management [3]

Figure 2 explains the message types of EDXL-RM and the actors are described with the request response orders. Some messages can be produced by both users, others can be produced by either the consumer or the supplier.

During the domain analysis, we therefore first looked at the EDXL-RM specification and try to understand the relationship between the concepts. **Figure 3** shows EDXL-RM class diagram.

Together with the domain knowledge that we have, we have produced a draft meta-model, and then we added the necessary meta-level notions from the BPEL meta-model. Thus, we have exploited the following entities under the umbrella of our proposed meta-model:

- Our existing domain knowledge and experience
- OASIS EDXL-RM domain model [3]
- OASIS WS-BPEL Specification, and meta-model [2]

Figure 1 summarizes the domain analysis process we did follow.

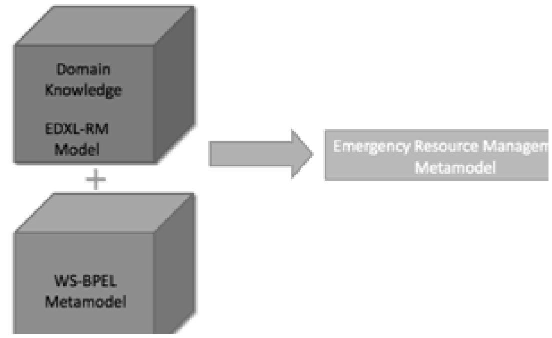

Figure 1 Domain Analysis Process

A. Domain Description/Context

Under this section, we provide information on the domain analysis process and the resulting domain model including glossary of domain concepts. The domain (meta) model is illustrated by **Figure 4**.

B. Domain Lexicon

The glossary of the domain is described in **TABLE 1**.

TABLE 1 Domain Glossary

Id	Concept	Description
C1	*ResourceCoordinationFlow*	Is the 'entry' meta-concept. Comprises a collection of resource coordination processes.
C2	*Flow*	Is a specialization of 'Activity'. Conceptually maps to the BPEL's Flow meta-model entity, therefore carries the very same semantics as defined in BPEL metamodel. Is a container for a number of 'parallel' activities.
C3	*Sequence*	Is a specialization of 'Activity'. Conceptually maps to the BPEL's Sequence meta-model entity, therefore carries the very same semantics as defined in BPEL metamodel. Is a container for a number of 'sequential' activities.
C4	*Process*	A conceptualization for a checklist or a workflow. Contains an activity instance, where the steps of checklist or workflow are defined. Process is a logical wrapper around the Activity meta-concept. At the same time, conceptually maps to the BPEL's Process meta-model entity, and shares the very same semantics as defined in BPEL metamodel.
C5	*Activity*	Base meta-level concept for concept including Flow, Sequence, Invoke, Receive and Reply. Imported from the BPEL meta-model. This meta entitiy allows for definition of compositions of activities. A single activity instance is wrapped by a Process instance.

C6	*Incident*	A meta incident concept. To be specialized by domain models; e.g. an 'earthquake', 'flood', 'manmade disaster', etc. Conceptually, Incident is a happening that has temporal and geospatial projections.
C7	*TemporalCoverage*	Used to assign temporal coverage (i.e. time point or duration) information to either actual incidents, or to the resource management messages
C8	*GeospatialCoverage*	Used to assign geospatial coverage (i.e. location) information to either actual incidents, or to the resource management messages
C9	*AbstractRMMessage*	The meta message concept. Conceptually, the model level specializations of this concept shall allow expressing recource coordination messages between emergency management entities. For EDXL-RM, this meta concept could be used in generalization of the 15 different EDXL-RM messages
C10	*TimeEntity*	Comprises the temporal coverage concept. Could be modelled at the M1 level as a point, duration, etc
C11	*LocationEntity*	Comprises the geospatial coverage concept. Could be modelled at the M1 level as a geospatial point (i.e. a location that has lat/lon), a polygon, etc.
C12	*Resource*	Resource is conceptually the base entity/notion. Anything that can be exchanged between providers and consumer at the time of an incident/emergency is a resource. Examples might include Search and Rescue (SAR) Teams, Mobile Camps, Construction Equipment, Blood Units, Vehicles, Tents, etc.
C13	*RMBaseEntity*	The base meta-concept for some RM concepts including Incident, TimeEntity, LocationEntity, Resource.
C14	*AttributeBag*	A generic attribute storage mechanism for being able to attach RMAttributes to the RMBaseEntity.
C15	*RMAttribute*	Meta-concept that corresponds to attributes to be appended to the AttributeBag.
C16	*Receive*	Is a specialization of 'Activity'. Conceptually maps to the BPEL's Receive meta-model entity, therefore carries the very same semantics as defined in BPEL metamodel. Is used to model reception of a (coordination) message from a sender.
C17	*Invoke*	Is a specialization of 'Activity'. Conceptually maps to the BPEL's Invoke meta-model entity, therefore carries the very same semantics as defined in BPEL metamodel. Is used to model an 'invocation', which essentially means passign a message between 2 entities.
C18	*Reply*	Is a specialization of 'Activity'. Conceptually maps to the BPEL's Reply meta-model entity, therefore carries the very same semantics as defined in BPEL metamodel. Is used to express sending a reply to a message.
C19	*EConfidentiality*	The confidentiality level that belong to an AbstractRMMessage. Is actually an enumeration comprising the standards confidentiality levels (unclassified, restricted, … etc.).

III. Mapping of domain concepts to grammar

A. Grammar

Our metamodel expressed in **Figure 4** is one to one mapped into grammar. This BNF grammar can be interpreted as follows:

ResourceCoordinationFlow (the same class in metamodel) can be composed of many Process classes (again the same name in metamodel). Process class can be ProcessName (which is an identifier, terminal in some sense), Activity, IncidentRef (identifier), ProcessID (identifier). Activity can be ActivityName (identifier), Flow, Sequence, one or zero AbstractRMMessage, Reply, Invoke, and Receives. And so on.

The main idea here is that, there is one to one correspondence between metamodel and grammar. The relations between the components of the metamodel are directly expressed with BNF grammar.

The grammar in BNF notation is expressed below:

```
ResourceCoordinationFlow ::= (Process)*
Process ::= ProcessName Activity IncidentRef ProcessID
ProcessName ::= Identifier
IncidentRef ::= Identifier
ProcessID ::= Identifier
Flow ::= FlowName (Activity)*
FlowName ::= Identifier
Activity ::= ActivityName Flow Sequence
(AbstractRMMessage)? Reply Invoke Receive
ActivityName ::= Identifier
Sequence ::= SequenceName (Activity)*
SequenceName ::= Identifier
AbstractRMMessage ::= AbstractRMMessageName
(Activity)* TemporalCoverage GeospatialCoverage
AbstractRMMessageID (ResourceRef)* Confidentiality
Reply ::= Target Source
Invoke ::= Target Source
Receive ::= Target Source
Target ::= Identifier
Source ::= Identifier
TemporalCoverage ::= TimeEntityName TimeEntity
GeospatialCoverage ::= LocationEntity
AbstractRMMessageID ::= Identifier
ResourceRef ::= Identifier
Confidentiality ::= ConfidentialityType
ConfidentialityType ::= Identifier
RMBaseEntity ::= Incident TimeEntity Resource
LocationEntity AttributeBag
Incident ::= TemporalCoverage GeospatialCoverage
TimeEntity ::= Time
Resource ::= ResourceType
LocationEntity ::= Location
AttributeBag ::= (IncidentAttribute)*
IncidentAttribute ::= IncidentAttributeName
Time ::= Identifier
ResourceType ::= Identifier
Location ::= Identifier
IncidentAttributeName ::= Identifier
TimeEntityName ::= Identifier
AbstractRMMessageName ::= Identifier
```

IV. Definition of Metamodel Based on MOF-from Scratch

A. Abstract Syntax

The abstract syntax contains the metamodel and its mapping to MOF components. **Figure 5** shows the mapping between these two phases, which are meta-metamodel and metamodel.

B. Concrete Syntax

The concrete syntax of our metamodel is expressed in **Figure 6** .To create a consistent and user-friendly concrete syntax, existing UML's class diagram paradigm is extended. For this, a flow and sequence logic is embedded to a outer bounding box and an incident box is also added to this outer box. With this, the aim of incorporating the sequences of messages with the predefined EDXL-RM messages is achieved.

When the concrete syntax is analyzed, it will be easily seen that there is a similarity between the metamodel and the syntax. There is a process with a process name in the outer box, which has a sequence, a flow and an incident. In the flow, there are two activities in which the RM messages are held. RM messages have attributes such as MessageID (a number that identifies that message), Confidentiality (messages can have different levels of confidentiality), MessageType (determines the type of the message), Location, Resource and TimeDuration.

This concrete syntax can be theoretically created via a tool. Think of a tool like EA where process, flow, sequence, activity, incident, RM messages are little boxes. With drag and drop property, these boxes are combined a syntax as in Figure 6 is produced.

C. Static Semantics

Static semantics of a metamodel defines the well-formedness rules of it. These well-formedness rules are used for both defining constraints on how models can be formed, and validating the models constructed upon a specific metamodel.

In order to express that every process has a unique process id and this is the case for all of the entities in the structure, below constraints are formed. Also, the geospatial coverage within the AbstractRMMessage should be single in one message instance since there can be only one geospatial range for a message.

```
context Process inv: Process::allInstances()->isUnique(processId)
context AbstractRMMessage inv: AbstractRMMessage::allInstances()->isUnique(messageId)
context RMAttribute inv: RMAttribute::allInstances()->isUnique(attributeName)
context Incident inv:self.geospatialCoverage = self.temporalCoverage
context GeospatialCoverage ERROR "exactly one local entity required":localEntity. size == 1;
context Process ERROR loc()+" processes must have unique Ids: "+processId:
        ((ResourceCoordinationFlow)coordiantionFlow).processes.select( c | c.processId == processId ).size == 1;
context AbstractRMMessage ERROR loc()+" messages must have unique message Ids: " + messageId:
        ((Activity)ctivitiy).messages.select( c | c. messageId == messageId ).size == 1;
context Process WARNING loc()+" id not specified: ["+process.processId+"]":
        processId != null;
context AbstractRMMessage WARNING loc()+" id not specified: ["+ abstractRMMessage.messageId+"]":
        messageId != null;
```

D. Example Models

The model in Figure 8 exemplifies an earthquake with a sequence, which contains two flows. In this model example, there has been an earthquake in Istanbul and all hospitals in Marmara and Kızılay were requested some resource. 10 doctors and 5 nurses were asked to Avcılar, 10 tents and 5 staff were asked to KüçükÇekmece province. This is indeed a very specific model which shows the expandability of our metamodel.

Other model in Figure 7 is, on the other hand, a generic model example. The idea here is to show what kind of messages can be generated via our metamodel. The flow is represented as in the format of a feature diagram. This model can be better analyzed with our concrete syntax example.

V. DEFINITION OF METAMODEL USING UML PROFILING

The extension mechanism of UML allows modeler to define stereotypes and introduce tagged values to them in a formal way. Using profiling mechanism of the UML 2. *, we redefine our metamodel. **TABLE 2** lists the stereotypes introduced with this extension process.

TABLE 2 PROFILE AND STEREOTYPES

Model Element	Stereotype	UML Metaclass
ResourceCoordinationFlow	*EMRMResourceCoordinationFlow*	Class
Flow	*EMRMFlow*	Class
Sequence	*EMRMSequence*	Class
Process	*EMRMProcess*	Class
Activity	*EMRMActivity*	Class
Incident	*EMRMIncident*	Class
TemporalCoverage	*EMRMTemporalCoverage*	Class
GeospatialCoverage	*EMRMGeospatialCoverage*	Class
AbstractRMMessage	*EMRMAbstractRM Message*	Class
TimeEntity	*EMRMTimeEntity*	Class
LocationEntity	*EMRMLocationEntity*	Class
Resource	*EMRMResource*	Class
RMBaseEntity	*EMRMRMBaseEntity*	Class
AttributeBag	*EMRMAttributeBag*	Class
RMAttribute	*EMRMRMAttribute*	Class
Receive	*EMRMReceive*	Class
Invoke	*EMRMInvoke*	Class
Reply	*EMRMReply*	Class
EConfidentiality	*EMRMEConfidentiality*	Class

Metamodel using UML profiling is shown in **Figure 9**.

VI. MODEL TO MODEL TRANSFORMATION

Model transformation is a key problem for MDD. Model analysis, refactoring, model synchronization, code generation, deployment, etc. are all handled with numerous tools that require different tools and input models. To achieve interoperability, various model transformation languages and tools are developed.

Model to model transformation is an important aspect of model driven software development. In our case, the input model is the emergency model which gets together aspects of EDXL-RM and BPEL workflow together.

As BPEL artifacts are supported by many tools, it is a wise approach to have the ability to transform our model to BPEL model. To do so, ATL is used. ATL (ATLAS Transformation Language) is a model transformation language and toolkit. ATL provides ways to produce a set of target models from a set of source models [6]. ATL requires mapping the source metamodel and target metamodel to be mapped with ATL language.

The source metamodel (Emergency.ecore) and output metamodel (bpel.ecore) are given in Figure 11 and Figure 12 respectively.

The atl file which handles the mapping between the source metamodel and target metamodel is also given in the run configuration. The output model file destination is set as an xpi file. The path is also given in the runtime configuration (See Figure 10).

The content of ATL file is as follows which handles the basic mapping of two metamodels:

```
module AltDeneme1; -- Module Template
create Out : Bepl from IN : Emergency;

rule ProcessMapping
{
	from
		a : Emergency!Process
	to
		p : Bepl!Process (
			name <- a.processId,
			activity <- a.activity

		)
}
rule FlowMapping
{
	from
		a : Emergency!Flow
	to
		p : Bepl!Flow
		(
			name <- a.activityId,
			activities <- a.activity
		)
}
```

```
rule InvokeMapping
{
	from
		a : Emergency!Invoke
	to
		p : Bepl!Invoke
		(
			name <- a.activityId,
			inputVariable <- a.message
		)
}


rule ReceiveMapping
{
	from
		a : Emergency!Receive
	to
		p : Bepl!Receive
		(
			name <- a.activityId,
			variable <- a.message
		)
}

rule ReplyMapping
{
	from
		a : Emergency!Reply
	to
		p : Bepl!Reply
		(
			name <- a.activityId,
			variable <- a.message
		)
}

rule AbstractRMMessageMapping
{
	from
		a : Emergency!AbstractRMMessage
	to
		p : Bepl!Variable
		(
			name <- a.messageId,
			messageType <- a.messageType
		)
}
```

Process in the emergency domain directly mapped to BPEL process. Flow, sequence, receive, reply, invoke are all similar to BPEL processes. The challenging part is to transform AbstractRMMessage to BPEL counterpart. For instance, an invoke instance in emergency domain contains an AbstractRMMessage. The Variable in BPEL model contains similar features with our AbstractRMMessage. These are name, messsageType. These two enable us to map these two and transform AbstractRMMessage to a BPEL Variable.

Our model example can be seen in Figure 13. After applying ATL mapping, the following xmi code is produced:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xmi:XMI xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:bpws="http://docs.oasis-
  open.org/wsbpel/2.0/process/executable">
      <bpws:Process name="12345">
        <activity xsi:type="bpws:Invoke" name="i1"
    inputVariable="/1"><bpws:Variable name="m1"/>
        </activity>
    </bpws:Process>
  </xmi:XMI>
```

VII. MODEL TO TEXT TRANSFORMATION

In Model-Driven Software Development, the generation of textual artifacts – often source code – plays an important role. However, often, code generation is seen as the “less important brother” of model-to-model transformations and is consequently treated as a second-class citizen. However, most developers come into MDSD through “simple” code generation and in most cases, the last step of a transformation chain is actually a code generator. It is therefore important that the generator is up to the challenge of generating non-trivial software systems.

openArchitectureWare [7] is a framework for model-driven software development. oAW comes with a host of features necessary for MDSD, including M2M transformations, declarative constraints checking, a workflow engine, adapters for the XMI of a variety of UML tools, EMF integration, nice Eclipse IDE integration (with custom editors and static error checking) as well as a proven template language for code generation called Xpand. Specifically the code generation language has been available for a number of years, so there is considerable industry experience available for that language [8].

With the help of Xpand, complete java codes can be seen in page starting from 19 (in order to save some space, commands are deleted).

VIII. LESSONS LEARNED AND CONCLUSION

Model driven software development’s Achilles' heel is the process of getting used to thinking in ‘meta’. This is hard for a programmer since, he is used to thing in M1 level instead of M2 level. Therefore, it takes time to become familiar with to M2 level and start creating metamodel for a particular model. For this phase, we try to put ourselves in to the shoes of a tool developer and try to see our metamodel for her perspective. This approach helps us a lot, however, again, it is quite difficult to resist the temptation of modeling on the wrong level! (i.e. M1 in place of M2). In the process of creating the artifacts, we see that, MDSD is all about a universal, consistent platform for enable creation and exchange of models, in reality especially exchange of model and metamodel between platforms and tools are very cumbersome. For instance, we used Enterprise Architect (EA) and to create OCL we tried OpenArchitectureWare (OAW) but what we see is the XMI output of EA is inconsistent with the import mechanism of OAW.

Even though Model Driven Software Development has been one of the brightest subjects in Computer Science for few years now, there are still so many problems yet to be solved. The main problem of MDSD is the inconsistency of the tools. Most of the projects in Eclipse are in incubation state. Although the idea of MDSD is to increase productivity, these bugs and problems in the tools reduces the productivity of the programmer / designer.

REFERENCES

[1] Rui Chen, Raj Sharman, H Raghav Rao, and Shambhu J Upadhyaya. Coordination in emergency response management. Communications Of The Acm, 51(5):8, Apr

[2] OASIS WS-BPEL Technical Committee. Web services business process execution language version 2.0. pages 1–66, May 2007.

[3] OASIS Emergency Management TC. Emergency data exchange language resource messaging (edxl-rm) version 1.0. pages 1–174, Nov 2009.

[4] Jim Steel, Renato Iannella, and Ho-Pun Lam. Using ontologies for decision support in resource messaging. Proceedings of the 5 International ISCRAM Conference, page 9, Apr 2008

[5] David Aylward, Jim Bound, Bonnie Gorsic, Steve Gross, Walter LeGrand, Paul Mangione, Harrison Miles, Nelson Santini, Amin Soleimani, and John Yanosy. Findings and recommendations for mobile emergency communications interoperability (meci). NCOIC, page 152, Jan 2007.

[6] Eclipse ATL Project Model To Model Transformation, http://www.eclipse.org/m2m/atl

[7] openArchitectureWare, http://www.openarchitectureware.org/

[8] Markus Völter, Bernd Kolb, Best Practices for Model-To-Text Transformations, Sep 2006

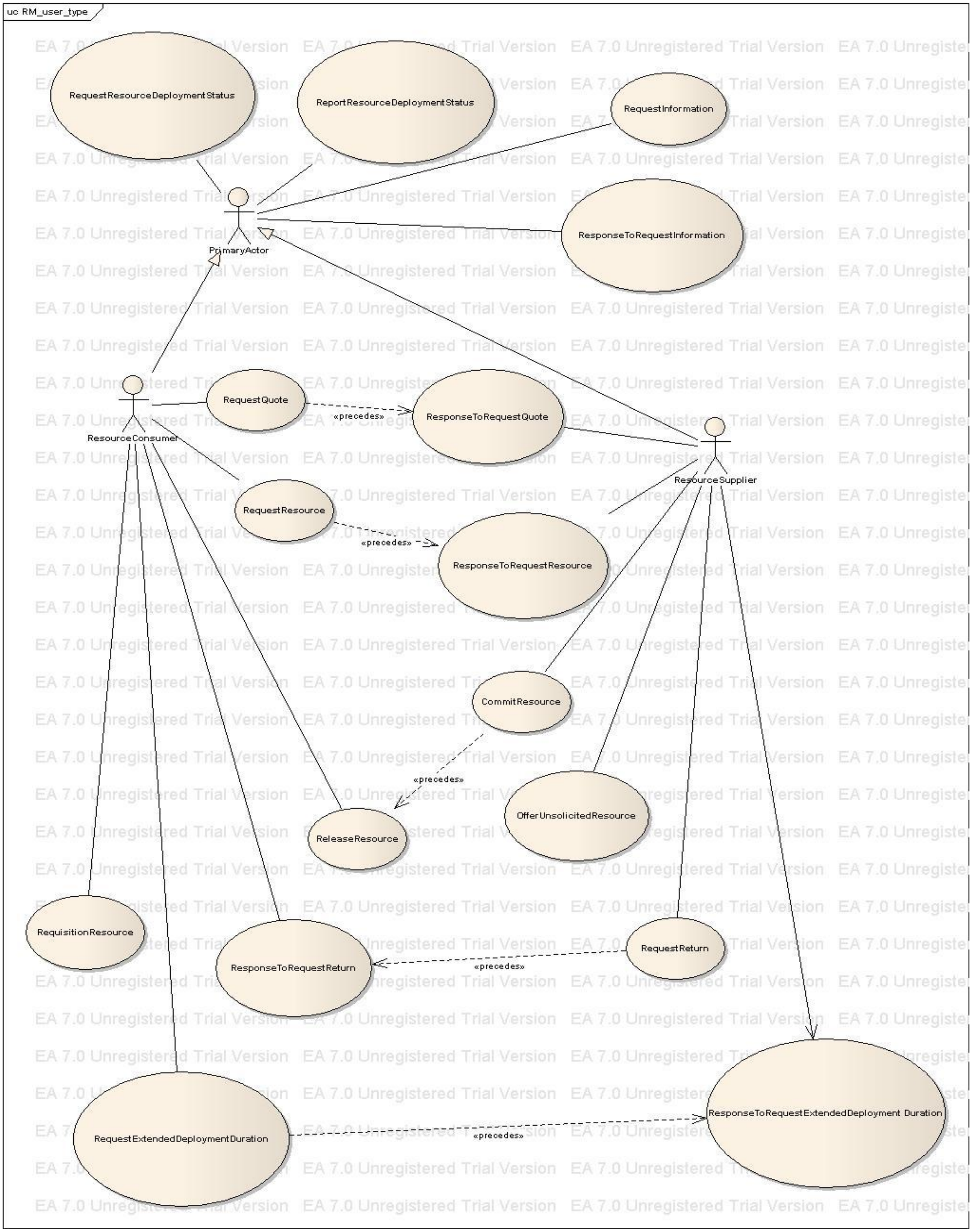

Figure 2 Use Case diagram for EDXL-RM

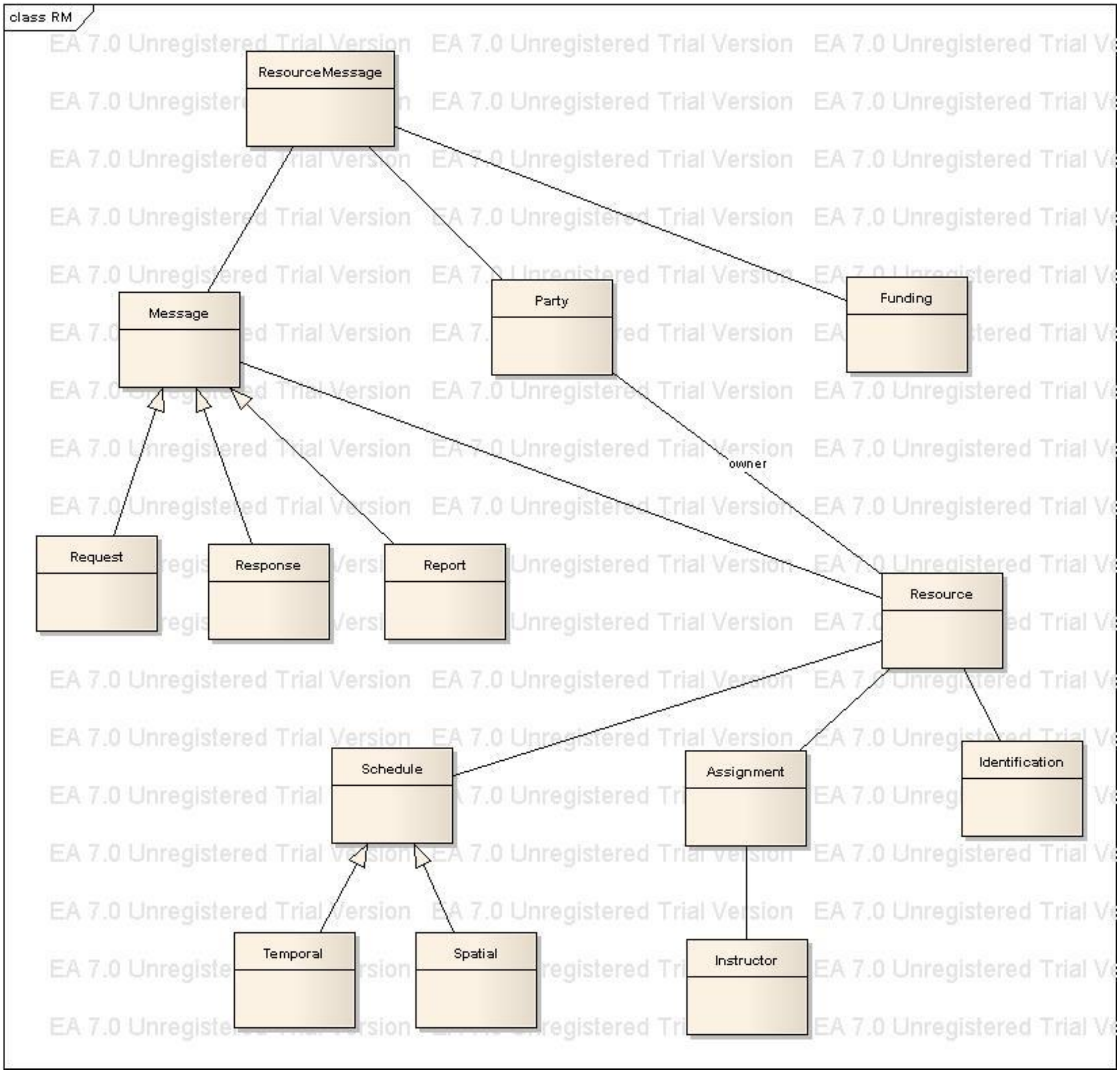

Figure 3 Class Diagram for EDXL-RM

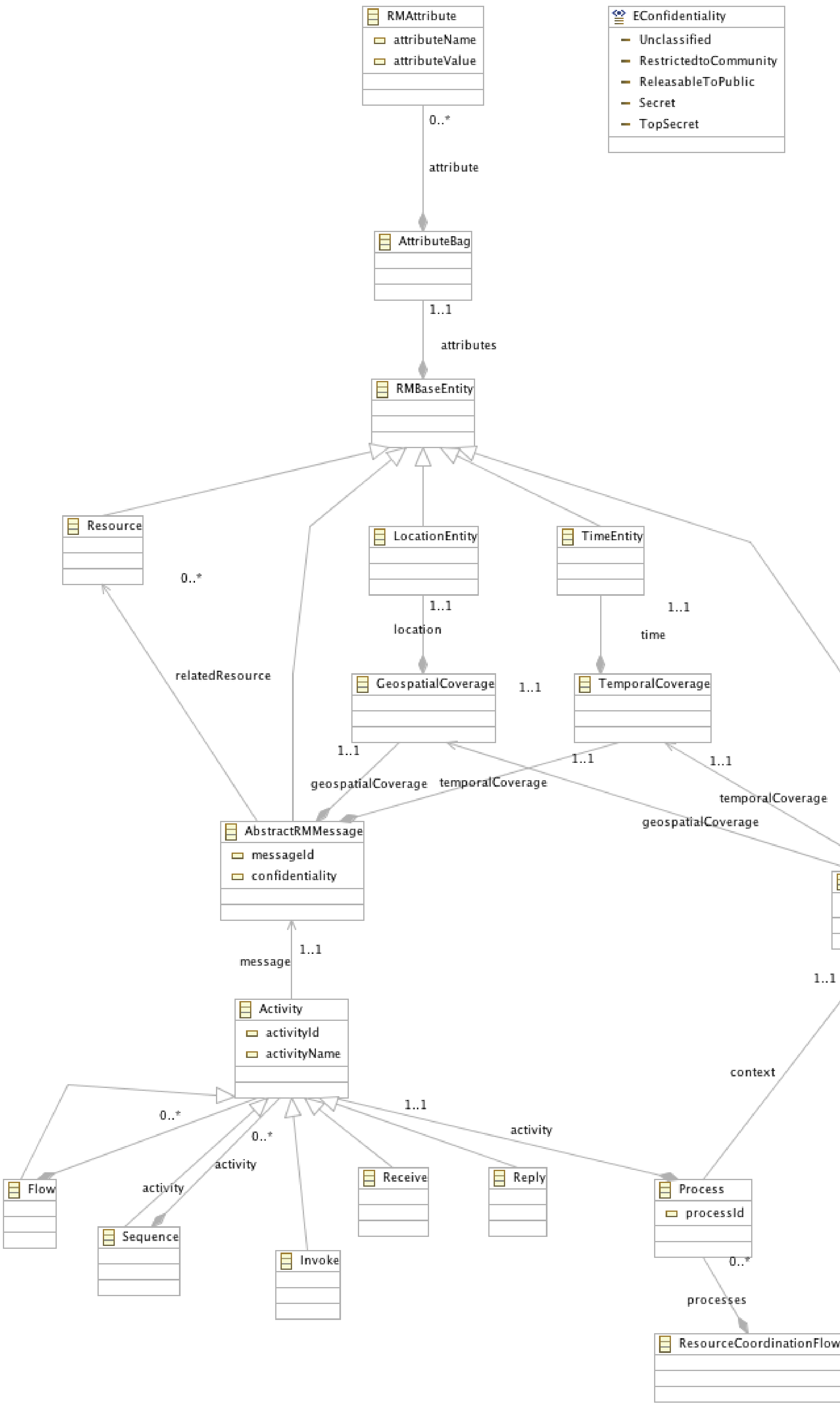

Figure 4 Metamodel of our domain

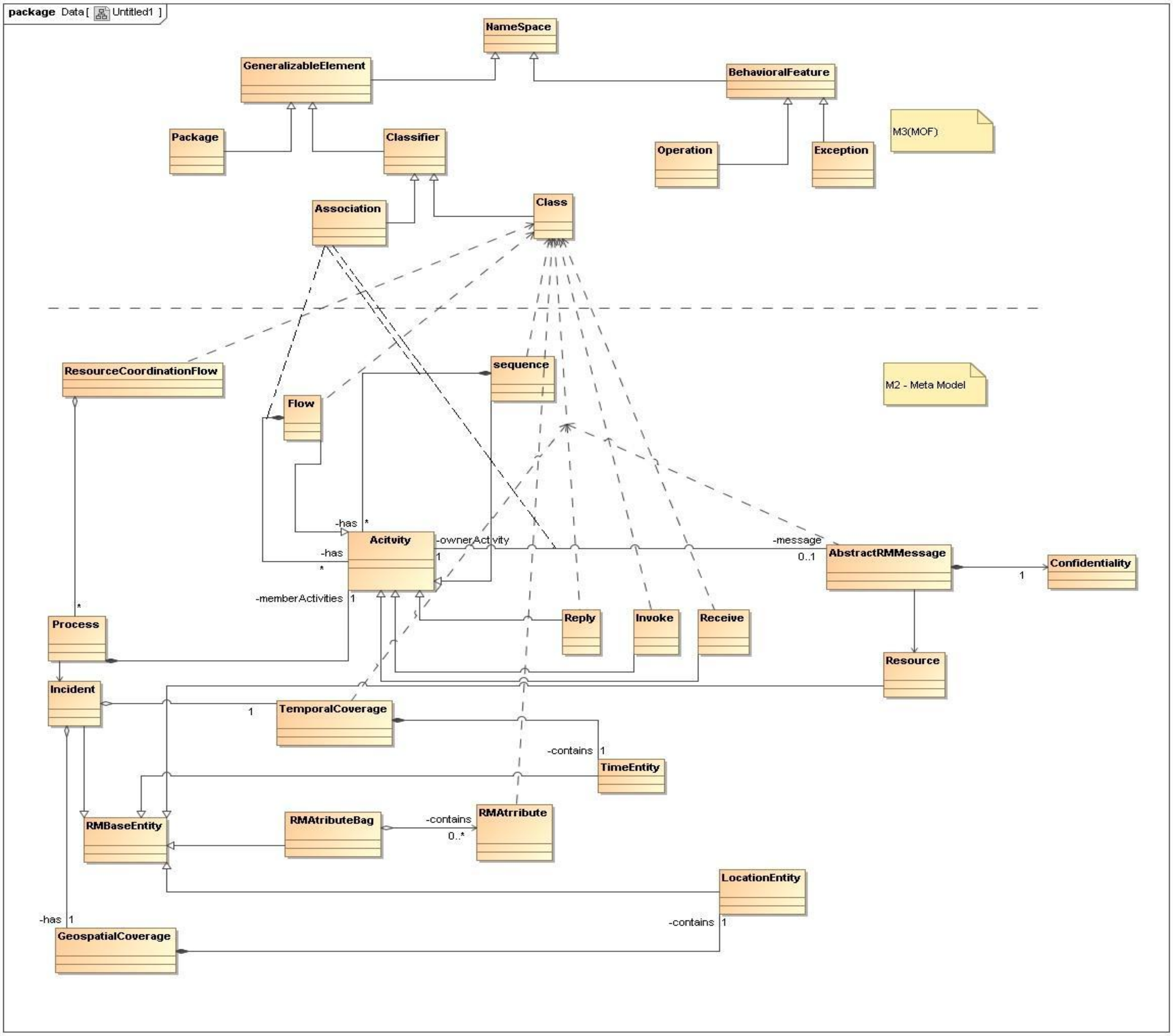

Figure 5 Metamodel and MOF Mapping

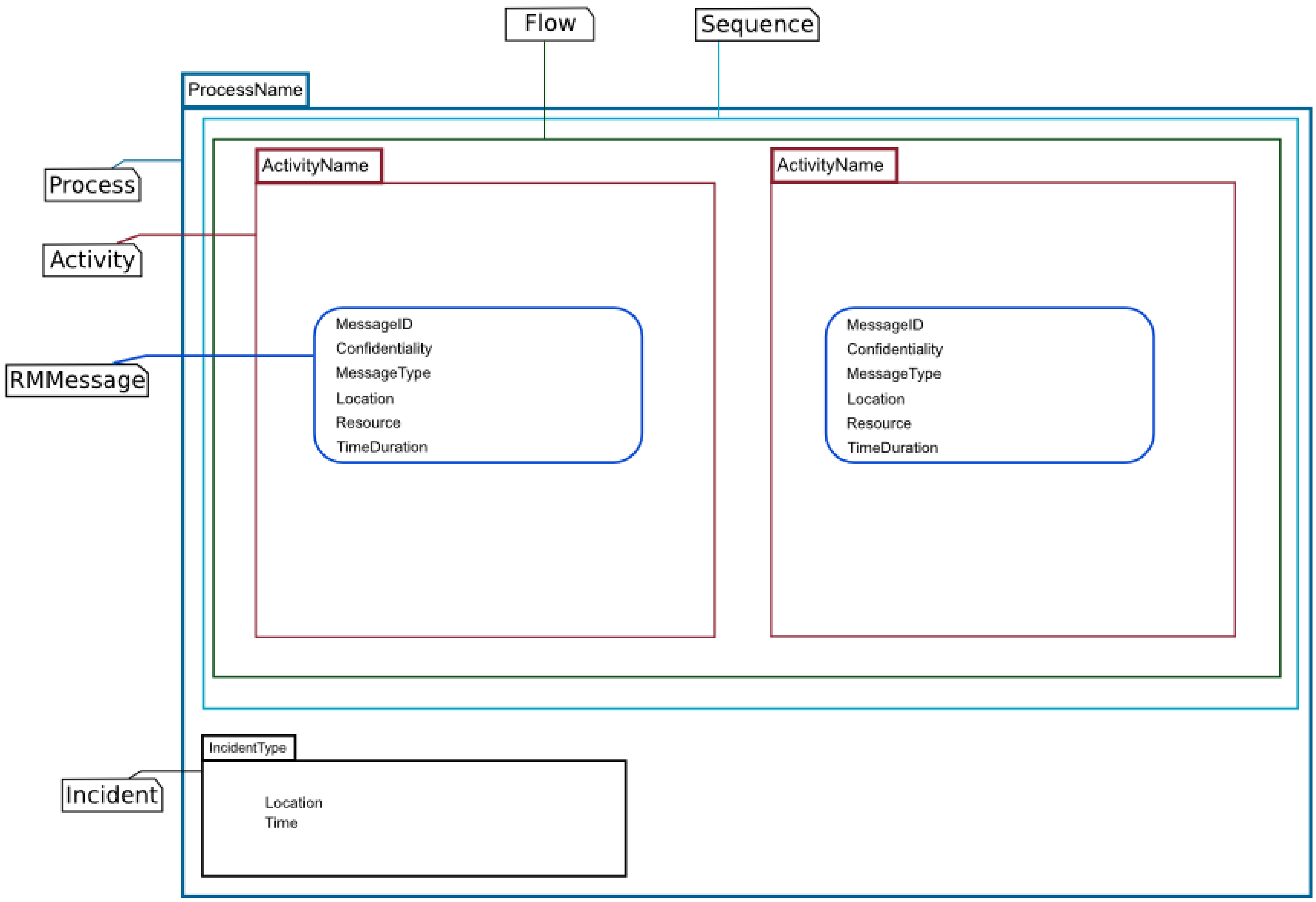

Figure 6 Concrete Syntax

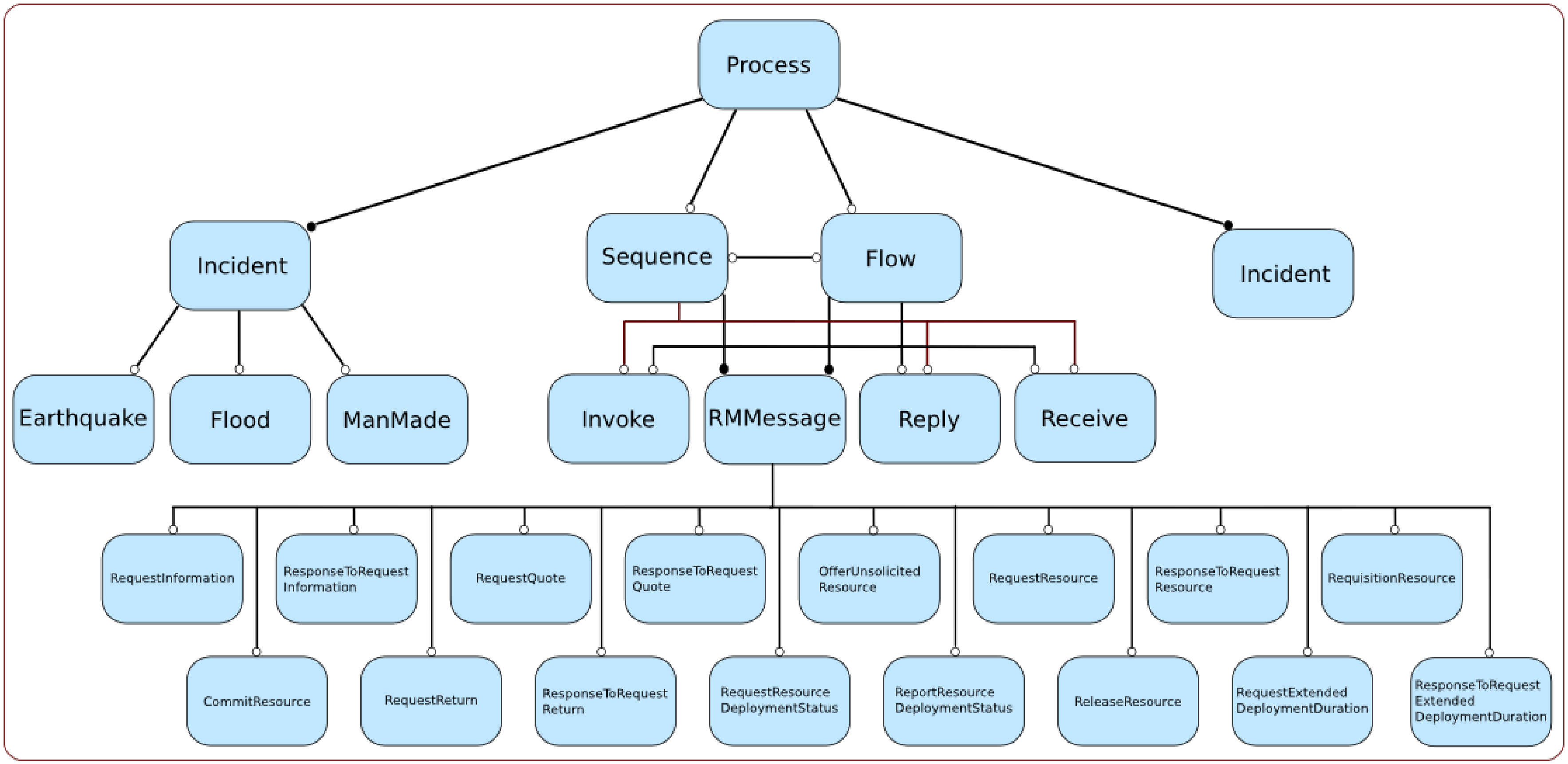

Figure 7 Model Example

```
<process name = "IstanbulEarthquake99"
	<incident name = "IstanbulEarthquake">
		incidentType = "Earthquake"
		geospatialCov = "lat41.1&lon29.0"
		location = "Istanbul"
		time = "17/08/1999 05.45AM"
	</incident>
	<partnerLinks>
		<partnerLink name = "AllHospitalsAround">
			partnerLinkAddress = "http://www.allhospitalsInMarmaraServer.com"
			myRole = "client" />
		<partnerLink name = "Kizilay">
			partnerLinkAddress = "http://www.kizilayyardimserver.com"
			myRole = "client" />
	</partnerLinks>
	<sequence name = "RequestingResourceFromHospitalsAndKizilay">
		<invoke name = "invokeHospitalServer"
			partnerLink = "AllHospitalAround"
			createInstance = "yes"
			operation = "sourceRequests"/>
		<invoke name = "invokeKizilayServer"
			partnerLink = "Kizilay"
			createInstance = "yes"
			operation = "sourceRequests"/>

		<flow name = "HospitalFlow">
			<requestResource name="10Doctors">
				geospatialCov = "lat41.0300319&lon28.6938"
				time = "17/08/1999 05.45AM"
				location = "Avcilar"
			</requestResource>
			<requestResource name="5Nurses">
				geospatialCov = "lat41.0300319&lon28.6938"
				time = "17/08/1999 05.45AM"
				location = "Avcilar"
			</requestResource>
		</flow>

		<flow name = "KizilayFlow">
			<requestResource name="10tents">
				geospatialCov = "lat41.0308088&lon28.7884712"
				time = "17/08/1999 05.45AM"
				location = "KucukCekmece"
			</requestResource>
			<requestResource name="5staff">
				geospatialCov = "lat41.0308088&lon28.7884712"
				time = "17/08/1999 05.45AM"
				location = "KucukCekmece"
			</requestResource>
		</flow>
	</sequence>
</process>
```

Figure 8 Model Example 2

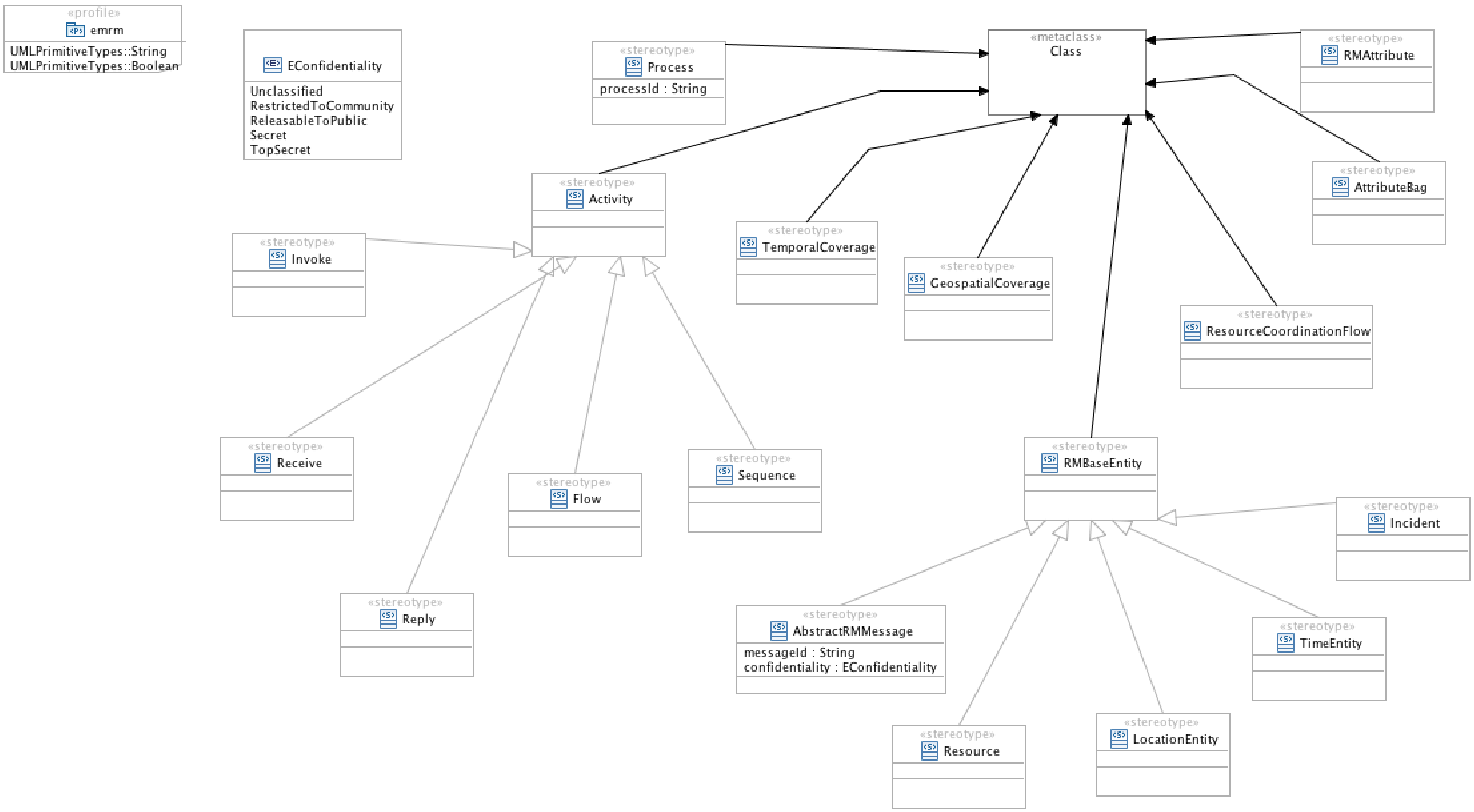

Figure 9 UML Profiling

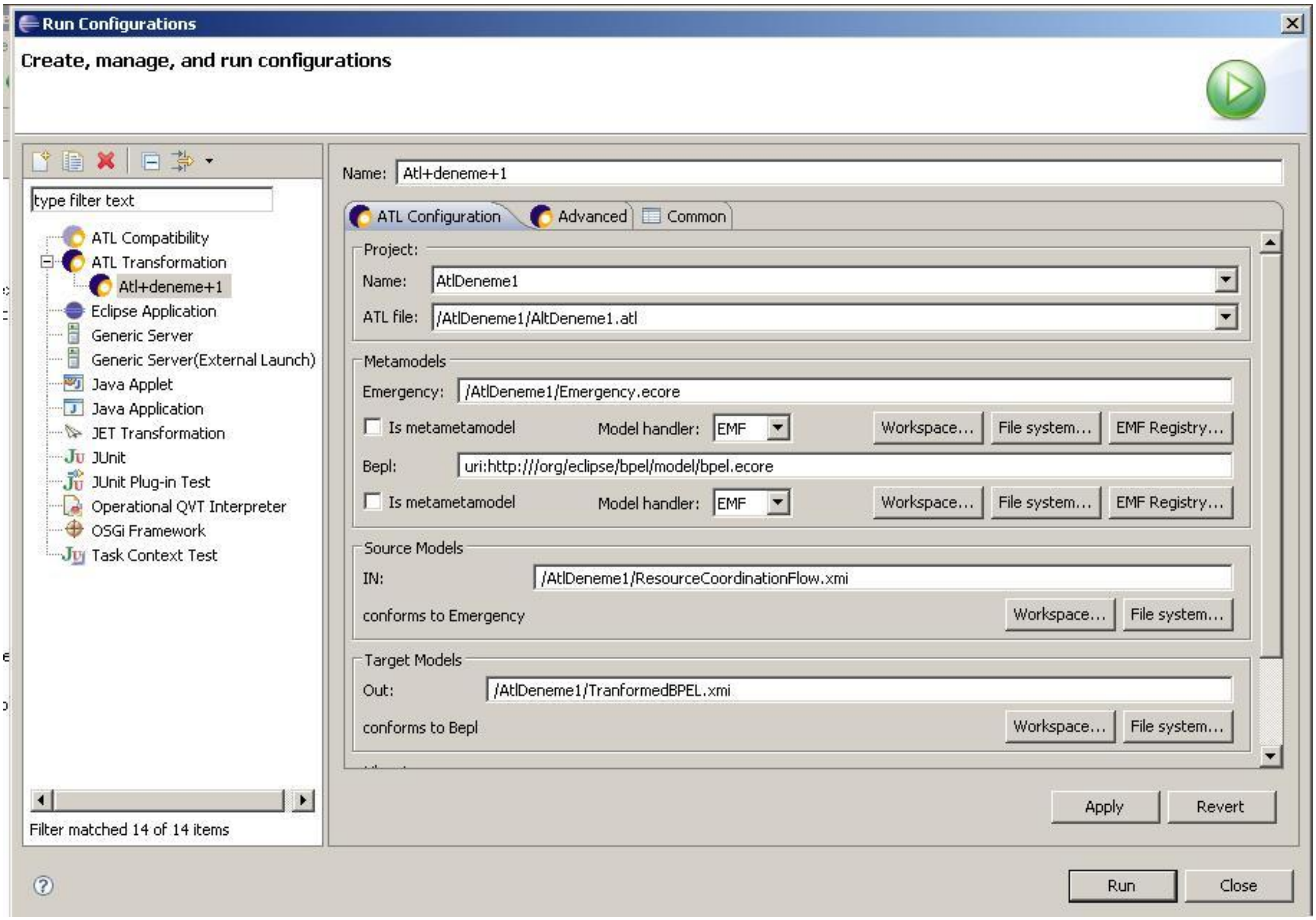

Figure 10 ATL Project

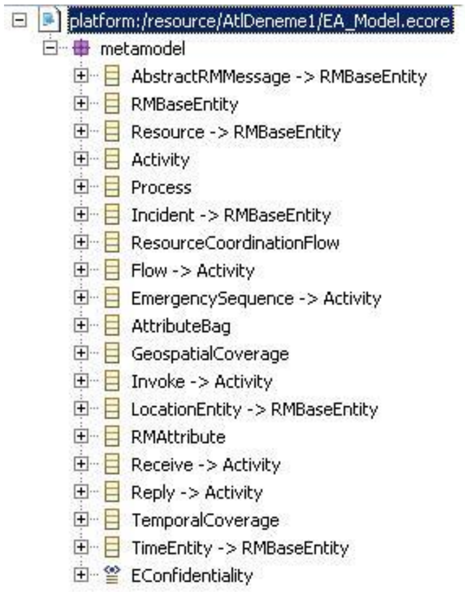

Figure 11 Our system Ecore

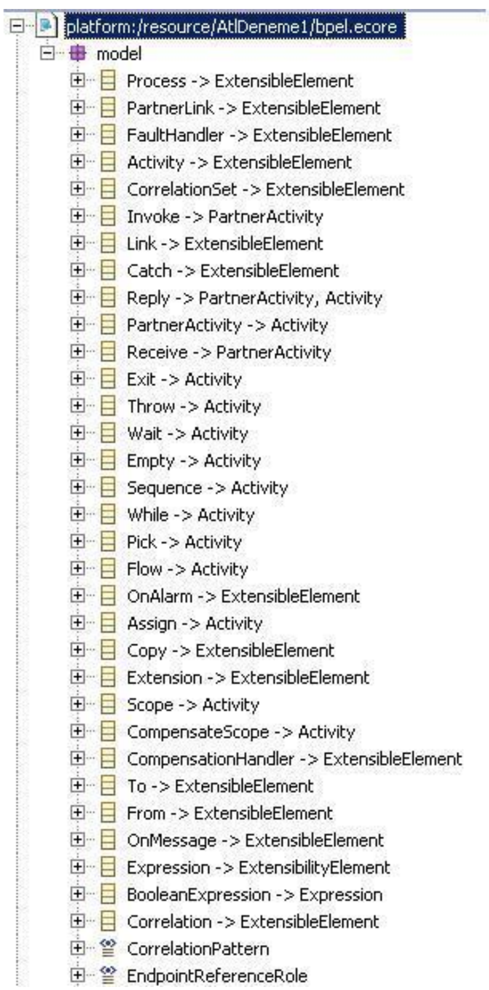

Figure 12 BPEL Ecore

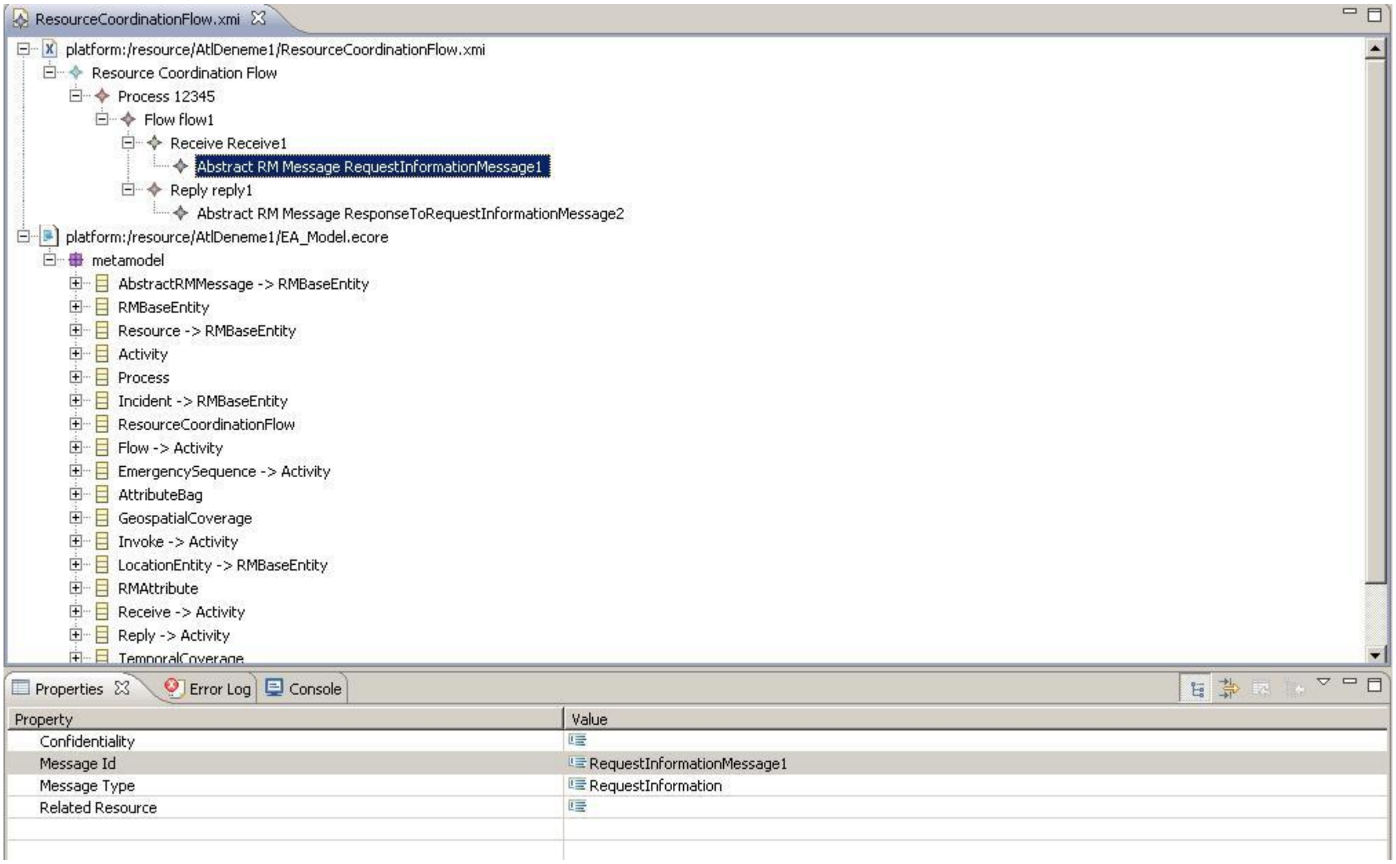

Figure 13 Example Model in xmi

Model2Text 1

```
import org.eclipse.emf.common.util.EList;

public interface AbstractRMMessage extends RMBaseEntity {
        String getMessageId();
        void setMessageId(String value);
        EList<Resource> getRelatedResource();
        EConfidentiality getConfidentiality();
        void setConfidentiality(EConfidentiality value);
        TemporalCoverage getTemporalCoverage();
        void setTemporalCoverage(TemporalCoverage value);
        GeospatialCoverage getGeospatialCoverage();
        void setGeospatialCoverage(GeospatialCoverage value);
        String getMessageType();
        void setMessageType(String value);

} // AbstractRMMessage


import org.eclipse.emf.ecore.EObject;
public interface Activity extends EObject {
        String getActivityId();
        void setActivityId(String value);
        String getActivityName();
        AbstractRMMessage getMessage();
        void setMessage(AbstractRMMessage value);

} // Activity


import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;
public interface AttributeBag extends EObject {
        EList<RMAttribute> getAttribute();

} // AttributeBag

import org.eclipse.emf.common.util.EList;
public interface EmergencySequence extends Activity {
        EList<Activity> getActivity();

} // EmergencySequence

import org.eclipse.emf.common.util.EList;
public interface Flow extends Activity {
        EList<Activity> getActivity();

} // Flow

import org.eclipse.emf.ecore.EObject;
public interface GeospatialCoverage extends EObject {
        LocationEntity getLocation();
        void setLocation(LocationEntity value);
} // GeospatialCoverage
public interface Incident extends RMBaseEntity {
        GeospatialCoverage getGeospatialCoverage();
        void setGeospatialCoverage(GeospatialCoverage value);
        TemporalCoverage getTemporalCoverage();
        void setTemporalCoverage(TemporalCoverage value);
```

```
        String getIncidentId();
        void setIncidentId(String value);

} // Incident

public interface Invoke extends Activity {
} // Invoke

public interface LocationEntity extends RMBaseEntity {
} // LocationEntity

import org.eclipse.emf.ecore.EFactory;
public interface MetamodelFactory extends EFactory {
        MetamodelFactory eINSTANCE = metamodel.impl.MetamodelFactoryImpl.init();
        AbstractRMMessage createAbstractRMMessage();
        RMBaseEntity createRMBaseEntity();
        Resource createResource();
        Activity createActivity();
        metamodel.Process createProcess();
        Incident createIncident();
        ResourceCoordinationFlow createResourceCoordinationFlow();
        Flow createFlow();
        EmergencySequence createEmergencySequence();
        AttributeBag createAttributeBag();
        GeospatialCoverage createGeospatialCoverage();
        Invoke createInvoke();
        LocationEntity createLocationEntity();
        RMAttribute createRMAttribute();
        Receive createReceive();
        Reply createReply();
        TemporalCoverage createTemporalCoverage();
        TimeEntity createTimeEntity();
        MetamodelPackage getMetamodelPackage();

} //MetamodelFactory

import org.eclipse.emf.ecore.EObject;
public interface Process extends EObject {
        String getProcessId();
        void setProcessId(String value);
        Incident getContext();
        void setContext(Incident value);
        Activity getActivity();
        void setActivity(Activity value);

} // Process

public interface Receive extends Activity {
} // Receive

public interface Reply extends Activity {
} // Reply

public interface Resource extends RMBaseEntity {
} // Resource
```

```java
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;
public interface ResourceCoordinationFlow extends EObject {

	EList<metamodel.Process> getProcesses();

} // ResourceCoordinationFlow



import org.eclipse.emf.ecore.EObject;
public interface RMAttribute extends EObject {

	String getAttributeName();
	void setAttributeName(String value);
	String getAttributeValue();
	void setAttributeValue(String value);

} // RMAttribute

import org.eclipse.emf.ecore.EObject;
public interface RMBaseEntity extends EObject {
	AttributeBag getAttributes();
	void setAttributes(AttributeBag value);

} // RMBaseEntity

import org.eclipse.emf.ecore.EObject;
public interface TemporalCoverage extends EObject {
	TimeEntity getTime();
	void setTime(TimeEntity value);

} // TemporalCoverage

public interface TimeEntity extends RMBaseEntity {
} // TimeEntity


import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import org.eclipse.emf.common.util.Enumerator;
public enum EConfidentiality implements Enumerator {
	UNCLASSIFIED(0, "Unclassified", "Unclassified"),
	RESTRICTEDTO_COMMUNITY(1, "RestrictedtoCommunity", "RestrictedToCommunity"),
	RELEASABLE_TO_PUBLIC(2, "ReleasableToPublic", "ReleasableToPublic"),
	SECRET(3, "Secret", "Secret"),
	TOP_SECRET(4, "TopSecret", "TopSecret");
	public static final int UNCLASSIFIED_VALUE = 0;
	public static final int RESTRICTEDTO_COMMUNITY_VALUE = 1;
	public static final int RELEASABLE_TO_PUBLIC_VALUE = 2;
	public static final int SECRET_VALUE = 3;
	public static final int TOP_SECRET_VALUE = 4;
	private static final EConfidentiality[] VALUES_ARRAY =
		new EConfidentiality[] {
			UNCLASSIFIED,
			RESTRICTEDTO_COMMUNITY,
			RELEASABLE_TO_PUBLIC,
			SECRET,
			TOP_SECRET,
```

```
		};

	public static final List<EConfidentiality> VALUES = Collections.unmodifiableList(Arrays.asList(VALUES_ARRAY));
	public static EConfidentiality get(String literal) {
		for (int i = 0; i < VALUES_ARRAY.length; ++i) {
			EConfidentiality result = VALUES_ARRAY[i];
			if (result.toString().equals(literal)) {
				return result;
			}
		}
		return null;
	}
	public static EConfidentiality getByName(String name) {
		for (int i = 0; i < VALUES_ARRAY.length; ++i) {
			EConfidentiality result = VALUES_ARRAY[i];
			if (result.getName().equals(name)) {
				return result;
			}
		}
		return null;
	}
	public static EConfidentiality get(int value) {
		switch (value) {
			case UNCLASSIFIED_VALUE: return UNCLASSIFIED;
			case RESTRICTEDTO_COMMUNITY_VALUE: return RESTRICTEDTO_COMMUNITY;
			case RELEASABLE_TO_PUBLIC_VALUE: return RELEASABLE_TO_PUBLIC;
			case SECRET_VALUE: return SECRET;
			case TOP_SECRET_VALUE: return TOP_SECRET;
		}
		return null;
	}
	private final int value;
	private final String name;
	private final String literal;
	private EConfidentiality(int value, String name, String literal) {
		this.value = value;
		this.name = name;
		this.literal = literal;
	}
	public int getValue() {
	  return value;
	}
	public String getName() {
	  return name;
	}
	public String getLiteral() {
	  return literal;
	}
	@Override
	public String toString() {
		return literal;
	}

} //EConfidentiality
```