UEML: Providing Requirements and Extensions for Interoperability Challenges

Maria Bergholtz (1), Paul Johannesson (1), Petia Wohed (2)

1. Department of Computer and Systems Sciences
Stockholm University / The Royal Institute of Technology
Forum 100, 164 40 Kista, Sweden
{maria, pajo}@dsv.su.se

2. Centre de Recherche en Automatique de Nancy,
CRAN, Université Henri Poincaré, Nancy 1/CNRS
BP239, 54506 Vandoeuvre les Nancy, France
petia.wohed@cran.uhp-nancy.fr

Abstract. Addressed in this paper is the domain of interoperability and interoperable information systems. Enterprise analysis and modelling is an essential tool for achieving interoperability. However, the big number and diversity of enterprise modelling tools and techniques available today challenges the usefulness of enterprise modelling. Addressing this problem, a Unified Enterprise Modelling Language is under development. The aim of this language is to function as a bridge between different enterprise modelling techniques and tools facilitating model translation and hence support model exchange and communication. The work presented here is an analysis of UEML and its further development. A number of requiems are identified and extensions to UEML for capturing these requirements are proposed.

1. Introduction

The growth of the number of information systems developed and in operation, as well as the development of techniques like the web facilitating communication and data exchange during the last couple of decades, has raised the need for developing interoperable information systems. By interoperable information systems is usually meant information systems which not only can operate in isolation for a single organisation's needs, but which are open and can also interact and function in co-operation with systems outside the organisational unit for which they were initially developed or procured [16].

As this puts focus on the technical aspects and a lot of solutions, e.g., communications and data exchange standards were developed, the attention paid on interoperable information systems has been broadened into the more general concept of interoperability. While interoperable information systems refer to the capability

of information systems for physically receiving and processing data from outside and sending data in required format to external sources, interoperability refers to interoperable information systems imbedded in well integrated environments. For achieving this, integration on an organisational level where different actors' businesses and processes are interplayed was focused on.

As enterprise modelling was a well recognized and widely applied technique for enterprise analysis, attention was paid on the communication problems caused by the big number and diversity of enterprise modelling techniques and focused was on "unification" or model translation and exchange between these. Diverse initiatives for facilitating enterprise model exchange were undertaken.

One of these initiatives was the UEML project[1], the goal of which was the development of a Unified Enterprise Modelling Language (UEML), which should act as an *Interlingua* between different enterprise modelling techniques and tools. For the development of UEML, three enterprise modelling languages, GRAI, IEM and EEML, were cross-analysed and a shared meta-model for them built [17]. In this meta-model, only concepts which appeared in at least two of the analysed languages were included. The choice of the analysed languages, GRAI, IEM and EEML, was not definite but only constituted a first step of the work. The development of the UEML language is therefore not considered to be completed. On the contrary, the intention is to further develop it by gradually analysing and adding new languages into it. Addressed in the work presented here are hence extension areas of UEML. The analysis is provided through identification of new requirements for UEML. We have followed the main principle that a construct shall only be added into UEML language if it appears in at least two other modelling languages.

The presentation starts by first introducing UEML. After that a number of new requirements are identified and discussed. For every requirement examples of the languages supporting it is presented and a possible extension of UEML suggested. The analysis is summarised by a presentation of an extended UEML meta-model.

2. UEML

In this section the UEML language is briefly introduced. The intention has been to keep the description as close to the original as possible. Many of the definitions are, therefore, directly copied from [18]. The meta-model is presented in Figure 1.

An Activity represents a generic description of a part of enterprise behaviour that produces outputs of a set of inputs. An Activity may be decomposed into other Activities. An Activity may require one or several ResourseRoles played by Resources. An Activity has at least one InputPort and at least one OutputPort, to which flows representing inputs or outputs of the activity are connected.

[1] UEML IST-TN 2001 34229, www.ueml.org

There are two ways to represent the fact that Resources are used in an Activity. It is either through the definition of a role (i.e. ResourceRole), that a Resource plays in an Activity (which is meant to be used when the origin of the Resource is not explicitly represented), or through a Flow, connected to the InputPort of the Activity, which carries the Resource (this is meant to be used when the origin of the Resource is explicit or if the Resource to be used is the result of some decision or grouping or decomposition of some other Resource(s) through a ConnectionOperator).

A Flow represents the flowing of an object from one origin to a target. The origin and target of a Flow are Anchors that can be InputPorts, OutputPorts or ConnectionOperators. Furthermore, a Flow is an IOFlow, a ResouceFlow or a ControlFlow.

- An IOFlow represents the flowing of an Object between two activities. If the Object is an input, then the IOFlow is connected to an InputPort of an Activity and this means that the Object is needed for the Activity to be executed. The Object can possibly be consumed or modified by the Activity. If the object is an output of an Activity the IOFlow is, connected to the activity's OutputPort and this means that the Object has been produced by the Activity.
- A ResourceFlow represents the flowing of a Resource between two Activities. The Flow then connects an OutputPort of an Activity that produces it and an InputPort of the Activity that requires it.
- A ControlFlow represents either the precedence relationship between Activities (i.e., a ControlFlow without carried Objects); or the triggering of an Activity after another (i.e., an TriggerFlow, carrying an InformationObject that represents the event triggering the second Activity); or the flowing of an InformationObject that is used to constraint the execution of the Activity (i.e., ConstraintFlow carrying a constraining InformationObject such as e.g. a description of a procedure to be followed to execute the Activity).

As mentioned above, an Anchor is an InputPort, an OutputPort, or a ConnectionOperator. An InputPort represents the entry of a Flow in an Activity. An OutputPort represents the exit of a Flow in an Activity. A ConncetionOperator represents the grouping or splitting (Join and Split) of Flows between two Activities. A ConnectionOperator of type "Join" is target of at least two Flows and is origin of exactly one Flow. A ConnectionOperator of type "Split" is origin of at least two Flows and is target of exactly one Flow.

Furthermore, a ResourceRole defines a Role played by a Resource in an Activity. A Resource may be a MaterialResource or a HumanResource.

Finally, an Object is anything that can be attached to a Flow. In other words it is anything that may be needed or produced by an Activity. It can be an InformationalObject or a Resource.

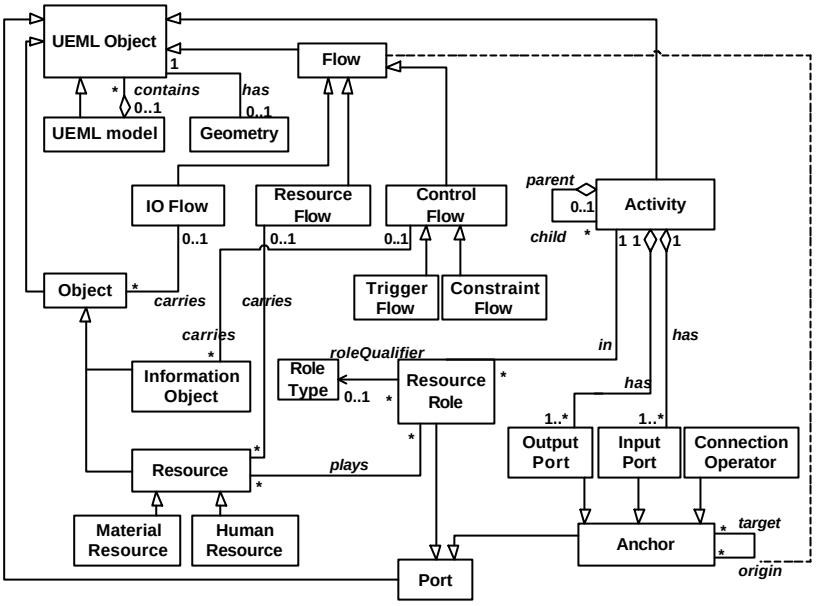

Fig. 1. UEML meta-model (reprinted from [18])

3. New requirements and extension areas for UEML

In this chapter a number of new requirements are presented. The basic reason for each requirement is that it introduces concepts that occur in several other enterprise modelling languages. These concepts should therefore also be included in UEML. As motivation for every requirement a number of languages supporting it are given. The requirements are divided into three groups: dynamic aspects, modelling aspects, and social/institutional/organisational aspects.

3.1 Dynamic Aspects

In this sub-section we consider requirements for dynamic aspects, in particular how to handle events and different kinds of activities.

3.1.1 Distinguishing between different kinds of activities

UEML should distinguish between two kinds of Activities. The concept of Task (i.e. an atomic, not further decomposable unit of work) shall be distinguished from the concept of Process (a unit of work consisting of a number of steps, where each step

is either a task or a sub-process which in turn can be decomposed into a number of steps). Tasks and Processes have several properties in common, e.g., they both have inputs and outputs and someone responsible for their execution. Therefore, the class Activity, capturing these common properties, shall be used as a generalization of them (see the shaded classes in Figure 2). The reason for distinguishing Tasks from Processes is the basic difference in that a process orchestrates a number of activities (i.e. shows the control flow between these), while a task does not (but is still important to introduce as it represents units of work). Note that this also implies that the aggregate relationship from class `Activity` to itself can now be more accurately drown from class `Process` to class `Activity`.

The distinction between atomic and compound activities is made in several languages. In UML [14] *Action* and *Activities* are used for atomic and compound activities, respectively. In BPMN [4] the constructs *Task* for atomic activity is separated from the constructs of *Process* and *Sub-process* for representing compound activities. In BPEL4WS [3] the constructs used are *Basic Activity* and *Process* and in YAWL [1] *Task* is distinguished from *Composite Task* and *Process*.

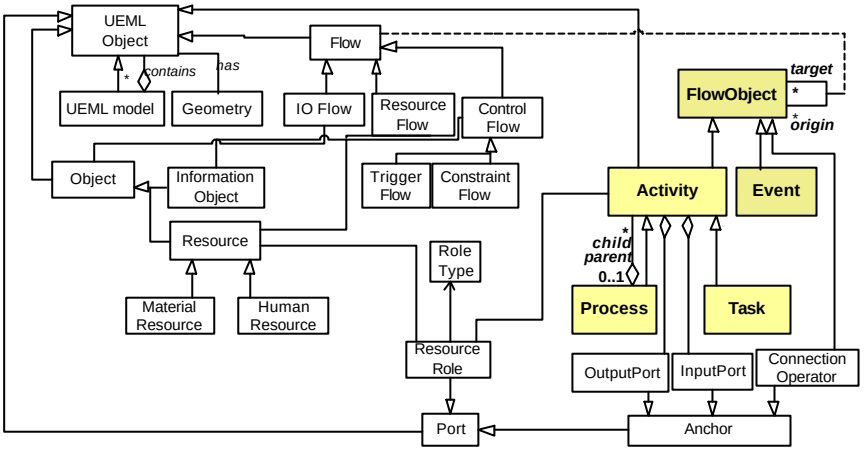

Fig. 2. Extension of UEML with dynamic concepts such as Process, Task and Events

3.1.2 Capturing events

UEML should include an Event concept (see the shaded and doted classes Figure 2). An Event is something that happens outside the system but that has effects on the system. Hence, an Event is something that (from the perspective of the system) happens instantaneously, as apposed to an Activity, which has duration. An Event is triggered by a source outside, while an Activity is executed by resources within the system. Examples of Events are timeouts and the receiving of a message. When an Event occurs it affects the control flow of the process. For example, an external agent sending a message, e.g. a purchase order, into the system would result in a

number of steps being executed, e.g. for processing and delivering the purchase order. A timer signalling a timeout would often result in a reminder sent or in an interruption and, possibly reset, of the process. The concept of Event is present in several process modelling languages: BPMN [4] (*Event with Message or Timer triggers*), ebXML/BPSS [6] (*Responding/Requesting Activity*), AOR [20] *(External Events* as opposed to *Internal Events*).

UEML should, therefore, be able to distinguish between instantaneous activities (i.e., Events) happening outside the system and Activities with duration within the control of the system. Furthermore, as Events influence the execution of the system, it should be possible to define and model the effects of an event as well as their relationship with Activities. Hence, likewise Activities, Events are concepts that can and shall be choreographed (sequenced, forked, etc.) in the process flow. This similarity motivates the introduction of the concept FlowObject as a common generalisation of Events and Activities in the model, see Figure 2. FlowObjects are associated to class Flow. In addition, class ConnectionOperator should be a specialisation of FlowObject. This view is in accordance with the one taken by BPMN.

Note that the suggested extension makes explicit the concept of Event. It was actually mentioned in the definition of a TriggerFlow in the UEML specification, i.e. "A ControlFlow connects directly or indirectly two Activities and represents either […], the triggering of an Activity after another (TriggerFlow, eventually carrying an InformationObject that represents the **event** triggering the second activity), or [...]." [18]. However, it was not made explicit there.

3.2 Modelling Aspects

In this sub-section we consider requirements concerning general modelling constructs, in particular type levels and control flow constructs.

3.2.1 Capturing advanced control flow constructs

UEML should support a wide variety of control flow constructs. It should not only support basic control flow constructs (like those of sequence, parallel split, synchronisation, merge, exclusive choice, and multi choice) as described by the Workflow Management Coalition [21] and supported by the majority of enterprise modelling and workflow management tools. It should also support more advanced control flow constructs as identified and described in the workflow patterns [2, 22].

An example of a more advanced control flow construct is Deferred Choice. There is a distinction between Decisions and Deferred Choices. A Decision operator can be seen as a question that will be answered based on process data. In contrast, "A Deferred Choice operator represents a branching point in the process where the alternatives are based on events that occur at that point in the process rather than the evaluation of expressions using process data" [4]. The Deferred Choice is hence

implicit, which is different from a Decision, which represents an explicit choice. An example where the Deferred Choice concept is needed is when a sales process waits for a customer confirmation. Either a message with a confirmation will arrive from the customer or a time-out will occur. The process proceeds in different ways in these two cases and the actual choice is deferred to runtime. Examples of process languages that support Deferred Choice are: YAWL [1] (*Deferred Choice*), BPMN [4] (*Event Gateway*), BPEL4WS [3] (*Pick*).

Another example of an advanced control flow construct is the presence (creation, management and possibly synchronization) of Multiple Instances running in parallel for the same task within one and the same case. For instance[2], for an insurance claim, a number of witnesses need to be interviewed. The number of witnesses is different from case to case (i.e. not know at design time). It may not even be known at run-time when the initial interview task starts as new witnesses may appear as the result of already running interviews. Some recent languages supporting the concept of arbitrary multiple instances are YAWL [1] and BPMN [4]. Therefore, UEML should support the run time creation and synchronisation of arbitrarily many Multiple Instances.

3.2.2 Capturing knowledge and operational levels

UEML should support the modelling of concepts on both the knowledge and operational levels. The *operational level* models concrete tangible individuals in a domain, while the *knowledge level* models information structures that characterise categories of individuals at the operational level [13]. An example of the distinction between knowledge level and operational level is the one given by Martin and Odell [11], who employ the concept of power type to refer to the difference between knowledge level and operational level. A *power type* is a class whose instances are subtypes of another class. A UEML example can be taken for the class Resource. This class contains concrete tangible individuals such as a particular car with registration number ABS125, a particular agent with name Stephen Dahl, etc. A class Resource Type added to the model would then model the different categories of cars, e.g. trucks, wagons etc and different kinds of agents, e.g. teachers, students, graduate students, etc. Examples of approaches that support the distinction between knowledge level and operational level are REA [12], UMM (BRV) [6], Fowler's analysis patterns [8], and Coad's Components [5].

[2] This example is taken from the Workflow Patterns Homepage, http://tmitwww.tm.tue.nl/research/patterns/documentation.htm

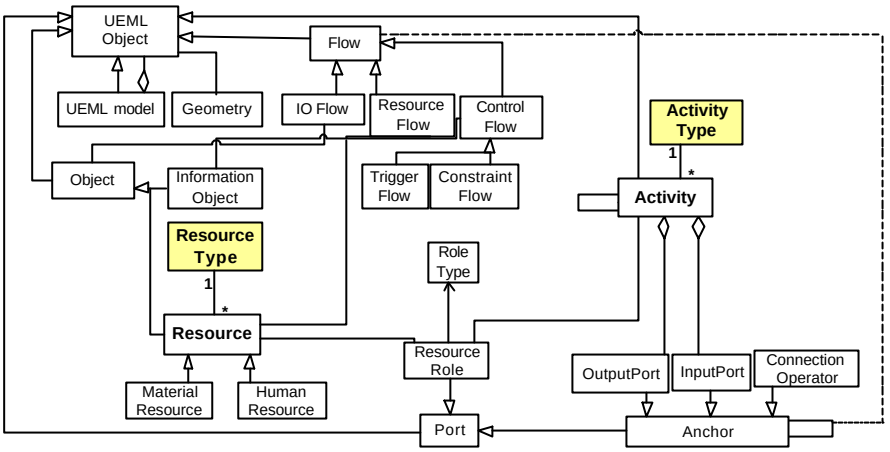

Fig. 3. Examples of explicitly distinguishing between classes at knowledge and operational level in UEML

Most UEML classes today are on the operational level only. These could be split so that they are represented at the knowledge level as well. See Figure 3, where such distinction is suggested for the classes Resource and Activity.

- Resource should be associated to a corresponding ResourceType class, in order to be able to model both individual resources and describe the distinctions or constraints that apply to certain types of resources. Example: properties of class ResourceType may be isbn-number, author, publisher, title, year etc. The corresponding properties of Resource are owner, id-number, place-in-library, condition etc.
- Class Activity should be associated with a corresponding ActivityType in order to model categories of activities as well as individual activities. Example: class ActivityType may contain properties like terms that may initiate or terminate an activity, delay-allowed or not, description, priority, etc. The corresponding properties of class Activity are responsible, id-number, delay-time, start-time, end-time, priority, state, etc.

3.3 Social/Institutional/Organisational Aspects

Human actions may be viewed as taking place in three different worlds; the physical world, the communicative world, and the social/organisational/institutional world. In the ***physical world*** people carry out physical Activities – they carry goods, utter sounds, wave their hands and send electronic messages. In the ***communicative world*** people express their intentions and feelings – they tell other people what they know and they try to influence the behaviour of others by communicating with them.

Communicative Activities are performed through Activities in the *physical world*. In the ***Social/Institutional/Organisational*** world people change the social and institutional relationships among them – people become married, people acquire possession of property. People change social and institutional relationships by performing Activities in the *communicative world*. In this sub-section we consider requirements for social/institutional/organisational aspects, in particular how to handle agents, values, and social relationships.

3.3.1 Capturing Agents

UEML should be able to distinguish between active and passive objects. UEML should include the Agent concept for capturing active objects as well as the Passive Objects as the opposite of the Agent concept, see Figure 4. Agents can perform Activities, exercise rights and have responsibilities and obligations towards other Agents. Passive Objects, on the other hand, cannot perform Activities on their own accord, nor can they possess rights versus other Resources or have duties towards them. Passive Objects are used in and acted upon in Activities. They may be physical objects or social objects like Contracts and Commitments.

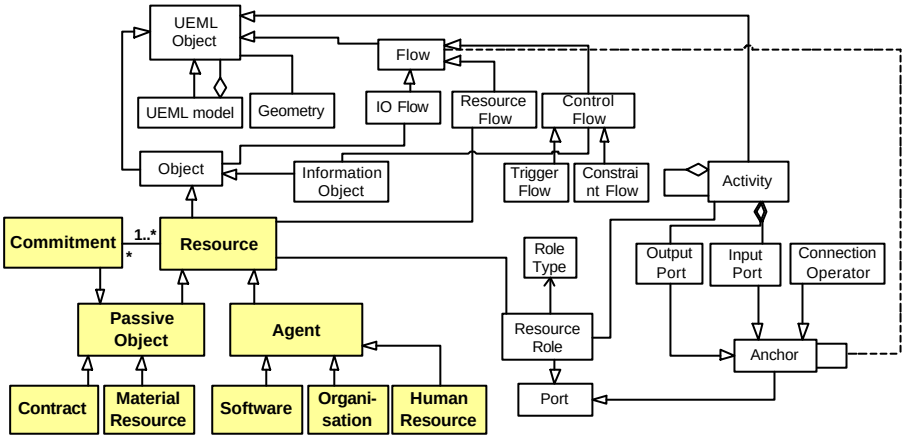

Fig. 4. Extension of UEML with Agent and Passive Object concepts

Furthermore, UEML should make explicit what kinds of Agents there exist. Human agents, Organisations, and artificial agents, e.g. Software, are some examples, but several can possibly be identified and added accordingly. The Human-Resource concept, which is present in UEML 1.0 as a specialisation of the Resource concept, will then only be one of several specialisations of the Agent concept (see Figure 4). The MaterialResource concept, which is the other Resource specialisation in UEML 1.0, will become a specialisation of the Passive Object concept, as it

partially captures it. Other examples of Passive Objects are Commitment and Contract. They are further explained in the following subsection.

Examples of approaches where the distinction between active and passive objects is explicit are AOR [20] and FRISCO [7].

3.3.2 Capturing social effects

UEML should include business oriented concepts including economic, value, and service aspects. I.e.,

- UEML should include a Commitment concept (see Figure 5). A Commitment is an obligation between two or more Agents, i.e. that of performing some kind of Activity. An example of a Commitment is a promise from a Supplier to a Customer to deliver an agreed upon quantity of goods. The corresponding Commitment on the Customer's behalf is to pay the Supplier. Examples of ontologies that include the Commitment concept are REA[12], AOR [20] and UMM (BRV) [19].

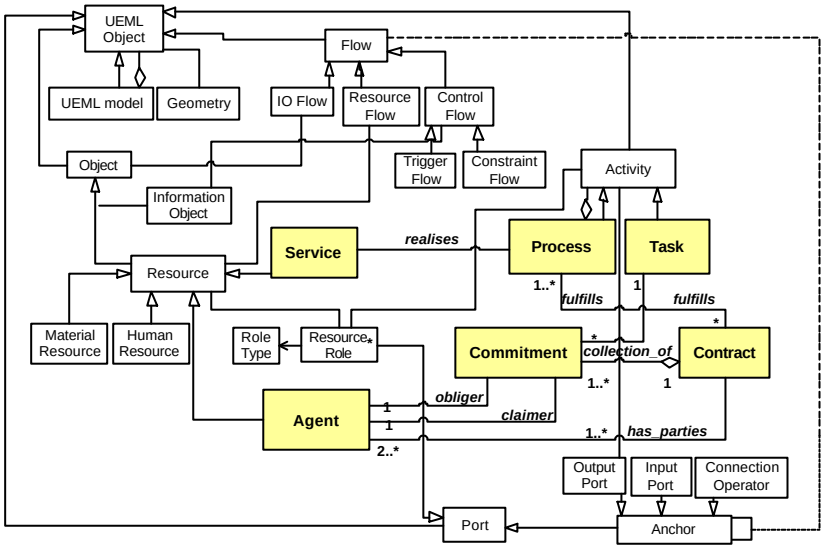

Fig. 5. Capturing social effects in UEML

- UEML should include a Contract concept. A Contract is an aggregation of Commitments. An example of a Contract is an order which aggregates a number of order lines. Each order line represents two reciprocal Commitments between Agents. Examples of ontologies that include the Contract concept are REA[12] and UMM (BRV) [19].

- UEML should include a Service concept. A Service provides value to one or several Agents and is realised through a Process. A Service may be realised through several different Processes and a Process may realise many Services. For instance, both Amazon.com and Ebay.com provide the service of supplying books online. This is however achieved by different processes. This requirement was inspired by the approach presented in [10].

UEML should relate the suggested social concepts with existing UEML concepts. By including these concepts, UEML will be able to bridge to business modelling languages and ontologies such as Gordijn's e3 [9] and Pigneur's and Osterwalder's ontology [15].

4. Summary

In this paper addressed were extensions of Unified Enterprise Modelling Language. A number of requirements for the language were identified and extensions for fulfilling these requirements were proposed. In order to justify the work, for every requirement examples of approaches supporting it were presented. The final result is shown in Figure 6. In this UEML meta-model, the suggested extensions are highlighted by shading the corresponding classes.

The requirements were divided into three groups. The first group, focusing on the dynamic aspects, resulted in the introduction of the class Event, as well as the separation of the concept of Task from this of Process.

The second group of requirements focused on the modelling aspects. Extension areas for modelling advanced control-flow constructs were suggested. They are however not, for the moment, graphically shown in Figure 6. This would require a rather detailed discussion and falls therefore outside the scope of this paper. Furthermore, a requirement for explicit distinguishing between knowledge and operational level was discussed. This is in the model exemplified by separating the classes Activity Type and Resource Type from the classes Activity and Resource.

The third group of requirements focused on capturing Social/Institutional/Organisational aspects in enterprise modelling. Firstly, distinguishing Agents carrying out activities, from Passive Objects, i.e. artefacts on which activities are carried out on, was proposed. After that the attention was turned into Contracts, Commitments, and Services, as they constitute an important part of the Social/Institutional/Organisational reality we are in.

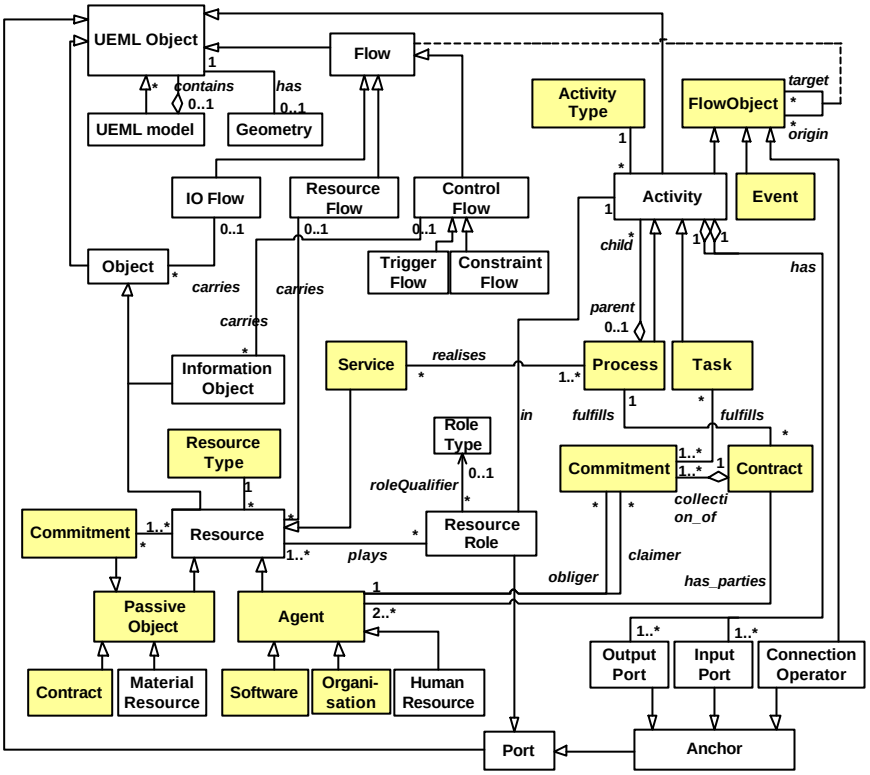

Fig. 6. The extended UEML meta- model

References

1. W. van der Aalst & A. ter Hofstede, Yet Another Workflow Language (YAWL), http://www.citi.qut.edu.au/yawl, Valid on July 2004
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns", *Distributed and Parallel Databases*, 14(3), pages 5-51, July 2003, also accessible at the Workflow Patterns homepage, http://tmitwww.tm.tue.nl/research/patterns/ (December 2004)
3. T. Andrews, et al, Business Process Execution Language for Web Services, Version 1.1, 5th May 2003, available at http://www.ibm.com/developerworks/library/ws-bpel/
4. BPMI.org, Business Process Modeling Notation (BPMN), Version 1.0 – May 3, 2004, downloadable from www.bpmi.org
5. P. Coad, E. LeFebvre, J. De Luca, *Java Modeling In Color With UML: Enterprise Components and Process*, Prentice Hall, 2000

6. ebXML Deliverables, Electronic Business eXtensible Mark-up Language (ebXML),Valid on 20030328, http://www.ebxml.org/specs/index.htm
7. E.D. Falkenberg et al, FRISCO – A Framework of Information System Concepts, Technical Report, ISBN 3-901882-01-4, IFIP, 1998
8. M. Fowler, *Analysis Patterns: reusable object models*, Addison Vesley, 1996
9. J. Gordijn's homepage, url: http://www.cs.vu.nl/~gordijn/research.htm, (accessed December 2004)
10. M. Klein and A. Bernstein, "Searching for Services on the Semantic Web Using Process Ontologies", in proc. of the *1st International Semantic Web Conference (ISWC)*, 2002
11. J. Martin, J. Odell, *Object-Oriented Methods. A Foundation*, Prentice Hall 1994
12. W.E. McCarthy , "REA Enterprise Ontology", Valid on 20040419, http://www.msu.edu/user/mccarth4/rea-ontology/
13. W.E. McCarthy, "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment", *The Accounting Review*, 1982
14. OMG, UML 2.0 Superstructure Specification. OMG Draft Adopted Specification, ptc/03-08-02, August 2003, http://www.omg.org/cgi-bin/doc?ptc/2003-08-02
15. A. Osterwalder's homepage, url: http://www.hec.unil.ch/aosterwa/article.php?sid=12 (accessed December 2004)
16. A. M. Ouskel and A. Sheth, "Semantic Interoperability in Global Information Systems: A brief introduction to the research area and the special section", *SIGMOD Record Special Issue on Semantic Interoperability in Global Information Systems*, vol 28, no 1, March, 1999
17. H. Panetto and G. Berio and K. Benali and N. Boudjlida and M. Petit, "A Unified Enterprise Modelling Language for enhanced interoperability of Enterprise Models", in *Proc. of the 11th IFAC INCOM2004 Symposium*, Bahia, Brazil, April, 2004.
18. Unified Enterprise Modelling Language (UEML) project homepage, url: www.ueml.org (accessed December 2004)
19. UN/CEFACT Modeling Methodology (UMM-N090 Revision 10), Valid on 20040419, http://webster.disa.org/cefact-groups/tmg/doc_bpwg.html
20. G. Wagner, "The Agent-Object Relationship metamodel: towards a unified view of state and behaviour", *Information Systems*, 28(5), pp 475-504, 2003
21. WfMC. "Workflow Maganement Coalition Terminology & Glossary", Document Number WFMC-TC-1011, Document Status – Issue 3.0. Technical report, Workflow Management Coalition, Brussels, Belgium, February, 1999
22. Workflow Patterns homepage, http://tmitwww.tm.tue.nl/research/patterns/ (December 2004)