Extending UML for Real-Time Component Based Architectures

José L. Fernández-Sánchez
Escuela Técnica Superior de Ingenieros Industriales
C/ Jose Gutierrez Abascal 2
28006 Madrid - Spain
Phone: 34 91 3363146
Fax: 34 91 5011202
e-mail: jlfdez@ingor.etsii.upm.es

Antonio Monzón
TCP Sistemas e Ingeniería
C/ López de Hoyos 327
28043 Madrid - Spain
Phone: 34 91 7489872
Fax: 34 91 7489876
e-mail: amonzon@tcpsi.es

Abstract

A new style for concurrent object oriented architectures was proposed by the author in his doctoral thesis report [7]. The style can be used in object oriented systems when individual paths of execution are required to be concurrent and multiple processes may be positioned along the path to control the action. Therefore each process controls a segment of the path sequentially. So he called the style PPOOA, 'Pipelines of Processes in Object Oriented Architectures [8].

PPOOA promotes that architectural decisions related to concurrency in real-time systems should be made at the preliminary design phase.

Currently UML [12] is the standard notation for object oriented development. An important issue of UML is its capacity to be extended by using the profiling concept that allows the compatibility with other UML modeling issues. A UML profile is defined as a stereotype of package that groups model elements that have been customized for a specific domain. It allows one to develop variants of the UML suited for such specific domain [3].

The PPOOA profile described in this article is compatible with UML and improves its use in real-time architectures. The UML stereotypes are extended with the elements of the style (periodic and aperiodic processes, controller object, and coordination mechanisms).

The modeling of flows of activities with a textual but rigorous notation is also critical for the analysis of the time responsiveness in real-time systems.

The PPOOA profile has been implemented in a CASE tool as part of the research work of the CARTS project funded by the European Union, IST initiative.

Keywords
Software Architecture, Component Based Development, Real-Time Systems, UML, CASE Tools

1. Limitations of UML for Architecting Real-Time Systems

Current CASE tools are supporting UML as the universal notation for object oriented software development.

This general applicability is the cause of some limitations that difficult UML use in modeling real-time component based software architectures. Real-time systems are those where time responsiveness is a critical issue so a real-time architecture model requires the assessment of its time responsiveness properties typically in three times of the development cycle: after preliminary design or architecture phase, when coding is complete and during system testing.

The main limitations of UML regarding the real-time systems architecture modeling are:

- Not explicit support of the "design component" as a UML building element. The component concept in UML is considered from the implementation perspective.
- The Notation used for behavior description, for example sequence and collaboration diagrams, is not suitable for responsiveness assessment techniques, such as Rate Monotonic Analysis (RMA) [10]. The reason is their emphasis in interactions or messaging instead of activities or actions, as they are called in RMA.
- Some building elements necessary in real-time architectures such as aperiodic processes or coordination mechanisms are not considered as part of UML metamodel.

2. UML Extension Mechanisms

UML provides extension mechanisms to be used when either the methodologist or CASE tool builders require additional features beyond those defined in the UML standard.

UML extension mechanisms are stereotypes, constraints and tagged values. The three extension mechanisms can be used either separately or together. A UML building element may have only one stereotype, but many constraints and tagged values.

A stereotype is a semantic feature attached to an existing model element. Stereotypes not only classify model elements at the object model level, but they facilitate the addition of "virtual" UML metaclasses with new metaattributes and semantics. The other extension mechanisms allow users to attach additional properties and semantics directly to individual model elements, as well as to model elements classified by a stereotype. It is important to make clear that stereotypes may extend the semantics, but not the structure of preexisting elements.

Any modeling element may have arbitrary information attached to it, such as management information, code generation information or additional semantic information, in the form of a property list consisting of tag-value pairs. A tag represents the name of an arbitrary property with the given value. For example, preconditions and postconditions attached to operations in a class are examples of standard tagged values.

Constraints, as well as tagged values, are used to extend the semantics of model elements classified by that stereotype. Constraints can be attached to classes or objects, restricting in this way the semantics of these elements. The constraints must be observed by all model elements marked with that stereotype.

3. PPOOA Vocabulary

The PPOOA vocabulary of building elements is composed of Components, Coordination mechanisms and Casual Flows of Activities (See figure 1).

Each of these vocabulary elements has several metamodel attributes (See figures 3 and 4), that allow the architecture time responsiveness assessment using analytical method such as previous mentioned RMA.

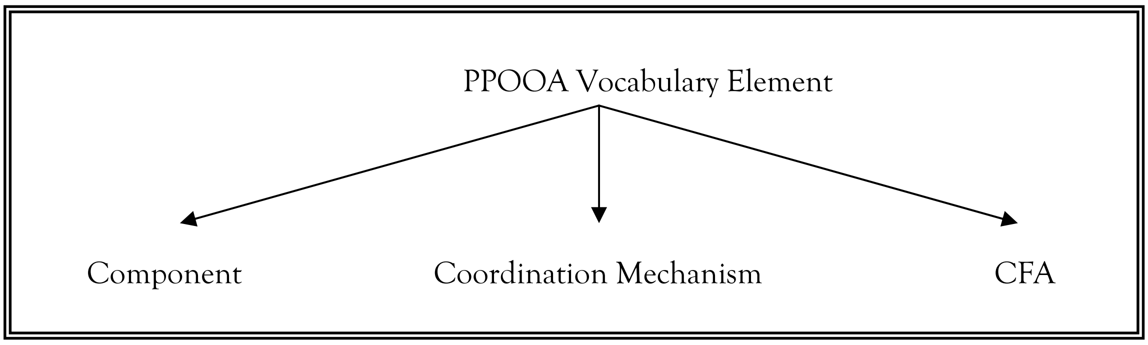

Figure 1. PPOOA Vocabulary

3.1 PPOOA Components

A PPOOA component is a conceptual computation entity that performs some responsibility and may provide and require interfaces to other components. Generally, it may be decomposed into small granularity parts.

Some of the components proposed are already considered in UML metamodel but others require UML metamodel extension.

Figure 2 shows the PPOOA vocabulary of components. A short description of each component type is following.

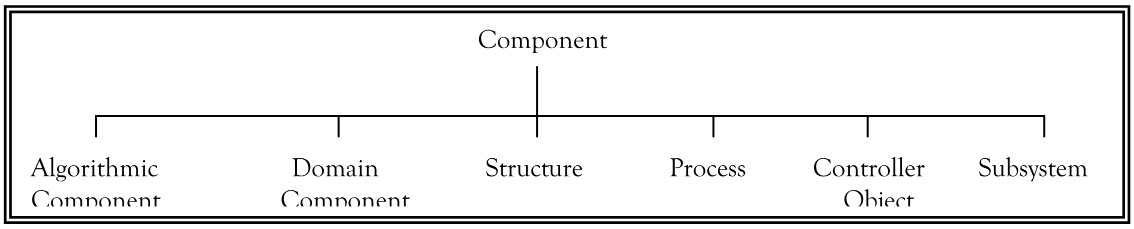

Figure 2. PPOOA Vocabulary of Components

- Algorithmic Component. Algorithmic components or utilities are elements of the architecture that perform calculations or transform data from one type to another but separated from its structural abstraction [1]. Data classification components, Unix filters or data processing algorithms typically represent them.

  The algorithm component in PPOOA is similar to the utility defined in the UML metamodel but the algorithmic component may have instances.

- Domain Component. The domain component is an element of the architecture that responds directly to the modeled problem. This component does not depend on hardware or user interface [4]. Its instances are similar to the classic object in object oriented design.

- Process. The process is a building element of the architecture that implements an activity or group of activities that can be executed at the same time as other processes. Its execution can be scheduled.

  The PPOOA vocabulary supports two different types of processes: periodic and aperiodic. Each of them is described in PPOOA profile and their real-time attributes are also considered (See figure 3).

- Structure. A structure is a component that denotes an object or class of objects characterized as an abstract state machine or an abstract data type.

  Typical examples are: stack, queue, list, ring and others defined by Booch [1].

- Controller Object. The controller object is responsible for initiating and managing directly a group of activities that can be repetitive, alternative or parallel. These activities can be executed depending on a set of events or conditions [15].

  Typically, a controller object receives an event. An event is something that can happen, like a Posix signal, a hardware interrupt or it represents a computed event, like an airplane entering a forbidden region or an alarm.

  Where one of these events occurs, the system schedules associated event handlers.

  A controller object manages two things: the dispatching of handlers when the event is fired, and the set of handlers associated with the event. The application can query the set and add or remove handlers.

- Subsystem. A subsystem is a component that clusters other components of the architecture based on the minimizing coupling and enhancing visibility criteria. Subsystems are the components of what is called Development View in the 4+1 View model [11].

  Subsystems have the following characteristics:

  - Logic coherence. The supplied services are related to one another logically.

  - Independence. Subsystems can be implemented as independent programs.

  - Simple interfaces. Subsystems communicate with each other through simple and well defined interfaces [14].

  The coupling reduction and the supply of simple interfaces are achieved in subsystems considering the design pattern "“Façade" [9]. A façade supplies only one view of the subsystem that is useful for its clients.

3.2 PPOOA Coordination Mechanisms

A coordination mechanism is an essential element of the proposed profile. Its classification and categorization have been an essential part in PPOOA development.

A coordination mechanism provides the capabilities to synchronize or communicate components. Synchronization is the blocking of a process until some specified condition is met. Communication is the transfer of information between components.

As part of the development of the PPOOA style, fourteen coordination mechanisms were analyzed [5] and a taxonomy was developed [6]. Of those mechanisms, the following have been considered as part of the profile:

- Bounded Buffer
- General-Semaphore
- Mailbox
- Transporter
- Rendezvous

3.3 Causal Flow of Activities (CFA)

CFA represents causality flows of activities that can be superimposed to the architecture representation describing in this way behavior patterns. Other authors reference them as timethreads [2].

The capacity to execute manually or by means of animation these CFAs or its analytic assessment is very important for comparing the different design alternatives.

Although CFAs are settled externally to the architecture components, they have to be outlined internally in the architecture components. This is when the activity concept appears. The activity is a computation block where no decision of assigning resources is taken inside it. The decision of assigning resources starts with scheduling points such as priority changes, other process invoking, initialization or termination of an I/O operation or a process abortion.

In general, there can exist several active CFAs at a moment and they can also interact with each other. It is also possible that two instances of the same CFA coexist. When a new instance of the CFA starts, it continues through a series of cause-effect elements until it finishes and it ignores other external stimuli at the starting point. Generally, a new stimulus will create a new instance of the CFA.

The CFA instances can stop at certain waiting points, where there is coordination with other instances. At these points, coordination mechanisms or component instances with concurrent access are located.

CFAs can split into parallel flows of activities; flows of activities can also merge together.

As already mentioned, the triggering stimulus of a CFA instance is called an “event”.

Events are generally external to the system, but they can also be internal or timer [10].

Possible external events are the pressing of a key, the arrival of a message from another system or the arrival of data from an external device.

An internal event takes place when an internal state of the system changes.

A timer event is caused by the flow of time. For example, if the reading of a sensor is required to occur every 100 ms., a timer event can trigger it.

An important aspect to consider is the identification of internal events. The architecture CFAs are built according to the following consideration: an event is identified as a new internal event when its arrival pattern or response time are different from the triggering event of the original CFA. Consequently, such internal event is considered the origin of a new CFA. A typical example of this situation is the detection of an alarm situation.

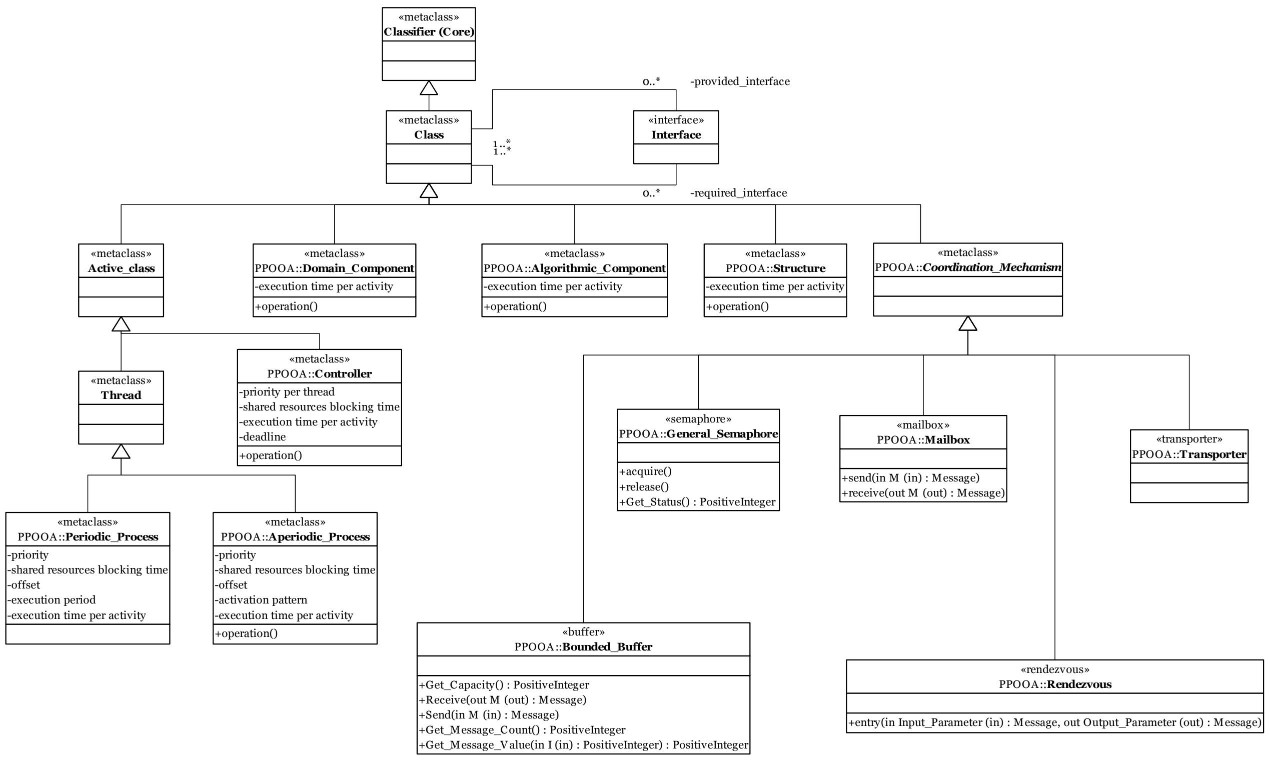
«metaclass» Classifier (Core)
«metaclass» Class
«interface» Interface
0..* -provided_interface
0..* -required_interface
1..*
1..*
«metaclass» Active_class
«metaclass» PPOOA::Domain_Component
-execution time per activity
+operation()
«metaclass» PPOOA::Algorithmic_Component
-execution time per activity
+operation()
«metaclass» PPOOA::Structure
-execution time per activity
+operation()
«metaclass» PPOOA::Coordination_Mechanism
«metaclass» Thread
«metaclass» PPOOA::Controller
-priority per thread
-shared resources blocking time
-execution time per activity
-deadline
+operation()
«metaclass» PPOOA::Periodic_Process
-priority
-shared resources blocking time
-offset
-execution period
-execution time per activity
«metaclass» PPOOA::Aperiodic_Process
-priority
-shared resources blocking time
-offset
-activation pattern
-execution time per activity
+operation()
«semaphore» PPOOA::General_Semaphore
+acquire()
+release()
+Get_Status() : PositiveInteger
«mailbox» PPOOA::Mailbox
+send(in M (in) : Message)
+receive(out M (out) : Message)
«transporter» PPOOA::Transporter
«buffer» PPOOA::Bounded_Buffer
+Get_Capacity() : PositiveInteger
+Receive(out M (out) : Message)
+Send(in M (in) : Message)
+Get_Message_Count() : PositiveInteger
+Get_Message_Value(in I (in) : PositiveInteger) : PositiveInteger
«rendezvous» PPOOA::Rendezvous
+entry(in Input_Parameter (in) : Message, out Output_Parameter (out) : Message)

Figure 3. Extension of UML Metamodel including PPOOA Vocabulary

4. PPOOA Metamodel. An extension of UML Metamodel

The UML Classifier is extended with other building elements that are characteristic of the PPOOA vocabulary. This is shown in the PPOOA Building Elements metamodel (Figure 3). A Classifier is a model element that describes behavioral and structural features. The Class element is a kind of Classifier that represents a concept within the system being modeled. It describes both the data structure and the behavior of a set of objects. A Class may provide or require interfaces.

In the PPOOA Building Elements metamodel different types of classes are considered (Figure 3): Active_Class, Domain_Component, Algorithmic_Component, Structure and Coordination_Mechanism. Real-time domain independent attributes are depicted for each building element. These attributes are essential for applying time responsiveness assessment techniques.

An Active_Class is a UML element. It is a class whose instances are active objects. Two kinds of Active_Class are considered: Thread and Controller.

- Threads are treated in a special way, as they are classified in periodic and aperiodic. The Periodic_Process and the Aperiodic_Process are PPOOA metaclasses. As shown in figure 3, several attributes such as priority, shared resources blocking time, offset or execution time per activity, have to be defined for each of the processes. The execution period in the periodic processes and the activation pattern in the aperiodic processes are also essential attributes that have to be determined during the architectural modeling of a real-time system.

- The Controller is a PPOOA element that is responsible for initiating and managing directly a group of activities that can be repetitive, alternative or parallel. Certain attributes have to be defined: priority per thread, shared resources blocking time, execution time per activity and deadline.

Figure 3 shows how coordination mechanisms are a specialization of the “class” UML concept. New stereotypes have been created in PPOOA for these coordination mechanisms: <<semaphore>>, <<mailbox>>, <<rendezvous>>, <<transporter>>. These stereotypes will allow different iconic representations for these mechanisms.

UML metamodel is also extended to include the CFA modeling element.

A CFA (Causal Flow of Activities) is a cause-effect chain of activities that cuts across different elements of the architecture. This chain progresses through time and executes the activities provided by the system components until it gets to an ending point.

As we can see in the CFA metamodel (Figure 4), a CFA consists of one or more Path_Elements. These Path_Elements are mapped to Classifiers. Therefore, Classifiers (Figure 3) implement Path_Elements.

Activities are part of the response associated to the Triggering_Event. The UML Partition is a generalization of Activity. A Partition divides the states of an activity graph into groups, helping in this way to allocate properties and actions among the states [13]. An activity is the smallest part in which a response can be divided and it can consist of one or more activities. In terms of RMA, activities are RMA actions. It is important to make clear that the computations that take place in an RMA activity cannot cause changes in the system resources allocation.

Finally, a Continuation_Element represents a location in the CFA where multiple path elements can connect together in a non-sequential way satisfying a continuation rule or guideline. The PPOOA Continuation_Element is a type of the UML State_Vertex. In UML, a vertex is defined as a source or target for a transition in a state machine [13].

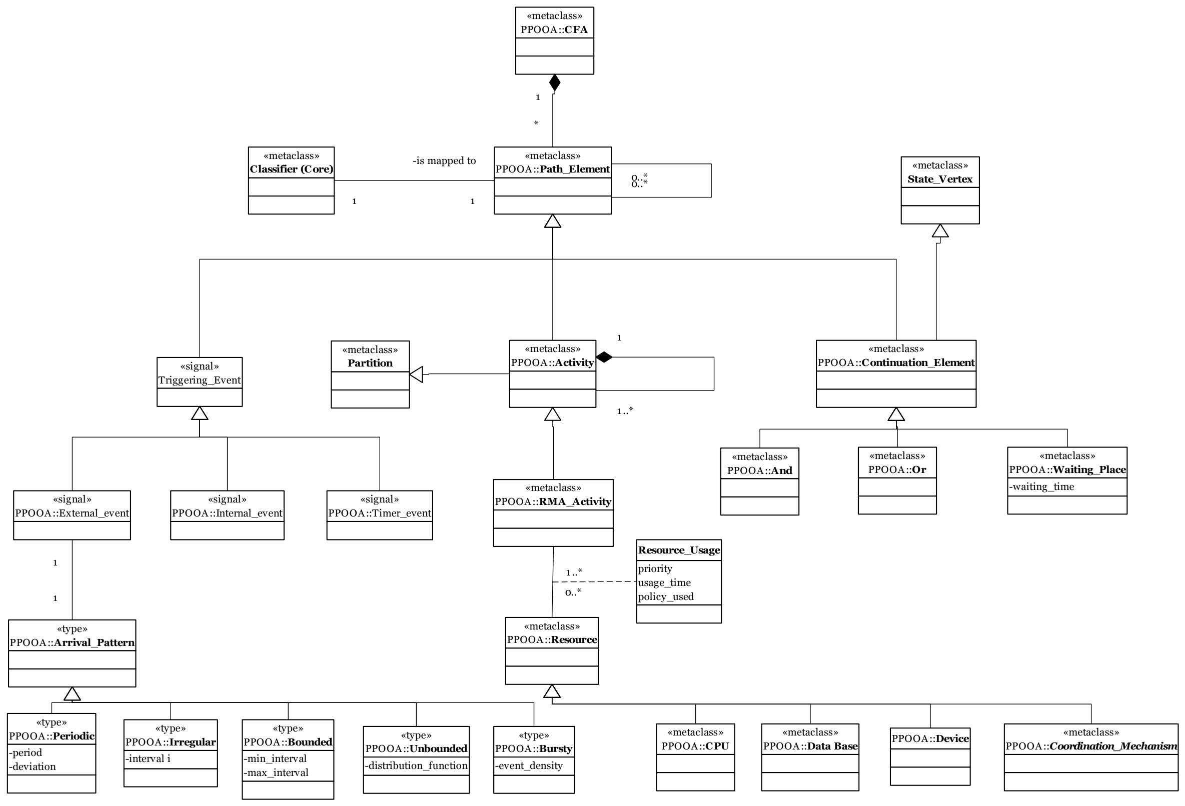

Figure 4. Extension of UML metamodel to include CFA

5. Implementing PPOOA Metamodel in CARTS CASE tool

A common practice in SW Engineering is to use a tool to support all the methodological issues involved in the systems modeling activity. In particular, diverse CASE tools have been developed in the last years to support UML notation. These tools allow representing (both graphically and logically) all the UML abstractions very precisely, as well as to exploit the different modeling diagrams described by UML.

Existing UML tools make no or very weak emphasis on the Engineering Process behind the construction of a system. In fact, an expensive training and mentoring are needed to take advantage of current UML CASE tools.

In Real-Time Systems modeling, a diversity of tools cover the detailed design and code generation, but with no architecting support and these tools are mainly oriented to the construction phase. Furthermore, they keep the previously mentioned deficiencies of UML to properly represent real-time systems.

5.1 CARTS Tool Metamodel

In addition to PPOOA Metamodel, new modeling concepts have to be described to implement the CASE tool. These concepts are considered in the tool-specific metamodel.

The basic element contained in this metamodel is the "Modeling Element" that generalizes all the modeling concepts involved in PPOOA vocabulary and the additional UML standard modeling concepts needed to create the conceptual design of a system using PPOOA architectural style. This item derives from the UML Core concept "Element".

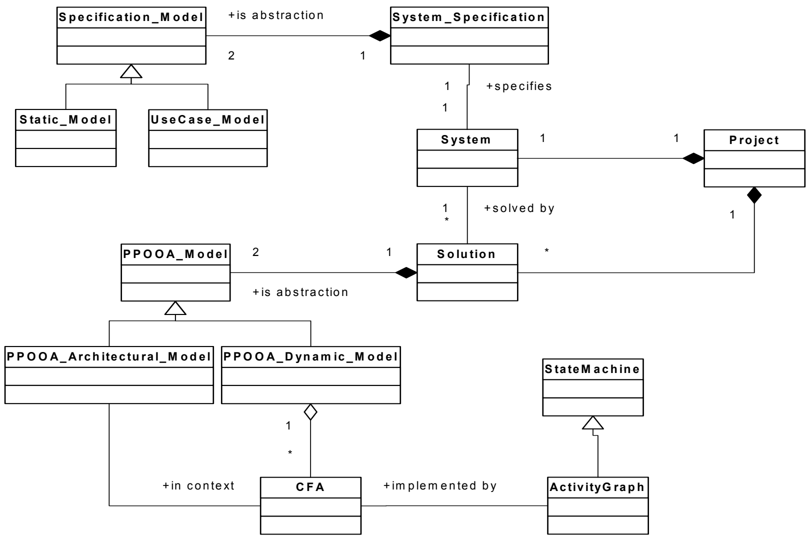

Figure 5. Tool Metamodel: Main Concepts

Figure 5 shows the main involved concepts and their relationships at a high level of abstraction. The concept "System" represents the modeled system. On the other hand, "Solution" represents all the possible design alternatives to implement the specified system. The "System" is specified through the "System_Specification", which is abstracted through the "Specification_Model". Two standard specification models have been used: "Static_Model" (Class Diagrams) and "UseCase_Model".

In the "Solution" side, other two models are used (described in previous sections). The only aspect worth to mention is that CFA's shall be represented using UML Activity Graphs.

The main Organization Elements are described in Figure 6. The concept "Repository" represents a common workspace (apart of each particular project workspace) to share standard or user-defined "Projects", "Packages" and "Model_Elements" to be shared among different "Users", and the architecture assessment tools, as well.

5.2 CARTS tool features

The features supported by the tool can be classified in the following categories:

- Graphic features: The tool supports several diagrams to model the problem description and the optional designs for the system modeled. All the diagrams share the same standard behavior of typical CASE tools. Specific diagrams supported are:
  - UML Class and Use Cases Diagrams: First one represents the system structure and second the expected system behavior.
  - UML Activity Diagrams: This modeling facility is used to represent CFA's through a common and sharable notation.
  - PPOOA Architectural View.
  - PPOOA CFA's Diagrams.
- Tool management features: In this group are included all those features related with the administration of the data contained in a project repository and its exploitation in a typical SW development process.
  - Configuration management (versions, access permissions, baselines, etc.)
  - Metadata Import/Export through XMI.
  - Basic report generation.

- Engineering features: Special tool features to help software architects while making decisions, both during the architecting process, and the whole development lifecycle.
  - RMA Assessment Module: This module provides different useful parameters to assess the proposed architecture and to select one or several of the different alternatives according to time responsiveness criteria.
  - Traceability Matrixes: This feature allows to show inherent design links between different modeling elements, and to analyze the impact while changing individual elements

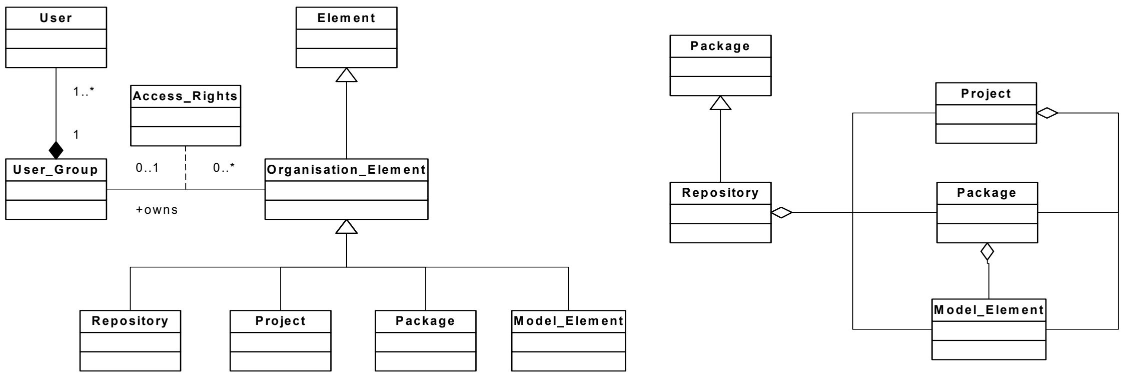

Figure 6. Tool Metamodel - Organisation Elements

5.3 Architectural assessment

One of the main features of an architecting tool is the possibility to evaluate a possible solution to design a system, according to specific quality criteria, required for the system.

This assessment is only possible based on the use of a coherent set of modeling elements to describe the system structure and behavior, enhanced with realistic time-critical attributes of the system elements.

6. Conclusions

The software architecture, the main components and how they will be interacting (dynamic view), must be defined during the preliminary design phase of development. We propose an architectural style "PPOOA" to model the architecture and even evaluate rigorously its time responsiveness properties at this phase of the development.

The style can be used in object oriented systems when individual paths of execution are required to be concurrent and multiple processes may be positioned along the path to control the action.

The proposed initial assessment may not be as accurate as it will be after the coding of components when time execution can be measured precisely, but we considered very useful to compare responsiveness of solution alternatives found in preliminary design. A more precise evaluation can be performed after coding, using the same techniques, and at testing. Testing is always necessary, but it is a costly and often tedious job.

7. References

[1] G. Booch. *Software Components with Ada*. Benjamin Cummings, Menlo Park, CA. 1987.

[2] R.J. Buhr. *Pictures that Play: Design Notations for Real-Time and Distributed Systems.* Software Practice and Experience. Vol 23 (5). August, 1993.

[3] T.Clark, A.Evans, S.Kent. *Using Profiles to Re-architect the UML*. ECOOP´2000. Sophia Antipolis, Cannes, France. June, 2000.

[4] P. Coad, D. North, M. Mayfield. *Object Models. Strategies, Patterns and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[5] J.L. Fernández. *A Taxonomy of Coordination Mechanisms Used in Real-Time Software Based on Domain Analysis*. CMU/SEI-93-TR-34. Software Engineering Institute. Pittsburgh, 1993.

[6] J.L. Fernández. *A Taxonomy of Coordination Mechanisms Used in Real-Time Processes*. ACM Ada Letters. March 1997.

[7] J.L. Fernández. *Arquitectura Software Genérica de Tiempo Real.* Tesis Doctoral. Junio, 1997 (in Spanish).

[8] J.L. Fernández. *An Architectural Style for Object Oriented Real-Time Systems*. Fifth International Conference on Software Reuse. Victoria (Canada), IEEE 1998.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading MA, 1995.

[10] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, M. González Harbour. *A Practitioner´s Handbook for Real-time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, Dordrecht, 1993.

[11] P. Krutchen. *The 4 +1 View Model of Architecture*. IEEE Software. November 1995.

[12] Object Management Group. *Unified Modeling Language Standard Version 1.3*. June, 1999.

[13] J.Rumbaugh, I.Jacobson, G.Booch. *The Unified Modeling Reference Manual.* Addison Wesley Longman, Reading, MA, 1999.

[14] I. Sommerville, R. Morrison. *Developing Large Software Systems with Ada*. Addison Wesley, Reading MA, 1987.

[15] R. Wirfs-Brock. *Stereotyping. A Technique for Characterizing Objects and their Interactions*. Object Magazine. November, 1993.