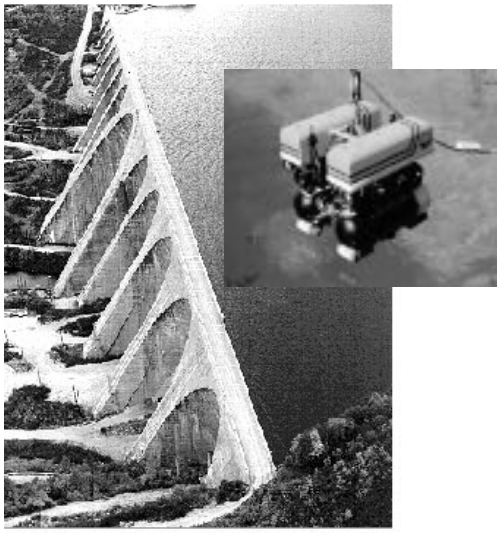
Vertex Project



IRIS-III

November 1997
Revised, March 1998

White Paper on Vertex

Denis Laurendeau & Denis Poussart
Vertex - Project Coordinators

Preamble

Organization of the document

Vertex involves a large number of people located at different places and participating in a complex project. In preparation for what may be envisioned as a very interesting challenge, it is imperative to present a set of guidelines that all team member should follow at both the technical and management levels.

This White Paper attempts to present a set of guidelines that, in our opinion, will help the Vertex team to *i)* meet its objectives, *ii)* allow a smooth integration of the different modules and *iii)* ease technology transfer between the universities and the industrial partners.

CHAPTER 1 presents a quick overview of Vertex's modules and sub-systems. This chapter is a basis for discussion and will certainly undergo major changes in the initial phases of the project. CHAPTER 2 presents guidelines relative to the technical aspects of *Vertex* while CHAPTER 3 covers management issues. The suggestions of team members are welcome on both these topics and on other matters that might have been overlooked. CHAPTER 4 disucsses the use cases for Vertex and is open for discussion by all team members.

CHAPTER 1 Overview of *Vertex*

Vertex stands for *Virtual Environments: from 3D Representations to Task Planning and Execution.*

The overall goal of *Vertex* is to design a system that allows to:

1. **simulate** complex **interactions** between a *site* and a group of *actors* (tools, etc);
2. **plan** the optimal sequence of operations in the complex interaction mentioned in 1.
3. **implements** the interactions in real-time on an actual site and with actual actors using the plan mentioned in 2.

Vertex is a complex system that brings together several fields in computer vision, robotics, human-machine interface and software/systems engineering. The design of a system of this complexity requires careful planning and sound engineering practices and methodology. CHAPTER 1 and CHAPTER 1 provide guidelines that should be followed through the duration of *Vertex* for the implementations of the various *Vertex* sub-systems.

From its user stand point, *Vertex* is designed to operate in two major modes:

Planning: where the environment provides resources which support a strategic view of the targeted task. Alternative scenarios can be explored, with tools for decision support, leading to the selection an *a priori* optimized scenario. This mode, which executes in a simulation context, operates with soft real-time constraints in order to provide effective human-machine interaction and realism.

Execution: where Vertex has direct physical linkage to the actual task undertaking, with a metaphor of teleoperation. Simulation resources are again put to task, but with hard real-time response and accurate physical modeling. This phase exploits tools for tactical decision support as well as “hands-on” telerobotics control.

The primary task which is targeted as a *Vertex* test case is the maintenance of underwater facilities, such as hydroelectric dams. However, the architecture of Vertex is to be as generic as feasible, so as to be applicable to a range of tasks where a sensitive context - hazard, danger, cost ... - requires detail strategic, tactical and execution integration.

Vertex is a large project within the IRIS-3 NCE programme, with a focus on the development of virtual environments to support both planning and execution of difficult tasks by remote systems. The tasks will form the large class of procedures required for the inspection and maintenance of large underwater facilities. This represents a very large socio-economic problem, especially when one considers the importance and cost of maintaining the thousands of hydroelectric facilities and related dams situated throughout Canada, and especially in the province of Quebec. Many of these facilities are critically in need of maintenance, yet their original plans and blueprints may be inaccurate (modifications may not have been fully documented) or they may be missing completely. Hydroelectric dams are extremely large and fairly complex structures. Some of them are now very old, and have begun to take on a life of their own, with sedimentary and biological deposits, base shifting, crack propagation, and fractures all occurring continuously.

Inspection and maintenance tasks need to be performed in fairly harsh environments, where it is not feasible to send humans to the site. Not only can the operations be dangerous; they can be quite costly. Shutting down a dam, or even a single large turbine can cost on the order of one million dollars per day in lost production of energy (and so minutes spared in procedural efficiencies can translate into thousands of dollars of

savings). Task rehersal and planning using the virtual environments provided by Vertex will have a very high payback.

We will be integrating tools from phase I and II of IRIS to build virtual environments (and virtualized interfaces) to support task execution and planning. In the early stages of IRIS, there was a clear segmentation of projects into Perception, Reasoning, and Action themes. They serve, not only as an architecture for autonomous systems, but also as a symmetrical model for the capacities of a human operator who typically will be at the heart of the loop of control. This provides a starting point for considering the design of HMI systems for teleoperative control.

Data will have to be integrated from several different sensing modalities, and presented in a way that can be easily understood by the human operator. Underwater vehicles can report their position and attitude, and can relay back images and sonar range data. Long core samples can be drilled into the concrete structures to examine the propagation of cracks by interpolating the cross-sections. Other telemetric sensors (position, vibration, geothermal) are routinely installed. As data is gathered over long periods of time, the models of the scene should also support the construction of behavioural descriptions of the structures and components, ranging from very long time scales to periods of short duration.

As these models are gathered over the thousands of sites that need to be monitored, there will also emerge a need to prioritize and schedule the inspection and maintenance activities. Concise and informative presentation of data will be needed to support this administrative burden.

Physical, Mechanical, and Behavioural descriptions need to be maintained at thousands of sites, and so it will be necessary to prioritize and schedule the activities at these locations. Concise and informative presentation of data will be needed to support this administrative burden. Indeed, this is also true for modeling, development of procedures, and execution of maintenance operations.

Once tasks have been scheduled, we will enter the phase of execution of remote procedures. For example, cracks may need to be followed while epoxy is injected or sediment and other deposits may need to be removed. These operations will be conducted through various Tele-Operative modes. Tasks will need to be analysed and decomposed into feasible blocks, then scheduled and executed. Each of these will be facilitated through the use of interactive virtual environments. We want to be able to

simultaneously display the current situation (from sensor data, images, and internal models) as well as predictive outcomes which might arise from certain actions. This is especially important for time-critical applications when the communications channel has limited bandwidth and therefore the network latency is high.

Scenario generation can be conducted within the simulated world before it is executed in real-time with its associated costs and hazards. Predictive displays will form the basis of interaction and planning with the simulated environment; in order to generate operational scenarios and evaluate their consequences as best as can be modeled (not just at the level of robot and tool modeling, but also at a high level where the coupling and interaction among various subtasks in an overall operational schedule). This will borrow heavily on work from operations research as well as modeling, simulation, visualization, and efficient human-machine interaction methods.

For complex tasks, there will be several types of users involved, people planning the operations, people performing the work, people supervising the work, and making strategic decisions. Each role may require different presentations of the information pertaining to the same field situation, and so it will be important to continually evaluate and refine the way that this information is being presented to the various users. It is critically important that we consider the perceptual, perceptual-motor, and cognitive capacities and constraints of the human operator when we design Human-Machine Interfaces.

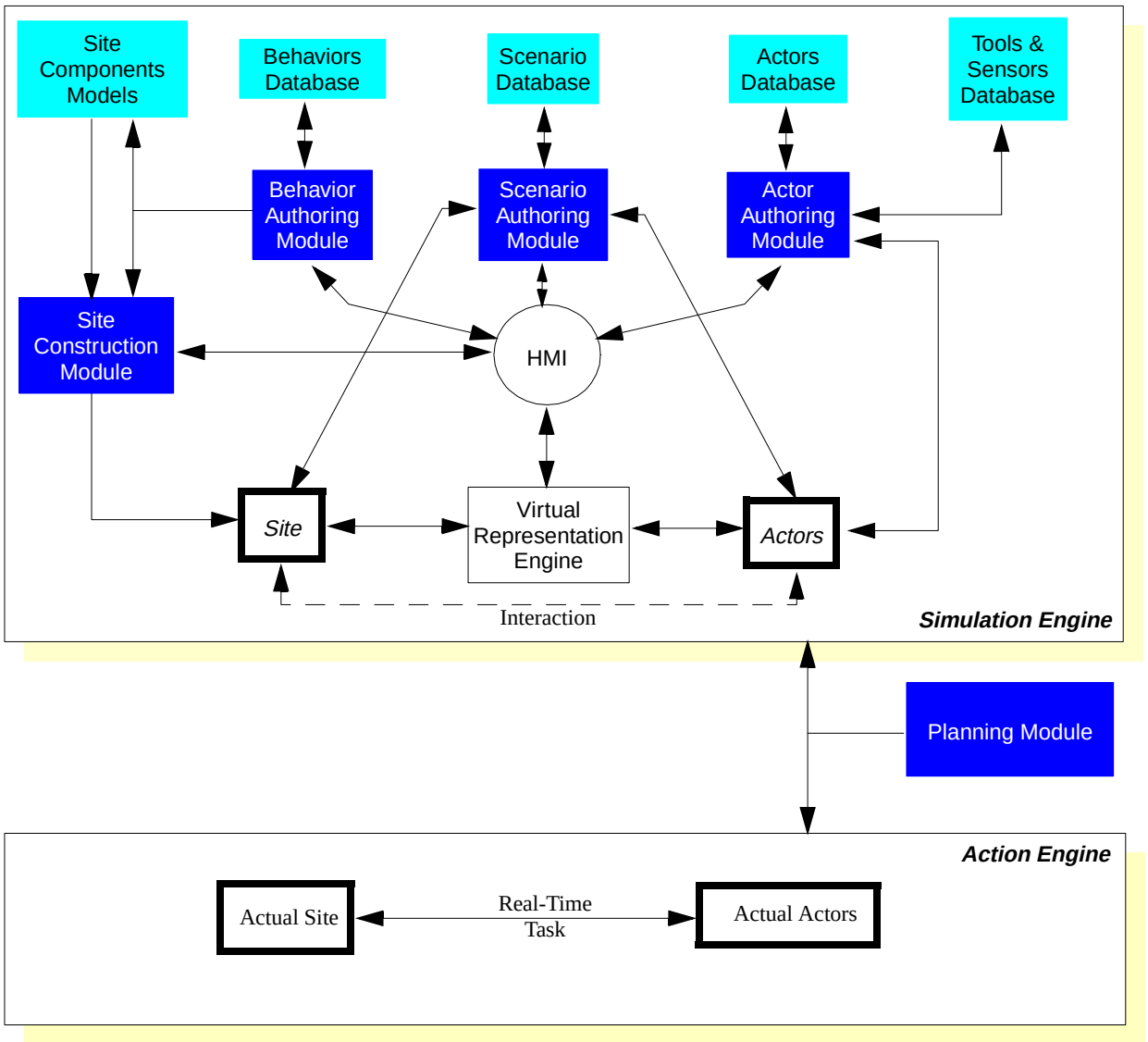
Considering the above context, Figure 1 is an attempt to describe the *Vertex* architecture in terms of modules and sub-systems. This description is by no means definite and should serve as the basis for discussion among team members.

Some of the important keywords in the Figure include:

Definition 1: site

collection of the objects of interest in a simulation or intervention. For instance, a dam, a control room. a penstock are sites which may be of interest for *Vertex*. As a class, it is a subset of the world.

FIGURE 1 Overview of *Vertex* architecture



Definition 2: **user**

individual or collective thereof using the system. For instance this entity encompasses a strategic planner, or an “officer-on-the-deck” tactician, or an operator who is telepiloting a robot through telepresence or exploiting *Vertex* in order to train for the task.

Definition 3: object

a component of a site. A site typically contains several components. Example of components may be a dam, or a Remotely Operated Vehicle (ROV) which includes tools which acts upon the dam. General (but not comprehensive) categories of entities include the scene, tools (“brawn”), computing engines (“brain”) and human-machine interfaces (“beauty”).

Definition 4: tool

resource used by a component to interact upon other components. For instance a drill can be used by the ROV to drill a hole in a dam.

Definition 5: sensor

a resource is used by a component to extract information which is relevant to its state, behavior or interaction with its surround. Such information may be external, for instance the distance of the ROV with respect to another object through sonar ranging or about its pose through inertia sensing. It may be internal, for instance the status of its battery charge.

Definition 6: model

a software construct that is used to describe properties of an object of a site. For instance geometry is described by a 3D model; colorimetric appearance is represented by a colorimetric / texture model.

Definition 7: behavior

A *behavior* is attached to an object. It models the action of an object in response to an intervention by another entity (tool, object, etc). For instance, a behavior can describe a mechanical or physical property (heat conduction, strength of material, elasticity, etc). One or more behavior can be attached to a model. Depending on the Vertex operating mode, behavior may have to be playable in “soft real-time”, “hard real-time”, or “faster than real-time (predictive)”.

Definition 8: actor

object or a tool that initiates a behavior on another object.

Definition 9: scenario

sequence of actions and behaviors that are triggered by actors. A scenario can also respond to probabilistic events under the control of the user.

A short description of the components in FIGURE 1 is as follows:

1. ***HMI (Human-Machine Interface)***: the HMI is responsible for all interactions between the user and *Vertex*. The HMI is equipped with a multimodal interface (visual, haptic, auditory...) which enhances a realistic and effective interaction with the system.
2. ***Actors***: actors are the components of the simulation that interact on the site. Actors can be equipped with tools and sensors stored in a ***Tools and Sensors Database***.

3. ***Actor Authoring Module***: the Actor Authoring Module allows to create actors that will interact with the Site. The edited actors are stored in an ***Actors Database***.
4. ***Site***: the component upon which the actors perform their interactions. The site can be a dam, a ship, a room, etc. An important characteristic of a site is that it can demonstrate a *behavior* of some sort.
5. ***Site construction module***: this module is responsible for the construction of the virtual site through the HMI. The module chooses scene components from the ***Models database***[1] and places these components in the scene. A behavior can be assigned to scene components
6. ***Scenario Authoring Module***: this module allows the editing of scenarios based on the behavior of each scene component and the behavior of the actors. The edited scenarios are stored in a ***Scenario Database***.
7. ***Behavior Authoring Module***: this module is responsible for assigning behaviors to scene components and actors. The available behaviors are obtained from the ***Behavior Database***.
8. ***Planning Module***: the planning module is responsible for optimal sequencing of operations in an interaction between the actors and the site. It uses the outcome of scenarios for the estimation of the best approach for performing a task on the site.

All the components described above form what is called the ***Simulation Engine***.

Vertex also contains the ***Action Engine*** which is responsible, through the HMI and with the help of the planning module, to conduct the actual interaction between the actual actors and the site of interest according to the plan that was established during the simulation phase.

The above description is incomplete and other utility modules will have to be added to *Vertex* in order to ensure its full functionality. The input of team members is welcome on this topic.

1. Such models are, in part, the output of VRES, an on-going Project at Laval (1997 - 1999) which targets tools and methods for the rapid production of Virtual Representations of Existing Sites. This Project deals with (mostly) static 3D geometrical modeling and also involves NRC, IREQ and Innovemetric Inc.

CHAPTER 2 Technical Issues

2.1 Software engineering approach (overall project)

Vertex involves the development and the integration of large software modules in each area covered by the Project. It seems that only a sound Object-Oriented software development process will allow a smooth integration of the research work performed by each sub-team. We thus propose to adopt the Booch OOA/OOD development process for the software implementation of *Vertex* [1]. In this approach, a project is divided in four steps at the macro-level:

1. establish core requirements;
2. model the behavior of the system (analysis);
3. create the architecture of the system (design);
4. evolve the implementation;
5. manage evolution (maintenance);

Steps 1 through 4 must be covered in *Vertex*. Step 5 is less significant since it applies to commercial software products.

A central tenet of Vertex is that its design and implementation adheres to a **user-centric approach**, i.e. that the user needs play a key role at all of the steps above.

In the start-up phase of *Vertex*, the team will have to establish clearly **what** *Vertex* should do and what it should **not** do. The proposal that was

submitted to IRIS (and the diagram shown in FIGURE 1 of CHAPTER 1) will serve as a starting point for this definition but it is clear that it is not a detailed enough document on which the project should be based. A Class Dictionary and Functional Points Dictionary is currently being developed and will be accessible for all team member to comment and improve.

Definition 10: Class dictionary

list of abstractions that are relevant to a problem and that can be described by a category in an object oriented language.

Definition 11: Functional points (or Use-case) dictionary

a functional point of a a system is a visible and verifiable behavior of the system when used by either a human operator or another software component. The overall behavior of the system is the collection of its functional points.

Once the requirements for *Vertex* are established, the strategy is to plan the architecture and design of the global system and to organize these modeling and design steps into a set of *releases* for the system, each release being an iterative and incremental improvement on functionality and structure of the previous one.

It is intended to schedule a new release every 6 months. Each release will serve as a constantly evolving demo of *Vertex* and will allow to detect the flaws in the architecture of the system and in the underlying algorithms. A 6 month deadline for each release may seem a difficult milestone to reach. It must be understood that early releases of the system will have very limited functionality (if any functionality at all) but will nevertheless demonstrate the actual status of the project. Such a release schedule is also a strong incentive for adopting sound software engineering practices in order to ensure smooth integration of new functionality to the existing system. It is also a good method to stimulate team members to keep focus

on their respective tasks as well as on the impact of these tasks over the entire project. Finally, the latest release can always serve as a demo.

Such a process of staged demos will help everyone involved maintain a focused view on the "big picture". Demos will also be a key driving force in the feedback loop of iterative design.

2.1.1 Analysis and design

The analysis and design of the system are two very important activities of *Vertex*. It is thus important to define guidelines describing each activity.

2.1.1.1 Global architecture for Vertex

Vertex software will be implemented using an Object Oriented approach. The architecture of the system including its class hierarchy, class diagrams, object diagrams, scenarios, and module diagrams will be designed using a top-down approach in order to define the **global architecture** of the system.

By global architecture, we imply the macroscopic behavior of the *Vertex* system and its basic (and essential) functional points. These functional points are closely related to the core system requirements mentioned at the beginning of Section 2.1.

The structure of the global architecture of *Vertex* will guide the researchers in their implementation of the functional points that will be integrated in the different releases of the system. Of course, these functional points will be implemented through active research on the scientific problems specific to *Vertex*. It is important to stress the fact that the design of a system such as *Vertex* is a research problem in itself and it is of utmost importance that the design of this architecture should not be "exploded" among team members. We thus propose that the global architecture of *Vertex* be designed by a limited number of individuals. This does not mean that the suggestions of team members will not be taken into account. Quite the opposite! After all, team work is all Vertex is all about. It simply means that the software architecture should be maintained by a limited number of persons who take into account the input of each team member but who also have to make the trade-off between possibly conflicting suggestions.

2.1.1.2 Research issues

As mentioned in Section 2.1.1.1 above, the functional points of *Vertex* will be the result of the fundamental research pursued in the specific

areas explored by team members. Research on this specific topics fundamentally adopts a bottom-up approach since paradigms, algorithms, etc..., must be tested separately in order to evaluate their performance and robustness as well as their capacity to meet *Vertex*'s requirements. Research results should be integrated in a *Vertex* release only when these important characteristics of reliability, robustness, and functionality are met[1] up to a level that is satisfactory for planning the next release. Testing will be performed by the system users and the test engineer[2]. The testing will be performed by reviewing the functional points and by verifying that the system's behavior meet the desired behavior for each functional point. By keeping in mind the global architecture of *Vertex*, bottom-up solutions will be easier to integrate since they will have to adopt this architecture from the start[3].

2.1.2 Prototyping

The validity of an approach or idea is better evaluated through rapid prototyping of solutions. These prototypes should be implemented in the form and with the tools that are the best suited for the problem[4]. However, such prototypes should **never** be (and ***will*** never be) integrated in *Vertex* as such. It is thus important that team members keep in mind that prototyping should be a vehicle to test ideas and **should not** be considered as the end product of *Vertex*. *Vertex* **will not** be the integration of separate prototypes.

2.1.3 Implementation

The algorithms validated through prototyping will have to be implemented according to the guidelines provided by the global architecture of *Vertex*. A set of programming standards will be established once the software platforms are chosen (see Section 2.3).

2.1.4 Integration

The integration of the various functional points will be performed by research engineers under the guidance of the team members. The

1. We could also mention computational tractability as an important characteristic.
2. Who must be different from the software architect and designer.
3. Of course, the architecture will have to evolve in the development process. However, if this architecture is well thought right from the start, only incremental changes should be brought to the basic structure of *Vertex*.
4. For instance, Matlab is sometimes a very good tool for prototyping parts of algorithms.

integrated releases of *Vertex* will be available to all team members for demos and as a basis for discussion for the next releases.

2.1.5 Reuse

Vertex classes should be designed with a constant concern for their potential for reuse. The effort (and money!) that will be put into *Vertex* should be used optimally since we all hope to get the 3-year extension(!) and that the work in this last three years will be easier if code can be reused.

2.1.6 Documentation

The class hierarchy, class diagrams, class library, object diagrams, module diagrams and scenarios will have to be clearly documented. We have been evaluating several CASE tools for OO-development and OO-documentation. Among these tools, Rational Rose 4.0 appears to be a very interesting OO development tool since it produces coherent documentation on all diagram that are built during the analysis / design phase. It also allows round-trip software engineering, a very interesting feature[1].

We will also build a class library with a class browser so team members will be able to check whether they could reuse classes designed by other members.

The most crucial information should be available through a Web-based repository, so that network-wide development can take place throughout the *Vertex* community. We are currently evaluating ways to make this, as well as other *Vertex* material, available in this way.

2.1.7 Object-Oriented issues and Human-Machine Interfaces

In a long-term project as large and geographically distributed as is Vertex, Software Engineering methodologies are critical for success. These techniques support the design and evolution of each component as well as the integration into a complete system. We would like to begin by focussing on how Object-Oriented design methods can be extended to incorporate the user-centred constraints imposed by the task and the system user. We wish to embrace this approach, in particular, for the

1. Round-trip engineering implies that the documentation and code can be produced from Rose diagrams and that the diagrams can be updated from code that has been developed.

Human-Machine Interface layer of each of these components. As Vertex project participants develop systems and software, the UWO role will be to coordinate the user interface components. We wish to address a critical tension between the user-centred design constraints, and the system-centric constraints imposed by the technology of the interface and the target systems. Furthermore, we need to have a systematic approach in order to channel the results of each interdependent project activity into an integrated system. To do otherwise would be to risk the possibility that, at the end of the day, our efforts come across like a set of disconnected demos. The design of the HMI components of the Vertex project need to be managed within a Software Engineering process which complements the design of all other system components.

This idea is consonant with formal methods which are emerging in the literature on UI design (cf. Harmelen et. al., 1997) which encompass the entire process that is required in order to design a user interface. This is an activity which involves the designers, users, evaluators, supervisors, and programmers. Each of these individuals have distinct roles, and the information that each of them deals with will certainly have different formats and procedures; yet each plays an important part in shaping a description which leads to the design and implementation of the system and its user interface.

Consider some of the basic entities that are central to the design process for a simple Human-Machine Interface. One way is to begin with the user (in the spirit of user-centred design, which will be stressed throughout the process). The role of the user is to accomplish tasks within some problem domain. To do so, they will make use of abstract tools -- things which are relevant to the task -- and because this methodology is suitable for a wide variety of applications, these entities will be labelled quite generically as ’Referents’.

In the user interface, the referents will be represented by objects that must be implemented by the UI and supporting system. The user-centred process for UI design begins with the UI designer meeting with typical users of the system through a process of formal enquiry. The goal is to produce formal descriptions of the abstract entities in the problem domain. The formal descriptions are a high level (abstract class) type of information about the overall system. The Object Model is a subtype of information, whose contents inherit their structure from the abstract descriptions, but which carry more concrete details leading towards an implementation in a computational system. Through aggregation, the Object Model is composed of classical ‘Objects’ (the abstract entities that

get implemented and supported by the UI system). It is a feature of this Object-Oriented approach that these entities will be sufficiently abstract so as to include: things within the physical world, things existing in the user interface, things about the task itself, and things involving the users themselves. These are representations of the 'Referents', which allows us to close the design loop back to the user.

2.1.7.1 Requirements Engineering: Task Analysis in HMI Design

We have chosen to focus on two methods which have been proposed to elicit formal descriptions of entities and objects of the task through an active process between the user and the designer; namely, Use Cases and Usage Scenarios.

2.1.7.1.1 Use Cases

One way to characterize the overall functionality of a system is to exhaustively list the set of all interactions between each agent and the system itself. Each action or transaction can be described by a ''use case'' (cf. Jacobson et al., 1992). In the initial phase of a design process, the overall system does not exist, and so its entire functionality cannot be fully appreciated. However; it is often possible to bootstrap the implementation of a system prototype by carefully itemizing an initial set of typical use cases. As the system evolves, the goal is to expand the list of use cases to account for all functional roles which may be invoked by any particular actor or system component. As an example, consider the follwing initial use cases which describe the functionality of a very simple system prototype.

Identifying the 'requirements model' is only the initial stage of a process of iterative specification and refinement of the analysis model for the system. As the overall structure of use cases begins to take shape, some use cases can be split into more specific instances and then elabourated. A good example of this (modified from Jacobson et al, p.164) would be the situation in which any robot motion is blocked by an obstacle. This becomes a use case which extends the Explore Environment description. When the robot path is blocked and the obstacle cannot be avoided automatically, and alarm should be sent to the Operator. The Operator can then control the robot manually to move it to a free point in the configuration space (by having switched to a real-time control mode). Control can then be passed back to the automated environment exploration mode. In the Use Case model, an extension association

(drawn with a dashed arrow) is setup, and the use case description for ‘‘Robot Blocked by Obstacle’’ may be specified.

For each use case, the descriptions should be expanded until the basic course of action emerges with a clear flow (alternative courses of action can also be specified for certain error conditions). A feature of this approach is that if the initial requirements were underspecified initially, critical details will be noticeably missing at this point, and automatically call for tightening up. This is an efficient design method, since only the critical functions will present themselves first to be fleshed out in more detail.

2.1.7.1.2 Usage Scenarios

Mary Beth Rosson (1997) suggests the adoption of a related approach to Use Cases, called ‘‘Usage Scenarios’’ as a method for integrating task analysis with object-oriented Software Engineering. At the heart of the method is the itemization of a list of scenarios, taken from a process of formal enquiry with current and future users of the system. The scenarios initially have a functional characteristic. They are meant to capture basic actions, goals, and tasks within the problem domain, without capturing any details about the look-and-feel of the user interface or the system being controlled. Example scenarios within the Vertex domain might include:

Sample Usage Scenarios for Vertex Problem Domain

GOAL	Move robot to specified viewpoint.
GOAL	Move robot-2 to location relative to robot-1.
DISCOVERY	New feature seen in view-2.
GOAL	Initialize sweep of area with robot-2 and robot-3.
EVENT	Operator requests help with recognizing feature X.
TASK	Launch sensory-motor procedure ABC at relative location P.

2.1.7.1.3 ‘Noun Analysis’ and Object Models

Abbott (1983) was one of the first to suggest that, in an object-based approach to programming, the critical entities could be identified by first writing a textual description of the problem, and then identifying the nouns and verbs in the text. The nouns would correspond to the salient objects in the system, and the verbs would correspond to the actions to be performed on those objects.

Once a collection of Use Cases or Usage Scenarios has been established and refined, the next step is to identify the entities within the problem domain. This can be done by picking the nouns from the scenarios. These entities becom the potential objects that need to be supported in the user interface. From the scenarios listed above, these would form a list that could evolve along the following initial entities:

- Robot_i
- Operator_j
- WorldFeature_K
- ImageFeature_k
- Location(x,y,z)
- Location (relative (Point W))

The Wirfs-Brock (1989) approach to Object-Oriented design is then applied in Rosson's 'Scenario Browser' technique (called 'Point-of-View' analysis) which establishes an anthropomorphic vantage point for the distinct entities that arise from the object-based analysis. To be more specific, each entity is considered in turn, and a description of 'what it would be like to be that object' is articulated. In essence, this can consist of descriptive text such as,

''I am robot(i). I am inactive until I awake to event(y). I receive my commands from operator(j). My task is to move according to the specified direction. If I depart from my prescribed threshold by more than Q metres, I will stop and signal an alert. I will relay a stream of images using the communications link''

This allows us to identify the processes and functions embodied by each of the entities. Each of these object points-of-view can make reference to previously undiscovered entities, which should then be added to the list of objects. As these are added, a network of collaborating objects can be constructed, which establishes the foundation for the object-based method. Through further analysis, different uses for the same object can emerge and/or multiple related instances can be merged and abstracted into appropriate class hierarchies. In addition to the associations setup between each of the objects, their individual roles are identified. This is a critical step which leads to the identification of the functions and methods which will need to be implemented.

2.2 Analysis and Design Tools

The different models, diagrams, as well as the documentation mentioned in earlier Sections will have to be designed and maintained with proper OO Software development tools. Rational Rose 4.0 has been chosen as the OO Software design and analysis tool. This software is currently being used at Laval[1] and has proved to be very promising. Rose allows for the design and documentation of Use-Cases, class diagrams, object diagrams, interaction diagrams, etc. It allows code generation and round-trip engineering. Furthermore, it runs on Windows 95/NT-4 and Unix platforms. Under educational discount, the NT version can be bought for 800$ and the Unix version (1 token) for ~2500$. Each team member is encouraged to buy the software early in the project and use it for all software design/analysis of *Vertex's* modules.

2.3 Simulation environment and standards

2.3.1 Software platforms

We have also been investigating several Virtual Reality authoring software modules. Among others 3D Studio Max has been tested for the construction of 3D scenes with texture mapping and found to be quite attractive. Plug-ins have been developed within LVSN and its VRES Project for intelligent user-guided stereo matching and 3D scene graph generation. Additional plug-ins are being developed for the simulation of range finders. World Toolkit and the distributed WorldtoWorld environment by Sense 8 are currently being tested for similar issues.

The overall software platform should encompass a number of qualities, including

- extensible architecture
- capable of supporting distributed processing, with effective communication semantics
- supportive of and supported by open standards
- capable of supporting soft real time and linking to hard real-time resources.

1. Rational Rose C++ Version 4.0

We have been in contact with researchers in the community who have suggested other platforms that will have to be investigated[1]. It should be stressed that these qualities do not have to be fully available in the prototyping phases. They rather have to be kept as requirements for the ultimate implementation of *Vertex and* therefore considered right from the beginning of the Project.

At some point, the team will probably have to consider to use ORB technology (and related HLA technologies) for simulations that will require intensive computing resources.

2.3.2 Hardware platforms

Even though it seems premature to discuss hardware platforms at this stage, the team will have to consider this issue early on since early releases of Vertex will certainly demonstrate the limitations of current hardware. While Unix (Sun, SGI, Linux) are highly relevant platforms, the emerging importance of WinNT - with its rapid growth in OpenGL graphics - will also have to recognized. Furthermore, since Vertex has a real-time component, real-time OS will also be involved. At LVSN we have had extensive experience with QNX and IREQ and CRIM are also using it. *Vertex* will require a careful analysis of hardware requirements, including special components ranging from HMI interface to ROV sensing and communication.

2.4 Staffing

The research work and prototype development will be performed by graduate students and team members. However, it is doubtful that an integrated system can be assembled by graduate students. Experience has shown that it is difficult top accomplish a complete product by using prototypes. For this reason, a software engineer will be responsible for integrating the research results for each release of *Vertex*. This justifies even more the requirements for the use of a sound software engineering approach by each team member.

1. For instance the Bamboo system being developed at the Naval Postgraduate School is an attractive candidate. See http://npsnet.nps.navy.mil/Bamboo/papers.html. The ACE environment for communications among distributed components is also relevant. See http://www.cs.wustl.edu/~schmidt/ACE.html.

CHAPTER 3 Management Issues

The size of the *Vertex* team and the level of complexity of the scientific problems that are tackled impose the adoption of a flexible yet efficient management structure as well as clearly defined responsibilities.

3.1 Team management structure

Vertex is not a business and should not be run as such. However, the importance of the project and the mode into which all team members must learn to operate is new and we must make sure that all members are constantly informed of the orientation of the project.

3.1.1 Steering Committee

The strategic orientations of *Vertex*, the budget, and other general *management issues* should be discussed by a steering committee composed of *Vertex* Principal Investigators, *Vertex* industrial partners and *Vertex* staff.

3.1.2 Technical Committee

The responsibilities of the technical committee will focus on scientific, engineering, and technical issues on all aspects of *Vertex*.

3.1.3 Technical Staff

Vertex Project Leaders and Principal Investigators will be responsible for supervising major *Vertex* activities, permanent technical staff will be responsible for the follow-up of daily activities. An important task that Project leaders will have to carry on as *Vertex* officially starts will be to hire this technical staff.

3.1.4 List of members

A list of *Vertex* member will be kept and updated regularly. Principal Investigators should communicate staff changes to the Project Leaders.

3.2 Communication

Important issues will have to be discussed during team meetings. However, taking into account that the team members are located all across the country and that it is almost impossible to schedule frequent meetings, issues of less importance will have to be taken care of by usual communications techniques such as e-mail (for exchanging ideas and opinions as well as objections!) and electronic transfers (for code, documentation, web pages, etc...). We should exploit video conferencing as it becomes more readily available. Phone seems to be the worst communication system ...

3.3 Reports

Vertex will have to maintain a repository for “corporate” as well as technical information. An electronic repository seems to be the most appropriate since the team is distributed in several locations. Consequently, all theses, reports, and other documents should be made available to the team in electronic format (preferably PDF files). A Vertex web site, with a public area as well as sections which are restricted to *Vextex* members will be available. Copyright issues will have to be discussed prior to the disclosure of any document.

3.4 Bibliography

As for reports, *Vertex* should keep a record of all pertinent references to papers, reports, theses that are used in the development of the functional points. This will be important for writing reports, papers, quarterly progress reports and so on. It is not yet clear how this database of

references will be implemented but it should be accessible (for deposit and retrieval of data) by all team members.

3.5 Meetings

To be completed

3.6 Technology transfer

To be completed

3.7 Intellectual property

To be completed

3.8 Actions to be undertaken by team members

After reading this draft the *Vertex* Principle Investigators are expected to take the following actions:

1. Return comments, suggestions, additions, corrections to this document, which should be understood as a work-in-progress to be enhanced after this initial release.
2. Prepare a description of the technical problems that they will addressed based on the basis of the area of contribution listed in the original *Vertex* proposal, the contents of the White Paper and the generic problem description that is outlined in FIGURE 1.
3. Provide a list of students that are expected to work on *Vertex* so we can prepare a complete list of members.
4. Return the information above to laurend@gel.ulaval.ca, poussart @gel.ulaval.ca and cedras@gel.ulaval.ca.
5. A meeting of the PI's will be arranged at the IRIS Conference in June 1998. Other meetings will take place in the meantime.

Denis Laurendeau and Denis Poussart

March 1998

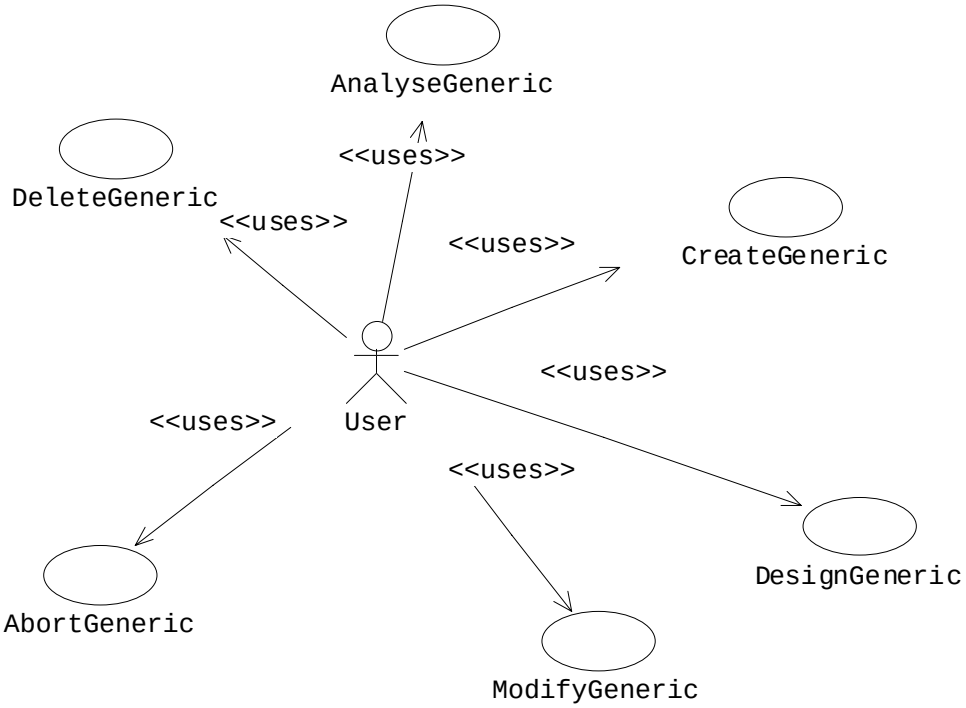
CHAPTER 4 Vertex Use Cases

The following is an attempt to define a set of Use-cases in UML format for Vertex. Major use-case diagrams of the main functionalities of Vertex are illustrated and their preliminary specifications are given.

The UML diagrams are given along with general comments. The complete description of the use cases can be found at the end of the chapter.

4.1 Generic Use Cases

The General Use case diagram shows the most important functionalities that Vertex should address. Each Use-case in this diagram is covered in more details in the following sections.

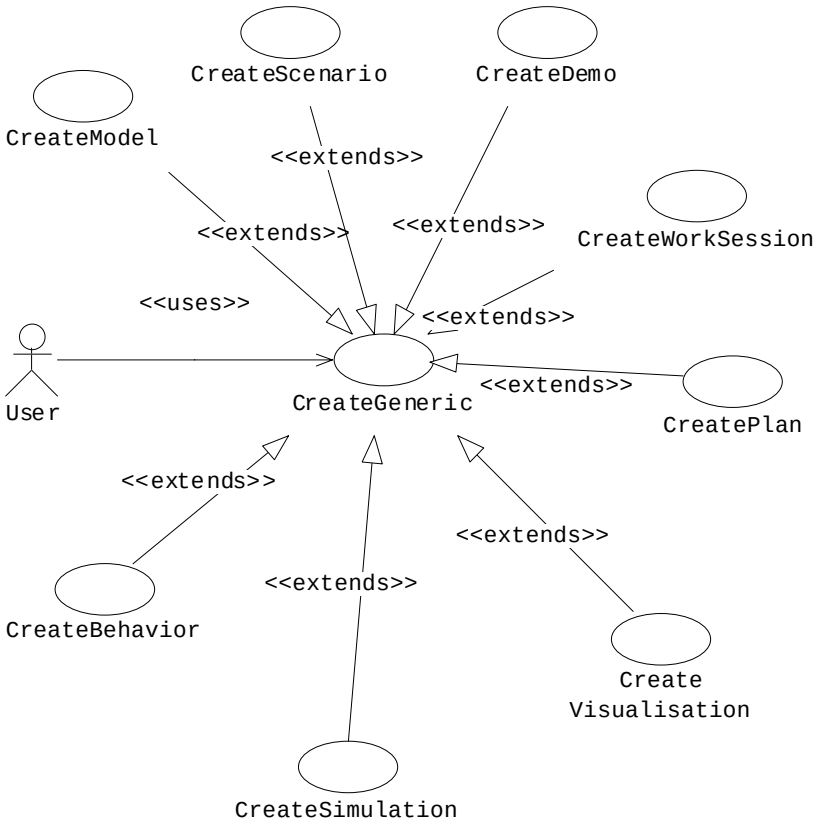


In the above diagram, all major generic interactions between the user and the Vertex system are shown:

1. CreateGeneric: use case for the creation of any instance in Vertex;
2. DesignGeneric: use case for the design of any instance in Vertex;
3. ModifyGeneric: use case for the modification of a previously created and designed instance in Vertex;
4. AbortGeneric: use case for the abortion of any interaction between the user and Vertex. Restores the previous state of the system prior to any modification.
5. DeleteGeneric: use case for deleting any instance previously cretated-designed-modified in Vertex.

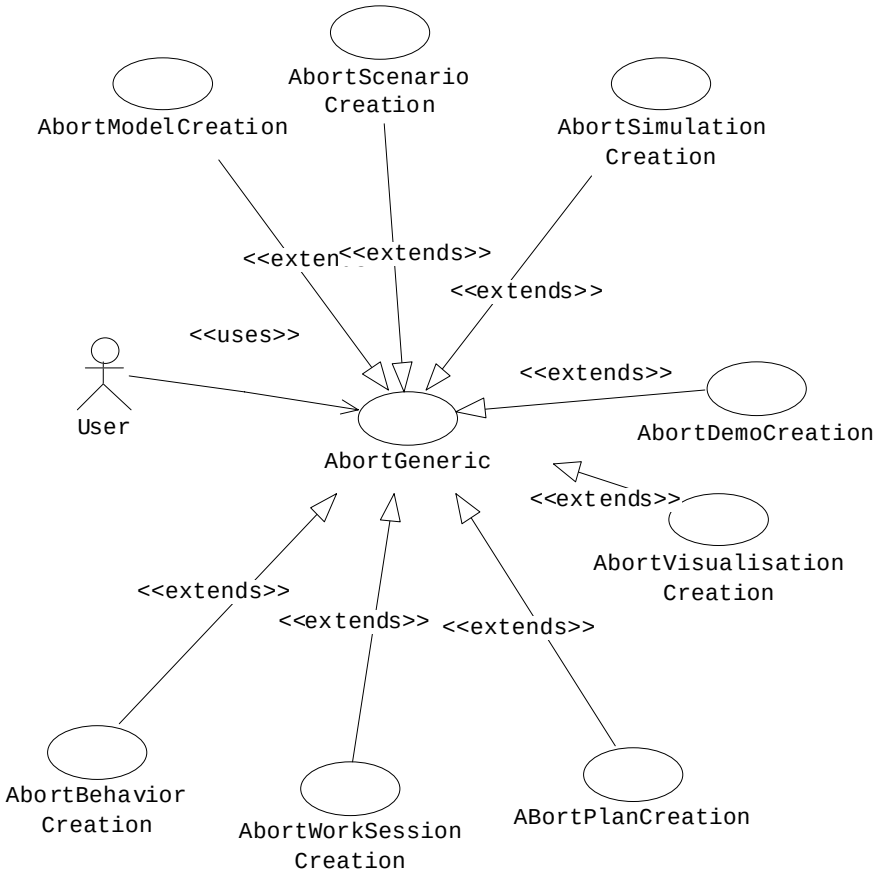
4.2 Create Use cases

These use cases are concerned with the creation of instances of Vertex elements.



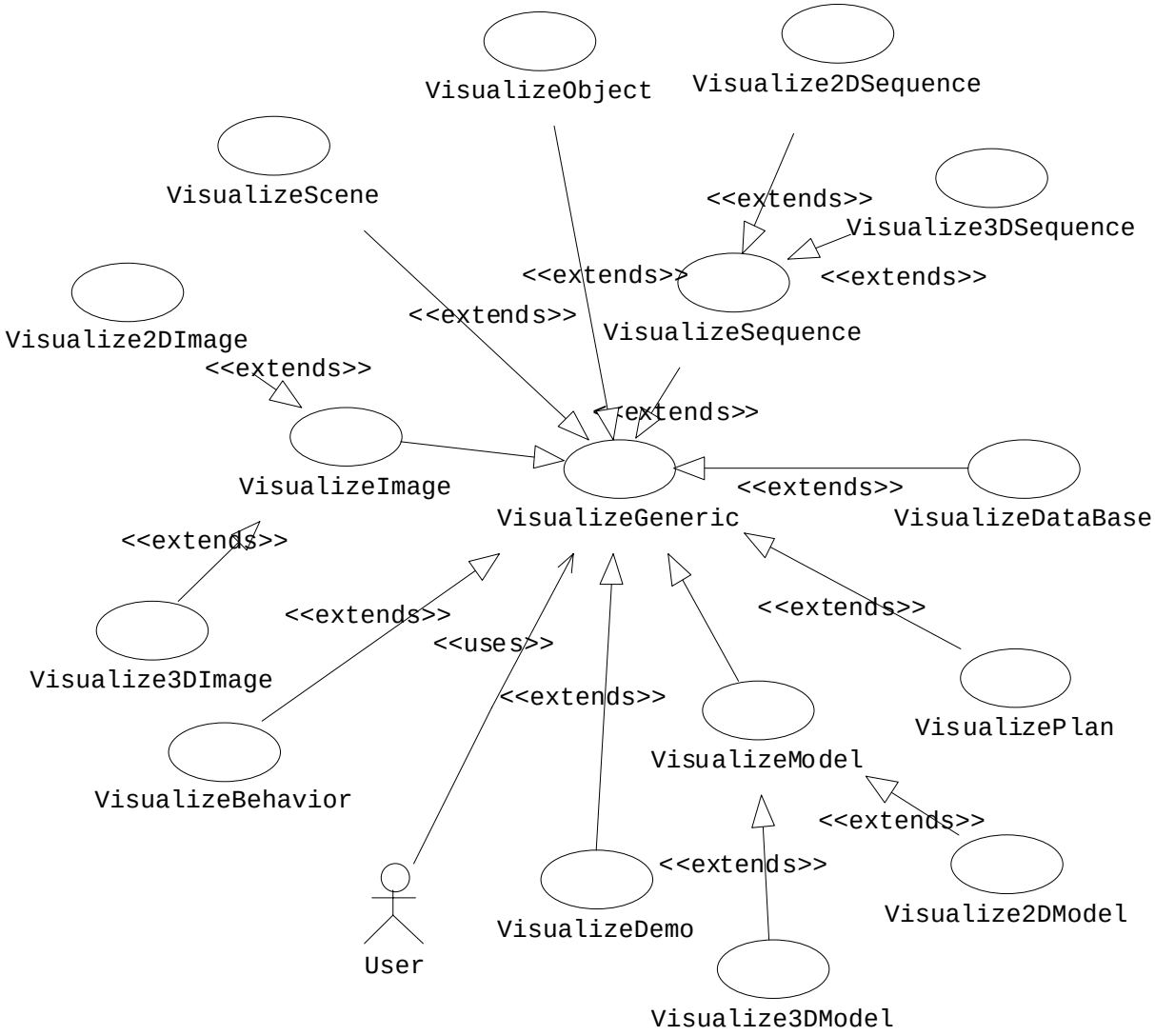
4.3 Abort Use Cases

These use cases are concerned with the abortion of interactions between the user and the Vertex system.



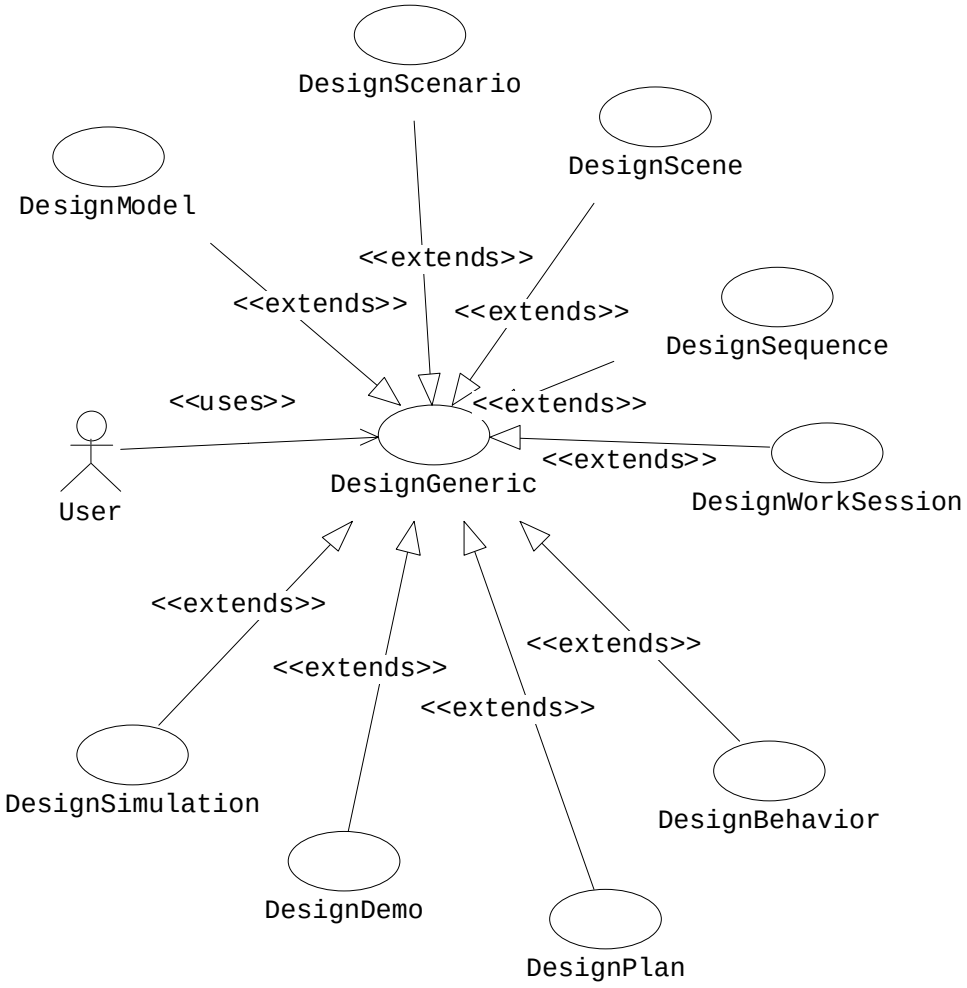
4.4 Visualization Use Cases

These use cases deal with the visualization of elements in Vertex.



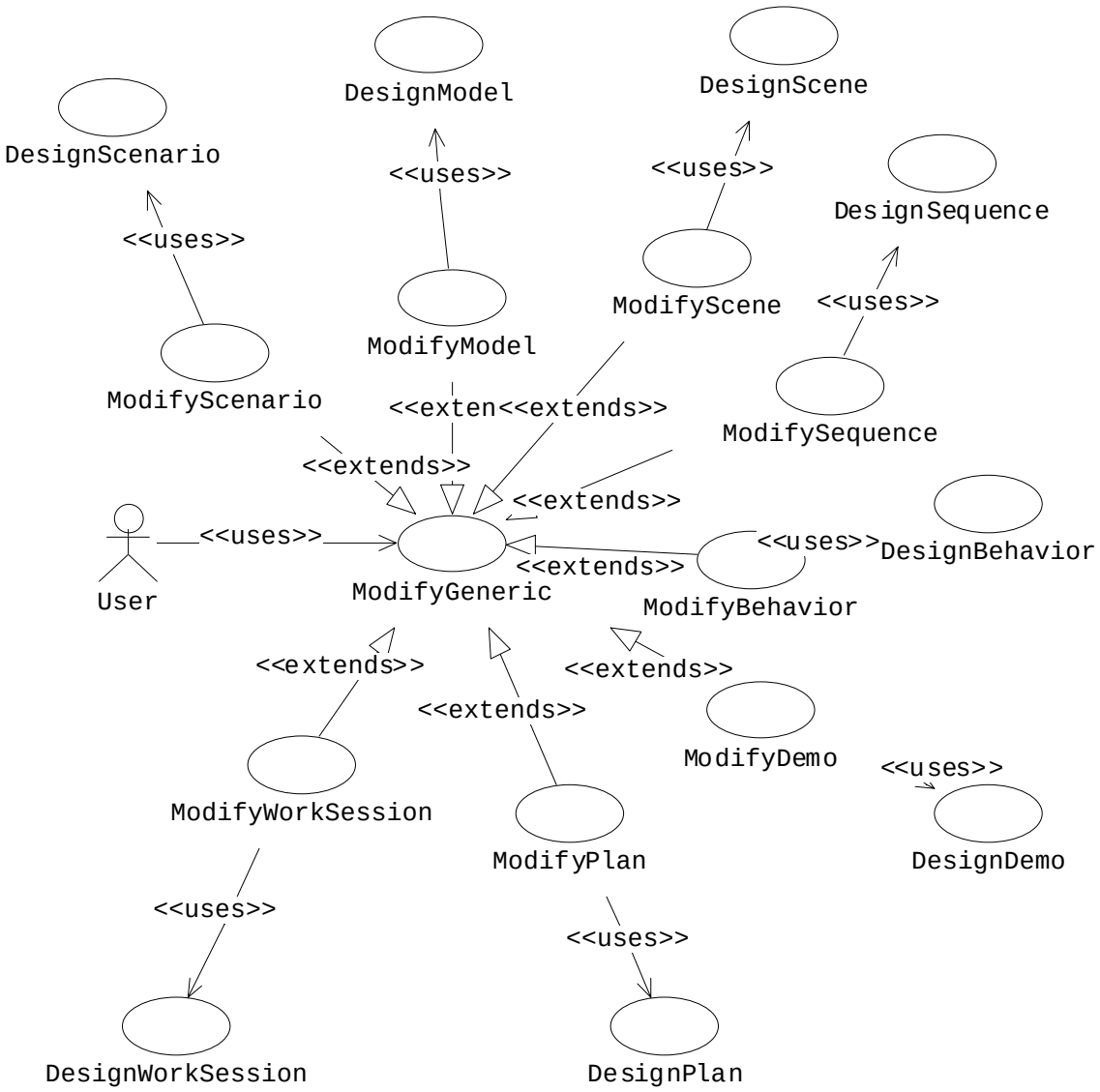
4.5 Design Use Cases

These use cases are concerned with the design of elements in Vertex



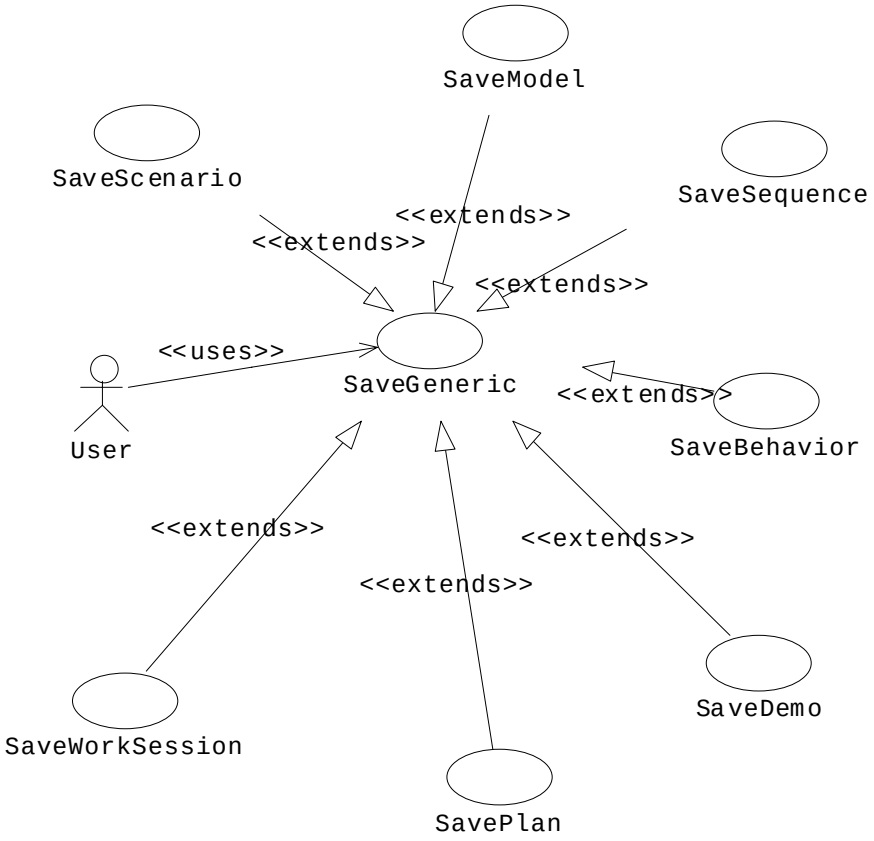
4.6 Modify Use Cases

These use cases are concerned with the modification of elements that were created-designed in Vertex



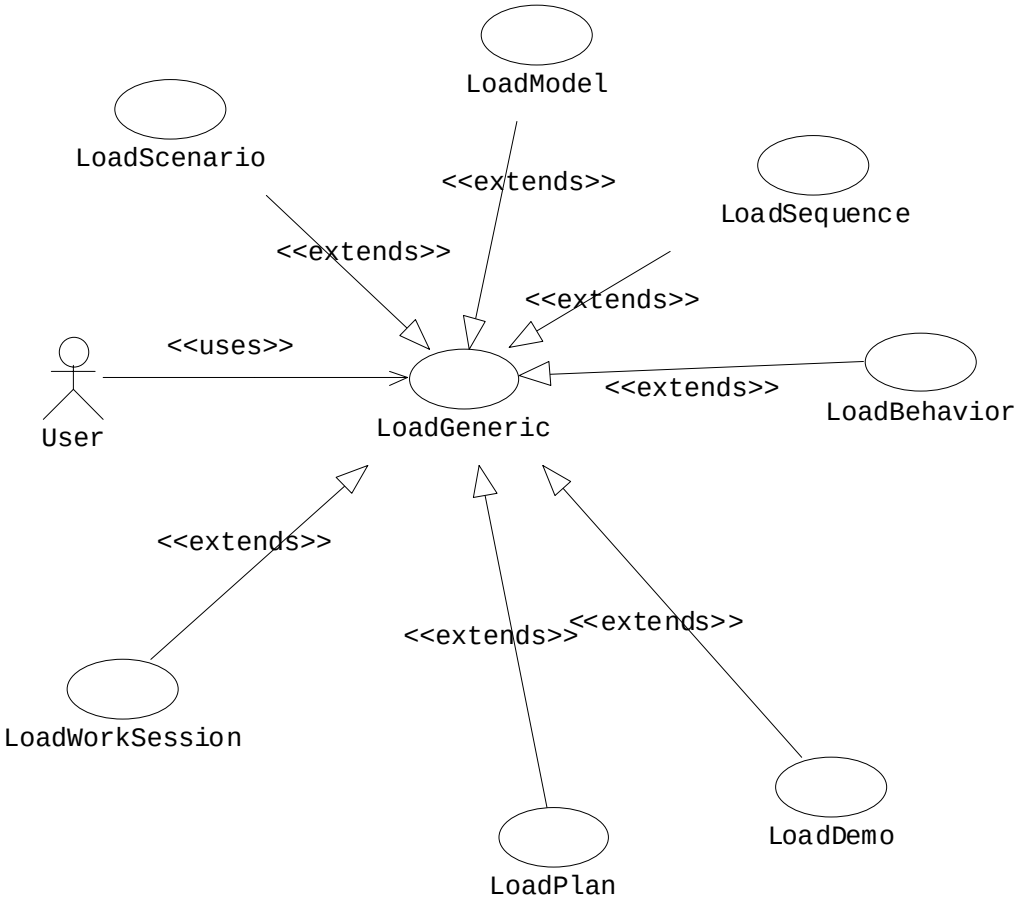
4.7 Save Use Cases

These use cases deal with the action of saving work performed during a Vertex Session.



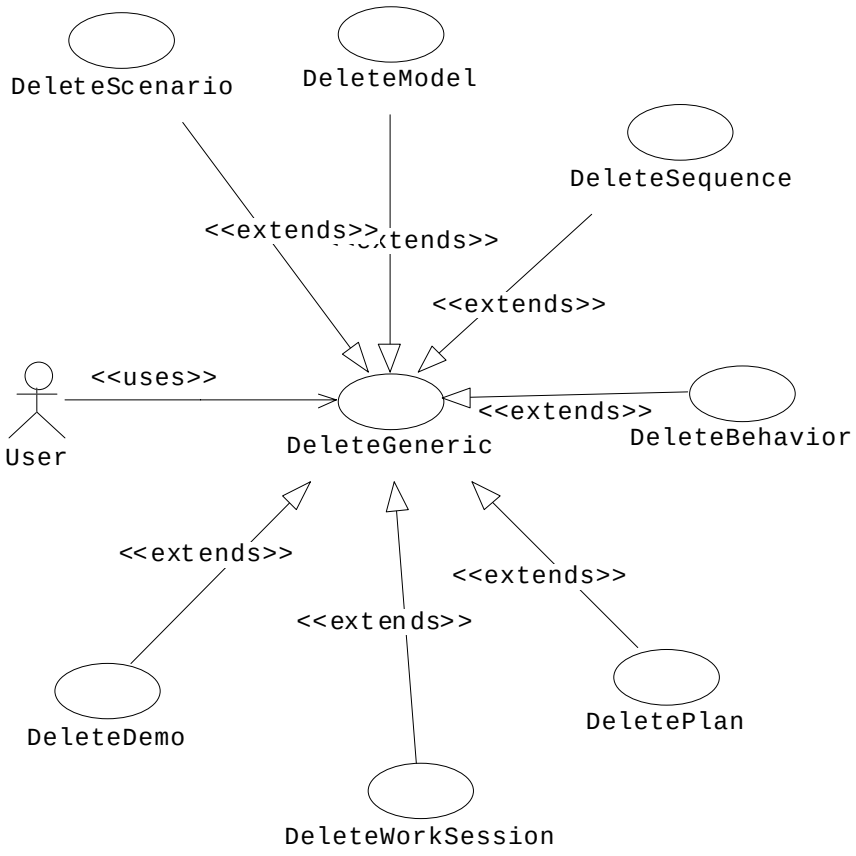
4.8 Load Use Cases

These use cases are concerned with the action of loading previously saved operations in Vertex.



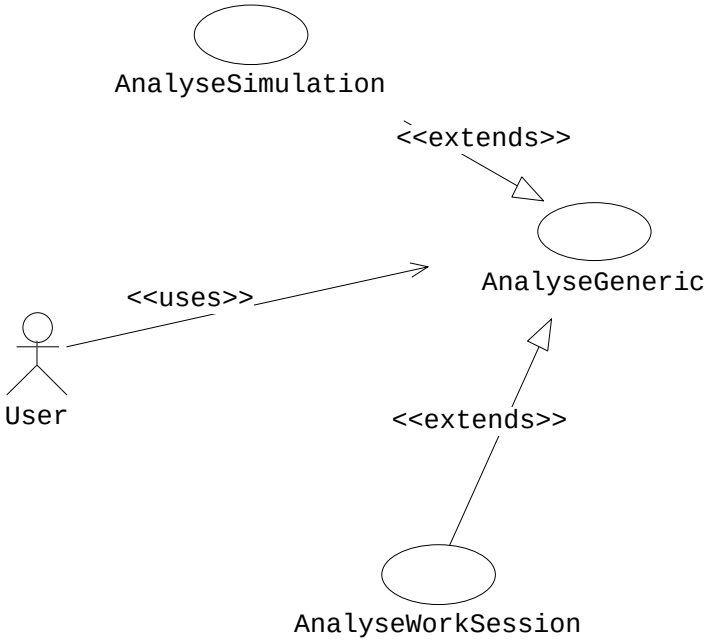
4.9 Delete Use Cases

These use cases are coincerned with the deletion of elements that were previously created-designed-modified in Vertex.



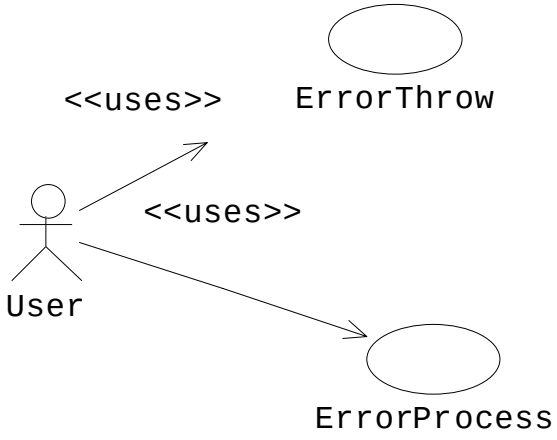
4.10 Analyse Use Cases

These use cases are concerned with the analysis of asequence of operations in Vertex.



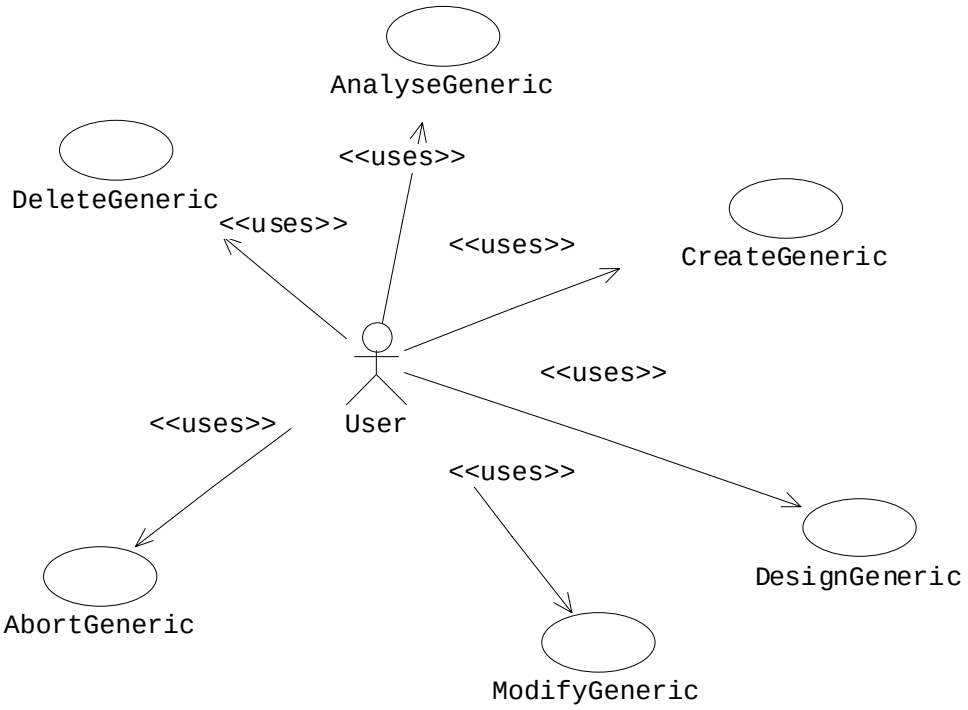
4.11 Error Use Cases

These use cases are concerned with the processing of errors / exceptions in the course of a Vertex manipulation.



Use Cases - Complete and Detailed Description

4.12 Generic Use Cases - Complete and detailed description



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
DesignGeneric

Category: Use Case View
Documentation:
Generic use case for designing VRES/Vertex elements. In design mode, the user wants to use raw data in order to build components of the world into which a simulated intervention will take place. Most of the specific use cases in the Design category are still vague and need more interaction with team members in order to get a more precise description.They will be discussed and documented in the upcoming months

Flow of events
A- Preconditions
The execution of the appropriate Create use case must have been executed prior the related Design use case.
B- Main flow of events
This use case begins when the user selects the Design mode in VRES/Vertex. The system enters the generic Design mode and prompts the user for the element he wants to design. A list of "designable" elements could be provided in a browsing window in order to assist the user in his choice. The user chooses an element. The system enters in the appropriate design mode for this element (design modes may be different for models, behaviors, etc. See corresponding use cases for the design of each element category). The user performs the required design operations on the data and exits the design mode (he can save his work with the Save use cases).
C- Subflows
None
D- Alternative flows
The user should be allowed to exit the design mode if he changes his mind and return to the environment.

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
CreateGeneric

Category: Use Case View
Documentation:
Generic use case for the creation of an element in VRES/Vertex. It is a very general and abstract use case that should capture the general operations that are relevant to the creation of an element in VRES/Vertex.

Flow of events
A- Preconditions
The VRES/Vertex environment must be running and waiting for user input.
B. Main flow of events
The use case begins when the user wants to create a new element in VRES/Vertex. An element is a generic component of the system. See specific use cases for the enumeration of the elements.The user should input the "create element" command in the HMI in order to initiate the CreateGeneric use case.The VRES/Vertex environment then enters the mode for the creation of elements. The commands he can enter while in this mode depend on the element that is created. See specific use cases for a description of these commands.
C. Subflows
None
D. Alternative flows
a-The user should be able to exit the "create mode" without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
AbortGeneric

Category: Use Case View
Documentation:
Generic use case for aborting a task in VRES/Vertex. It is a very general and abstract use case that should capture the general operations that are relevant to aborting a task in VRES/Vertex.

Flow of events
A- Preconditions
The VRES/Vertex environment must be running and waiting for user input.
B. Main flow of events
The use case begins when the user wants to abort a task he has begun without saving his work. A task is a generic task (see all other use cases) that is executed by the system. See specific use cases for the enumeration of the elements.The user should input the "abort" command in the HMI in order to initiate the AbortGeneric use case.The VRES/Vertex environment then enters the mode for the interruption of tasks without saving. The events that occur while aborting a task depend on the task at hand and is a subject that is discussed in the specific Abort use cases.
C. Subflows
None
D. Alternative flows
None

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
AnalyseGeneric

Category: Use Case View
Documentation:
Use case for the analysis of actions/results/etc in VRES/VERTEX.
To be defined...

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
DeleteGeneric

Category: Use Case View
Documentation:
Generic use case for deleting previously created/modified VRES/VERTEX elements.

It is a very general and abstract use case that should capture the general operations that are relevant to the deletion of an element into a VRES/Vertex database. The element is deleted from the current environment. A special use case described the procedure for deleting an element from a database.

Flow of events
A- Preconditions
An element should have been previously loaded in the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, the user selects (through the HMI) the element he wants to delete. A browsing window also displays a list of the elements that can be deleted at this moment. The user selects one element or type the name of the element he wants to delete. The element is deleted by the system and informs the user that the element has been deleted correctly.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the element he wished to delete is not available or that the name given in a text field is incorrect (assuming the user can choose objects with the mouse or with menus and textfields). It would be interesting to allow the user to select several elements simultaneously and to have them deleted in sequence.
External Documents:
Abstract: No
State machine: No
Associations:
<no rolename> : User in association <unnamed> (uses)

Use Case name:
ModifyGeneric

Category: Use Case View
Documentation:
Generic use case for modification of actions/operations in VRES/VERTEX.

This use case is very general and should be used for modifying existing elements of the environment. See specific use cases for details on each different type of modification.

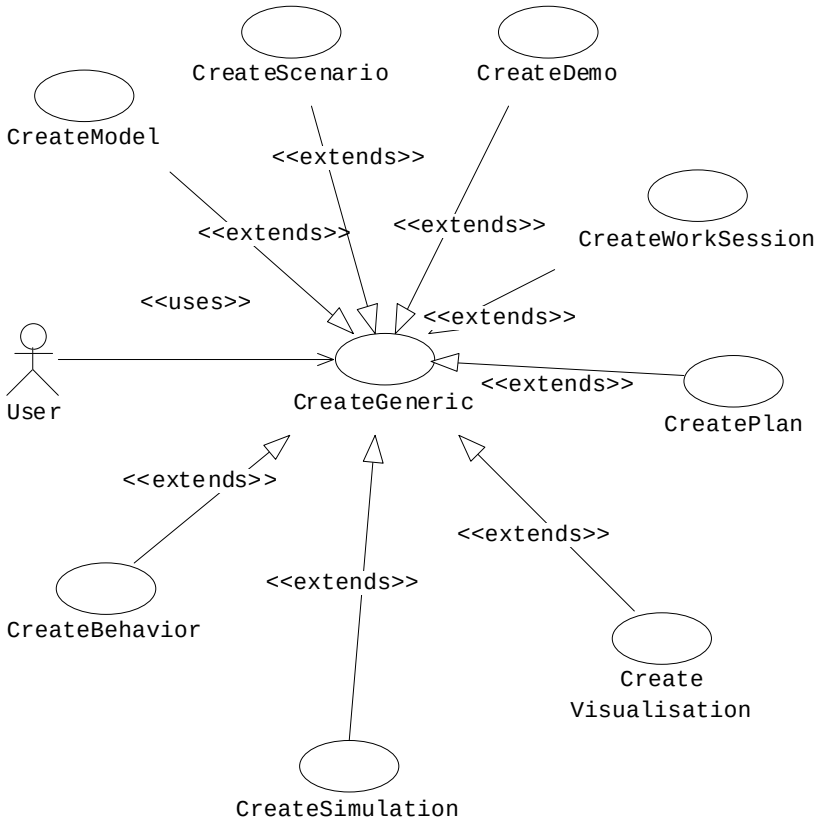
Flow of events

A- Preconditions
In order to be modified, an element must have been created and designed. It can be loaded in the environment or may be stored in the appropriate database. If it is stored in the database, the element must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Generic mode in the HMI. The element is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesigGeneric use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

4.13 Create Use cases - Complete and Detailed Description



User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor

```
External Documents:
Export Control:  Public
Cardinality:          n
Hierarchy:
   Superclasses: none
Associations:

         <no rolename> : VisualizeGeneric in association <unnamed> (uses)
         <no rolename> : SaveGeneric in association <unnamed> (uses)
         <no rolename> : LoadGeneric in association <unnamed> (uses)
         <no rolename> : ErrorThrow in association <unnamed> (uses)
         <no rolename> : ErrorProcess in association <unnamed> (uses)
         <no rolename> : DesignGeneric in association <unnamed> (uses)
         <no rolename> : AbortGeneric in association <unnamed> (uses)
         <no rolename> : AnalyseGeneric in association <unnamed> (uses)
         <no rolename> : DeleteGeneric in association <unnamed> (uses)
         <no rolename> : ModifyGeneric in association <unnamed> (uses)
         <no rolename> : CreateGeneric in association <unnamed> (uses)

State machine:   No
Concurrency:          Sequential
Persistence:          Transient
```

Use Case name:
CreateModel

Category: Use Case View
Documentation:
Use case for creation of a model in VRES/Vertex.
A model is taken in a very general sense and is inherited by mode specific use cases for the creation of different models associated with different components of VRES/Vertex. For instance a model can be a geometric model (e.g. triangulation) that is built from raw 3D data.

Flow of events
A-Preconditions
Raw data should be available for creating a model. Since the use case deals with the creation of a model, the model should not exist before its creation. See the "Modify Model" , " DeleteModel", "LoadModel", Save Model, or DesignModel" use cases for the manipulation of already existing models.
B-Main flow of events
The user selects the "CreateModel" command in the HMI and the enters the mode for creating a model.
The user is prompted for the type of model (geometric, photometric, ...), for the name of the model and the database into which the model should be stored.
C-Subflows
None
D- Alternative flows
a-The user should be able to exit the "CreateModel" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateScenario

Category: Use Case View
Documentation:
Use case for the creation of a scenario for a simulation in VRES/Vertex. A scenario is used for describing the evolution of a simulation in time.

Flow of events
A- Preconditions
a-Models should be available for describing the simulation scene.
b-Behaviors should be available for allowing the user to chose from and associate with each model involved in the simulation (a "no behavior" behavior could be attached to passive objects in the simulation).
B- Main flow of events
The user selects the "CreateScenario" command in the HMI and the enters the mode for creating a scenario
The user is asked for the type of scenario (this should be refined) , for the name of the scenario and the database into which scenario should be stored.
C- Subflows
None
D- Alternative flows

a-The user should be able to exit the "Create Scenario" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
C:\laurend\Atop\Recherche\Subventions\Iris_iii\ModelesVRESVertex\use-Cases.txt
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateDemo

Category: Use Case View
Documentation:
Use case for creating a demo in VRES/Vertex. A demo is a sample simulation run that gathers elements of VRES/Vertex and put them at work.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateVisualisation

Category: Use Case View
Documentation:
This use case is concerned with the visualization of an environment/simulation in VRES/Vertex.

Flow of events
A-Preconditions
Since the use case deals with the creation of a visualization the visualization should not exist before its creation. See the "ModifyVisualization, " DeleteVisualization", "LoadVisualization", Save Visualizationl, or DesignVisualization" use cases for the manipulation of already existing visualizations
B-Main flow of events
The user selects the "CreateVisualization" command in the HMI and the enters the mode for creating a visualization.
The user is prompted for the name of the visualization he wants to create.
C-Subflows
None
D- Alternative flows
a-The user should be able to exit the "Create Visualization" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreatePlan

Category: Use Case View
Documentation:
Use case for creating a plan (planning phase) in VRES/Vertex. A plan is concerned with the optimization of steps in an intervention simulated/executed in VRES/Vertex.

Flow of events
A- Preconditions
Since the use case deals with the creation of a plan the plan should not exist before its creation. See the "Modify Plan, " DeletePlan", "LoadPlan", SavePlan, or DesignPlan" use cases for the manipulation of already existing plans.
B-Main flow of events
The user selects the "CreatePlan" command in the HMI and the enters the mode for creating a plan.
The user is prompted for entering the name of the plan and the database into which the plan should be stored.
C-Subflows
None
D- Alternative flows
a-The user should be able to exit the "CreatePlan" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.
External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateWorkSession

Category: Use Case View
Documentation:
Use case for initiating a work session in Vertex. This use case is concerned with the initialization of the VRES/Vertex environment prior to the execution of any other use case.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateSimulation

Category: Use Case View
Documentation:
This use case is interested in the creation of a simulation in VRES/Vertex. A simulation encompasses several other use cases that are detailed in each specific use case.

Flow of events
A- Preconditions
Since the use case deals with the creation of a simulation the simulation should not exist before its creation. See the "Modify Simulation" Delete Simulation", "LoadSimualtion", SaveSimulation, or DesignSimulation" use cases for the manipulation of simulations.
B-Main flow of events
The user selects the "CreateSimulation" command in the HMI and the enters the mode for creating a simulation
The user is prompted for entering the name of the simulation. It is still to be decided whether a simulation will be stored in a database or not.
C-Subflows
None
D- Alternative flows
a-The user should be able to exit the "Create Simulation" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No
Generalization:
CreateGeneric (extends)

Use Case name:
CreateBehavior

Category: Use Case View
Documentation:
Use case for the creation of a behavior for a model in VRES/Vertex. This use case is more specific to Vertex.

Flow of events
A-Preconditions
Since the use case deals with the creation of a behavior, the behavior should not exist before its creation. See the "ModifyBehavior, "DeleteBehavior", "LoadBehavior", SaveBehavior, or DesignBehavior" use cases for the manipulation of already existing behaviors.
B-Main flow of events
The user selects the "CreateBehavior" command in the HMI and the enters the mode for creating a behavior. The user is prompted for the type of model (physical, thermal, ...), for the name of the behavior and the database into which the behavior should be stored.
C-Subflows
None
D- Alternative flows
a-The user should be able to exit the "Create Behavior" use case without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No
Generalization:

CreateGeneric (extends)

Use Case name:
CreateGeneric

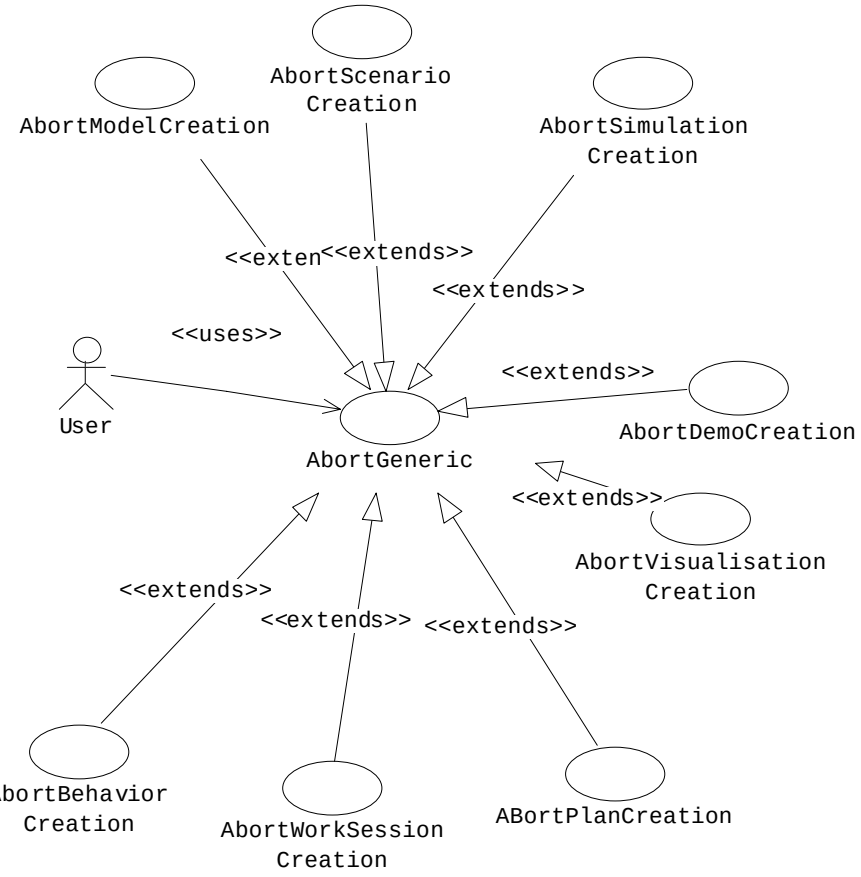
Category: Use Case View
Documentation:
Generic use case for the creation of an element in VRES/Vertex. It is a very general and abstract use case that should capture the general operations that are relevant to the creation of an element in VRES/Vertex.

Flow of events
A- Preconditions
The VRES/Vertex environment must be running and waiting for user input.
B. Main flow of events
The use case begins when the user wants to create a new element in VRES/Vertex. An element is a generic component of the system. See specific use cases for the enumeration of the elements.The user should input the "create element" command in the HMI in order to initiate the CreateGeneric use case.The VRES/Vertex environment then enters the mode for the creation of elements. The commands he can enter while in this mode depend on the element that is created. See specific use cases for a description of these commands.
C. Subflows
None
D. Alternative flows
a-The user should be able to exit the "create mode" without having to create anything if he wishes.
b-The user should be able to use the undo command in the history of the manipulations for restoring the previous context he was in.

External Documents:
Abstract: No
State machine: No

4.14 Abort Use Cases - Complete and Detailed Description

These use cases are concerned with the abortion of interactions between the user and the Vertex system



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
AbortModelCreation

Category: Use Case View
Documentation:
This use case is concerned with the task of cancelling the creation of a model in the CreateModel use case. A model is taken in a very general sense and is inherited by mode specific use cases for the creation of different models associated with different components of VRES/Vertex. For instance a model can be a geometric model (e.g. triangulation) that is built from raw 3D data.

Flow of events
A- Preconditions
The use case for creating a model must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a model. This aborts the command that initiated the CreateModel use case and return the user to the HMI. The model will not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the model and should return to where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortScenarioCreation

Category: Use Case View
Documentation:
This use case is for cancelling the operations of the creation of a scenario in the CreateScenario use case.

Flow of events
A- Preconditions
The use case for creating a scenario must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a scenario. This aborts the commands that initiated the CreateScenario use case and return the user to the HMI. The scenario is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the scenario and should return to where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortSimulationCreation

Category: Use Case View
Documentation:
This use case is for cancelling the CreateSimulation use case.

Flow of events
A- Preconditions
The use case for creating a simulation must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a simulation. This aborts the commands that initiated the CreateSimulation use case and return the user to the HMI. The simulation is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the simulation and should return where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortDemoCreation

Category: Use Case View
Documentation:
This use case is for cancelling the CreateDemo use case.

Flow of events
A- Preconditions
The use case for creating a demo must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a demo. This aborts the commands that initiated the CreateDemo use case and return the user to the HMI. The demo is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the demo and should return where he was prior to choosing the Abort command of the HMI.

External Documents:

Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortVisualisationCreation

Category: Use Case View
Documentation:
This use case is for cancelling the Create Visualisation use case.

Flow of events
A- Preconditions
The use case for creating a visualization must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a visualization. This aborts the commands that initiated the CreateVisualization use case and return the user to the HMI. The visualization is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the visualization and should return where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
ABortPlanCreation

Category: Use Case View
Documentation:
This use case is for cancelling the CreatePlan use case.

Flow of events
A- Preconditions
The use case for creating a plan must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a plan. This aborts the commands that initiated the CreatePlan use case and return the user to the HMI. The plan is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the plan and should return where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortWorkSessionCreation

Category: Use Case View
Documentation:
This use case is for cancelling the CreateWorkSession use case.

Flow of events
A- Preconditions
The use case for creating a session must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a session This aborts the commands that initiated the CreateWorkSession use case and return the user to the HMI. The Worksession is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the worksession and should return where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortBehaviorCreation

Category: Use Case View
Documentation:
This use case is for cancelling the CreateBehavior use case.

Flow of events
A- Preconditions
The use case for creating a behavior must have been initiated.
B- Main flow of events
The user chooses the Abort command while creating a behavior. This aborts the commands that initiated the CreateBehavior use case and return the user to the HMI. The behavior is not be created.
C- Subflows
None
D- Alternative flow
The user may decide not to abort the creation of the behavior and should return where he was prior to choosing the Abort command of the HMI.

External Documents:
Abstract: No
State machine: No
Generalization:
AbortGeneric (extends)

Use Case name:
AbortGeneric

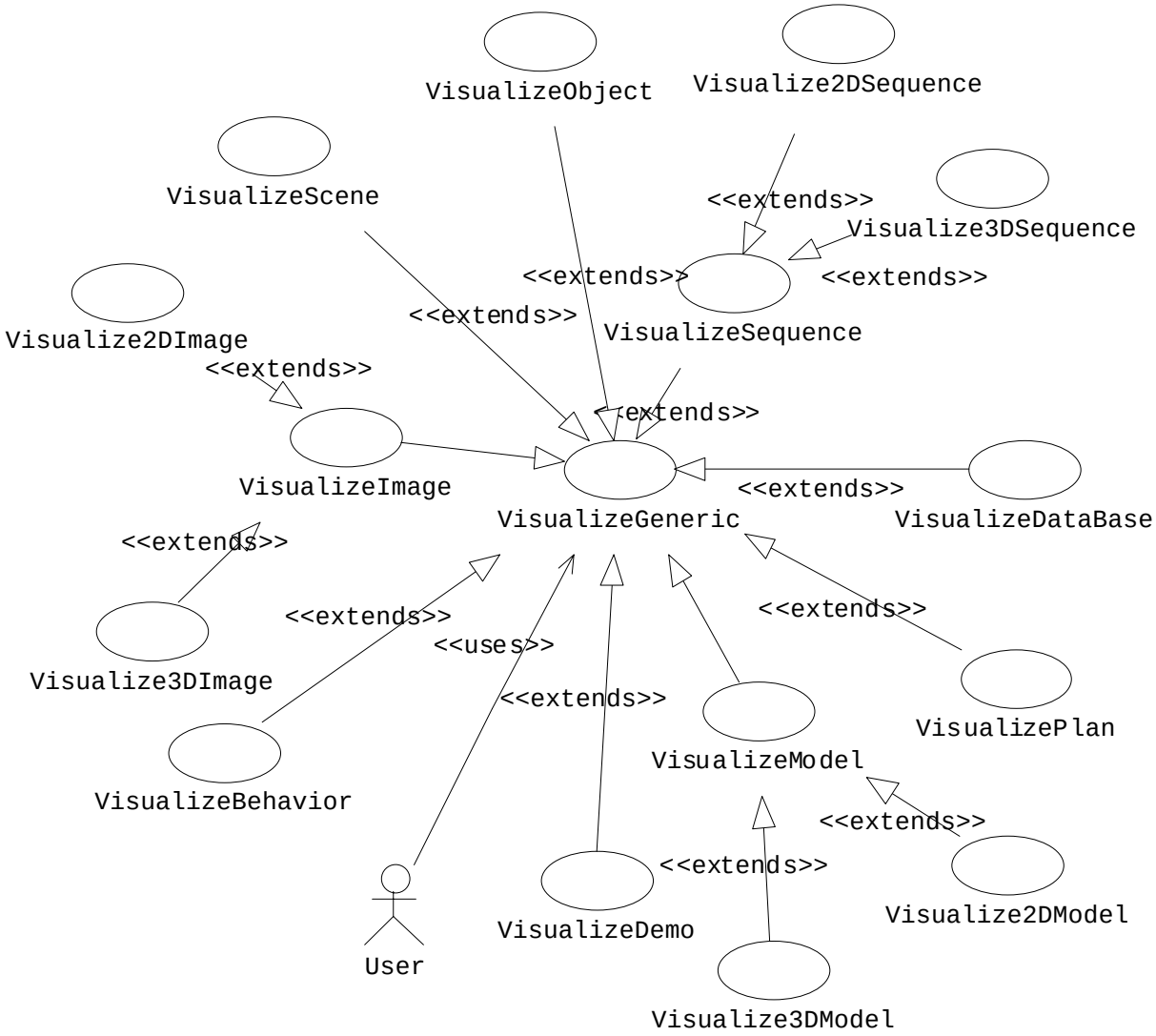
Category: Use Case View
Documentation:
Generic use case for aborting a task in VRES/Vertex. It is a very general and abstract use case that should capture the general operations that are relevant to aborting a task in VRES/Vertex.

Flow of events
A- Preconditions
The VRES/Vertex environment must be running and waiting for user input.
B. Main flow of events
The use case begins when the user wants to abort a task he has begun without saving his work. A task is a generic task (see all other use cases) that is executed by the system. See specific use cases for the enumeration of the elements.The user should input the "abort" command in the HMI in order to initiate the AbortGeneric use case.The VRES/Vertex environment then enters the mode for the interruption of tasks without saving. The events that occur while aborting a task depend on the task at hand and is a subject that is discussed in the specific Abort use cases.
C. Subflows
None
D. Alternative flows
None

External Documents:

4.15 Visualization Use Cases - Complete and Detailed Description

These use cases deal with the visualization of elements in Vertex.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
Visualize2DSequence

Category: Use Case View
Documentation:
Use case for the visualisation of a sequence of 2DImages

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeSequence (extends)

Use Case name:
Visualize3DSequence

Category: Use Case View
Documentation:
Use case for the visualisation of a 3DImage

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeSequence (extends)

Use Case name:
Visualize2DModel

Category: Use Case View
Documentation:
Use case for the visualization of a 2D model

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeModel (extends)

Use Case name:
Visualize3DModel

Category: Use Case View
Documentation:
Use case for the visualization of a 3D Model.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeModel (extends)

Use Case name:
Visualize2DImage

Category: Use Case View
Documentation:
Use case for visualizing a 2D color image.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeImage (extends)

Use Case name:
   Visualize3DImage

Category: Use Case View
Documentation:
   Use case for the visualization of a 3D image

External Documents:
Abstract: No
State machine: No
Generalization:
   VisualizeImage (extends)

Use Case name:
VisualizeScene

Category: Use Case View
Documentation:
Use case for scene visualisation.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric (extends)

Use Case name:
VisualizeObject

Category: Use Case View
Documentation:
Use case for visualising an object.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric (extends)

Use Case name:
VisualizeSequence

Category: Use Case View
Documentation:
Generic use case for visualising a sequence of images.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric (extends)

Use Case name:
   VisualizeDataBase

Category: Use Case View
Documentation:
   Use case for database visualization.

External Documents:
Abstract: No
State machine: No
Generalization:
   VisualizeGeneric (extends)

Use Case name:
VisualizePlan

Category: Use Case View
Documentation:
Use case for the visualization of a plan.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric (extends)

Use Case name:
VisualizeModel

Category: Use Case View
Documentation:
Generic use case for model visualization

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric

Use Case name:
VisualizeImage
Category: Use Case View
Documentation:
Generic use case for image visualization

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric

Use Case name:
   VisualizeDemo

Category: Use Case View
Documentation:
   Use case for the visualization of a demo

External Documents:
Abstract: No
State machine: No
Generalization:
   VisualizeGeneric (extends)

Use Case name:
VisualizeBehavior

Category: Use Case View
Documentation:
This use case allows to visualize a behavior created by the CreateBehavior use case.

External Documents:
Abstract: No
State machine: No
Generalization:
VisualizeGeneric (extends)

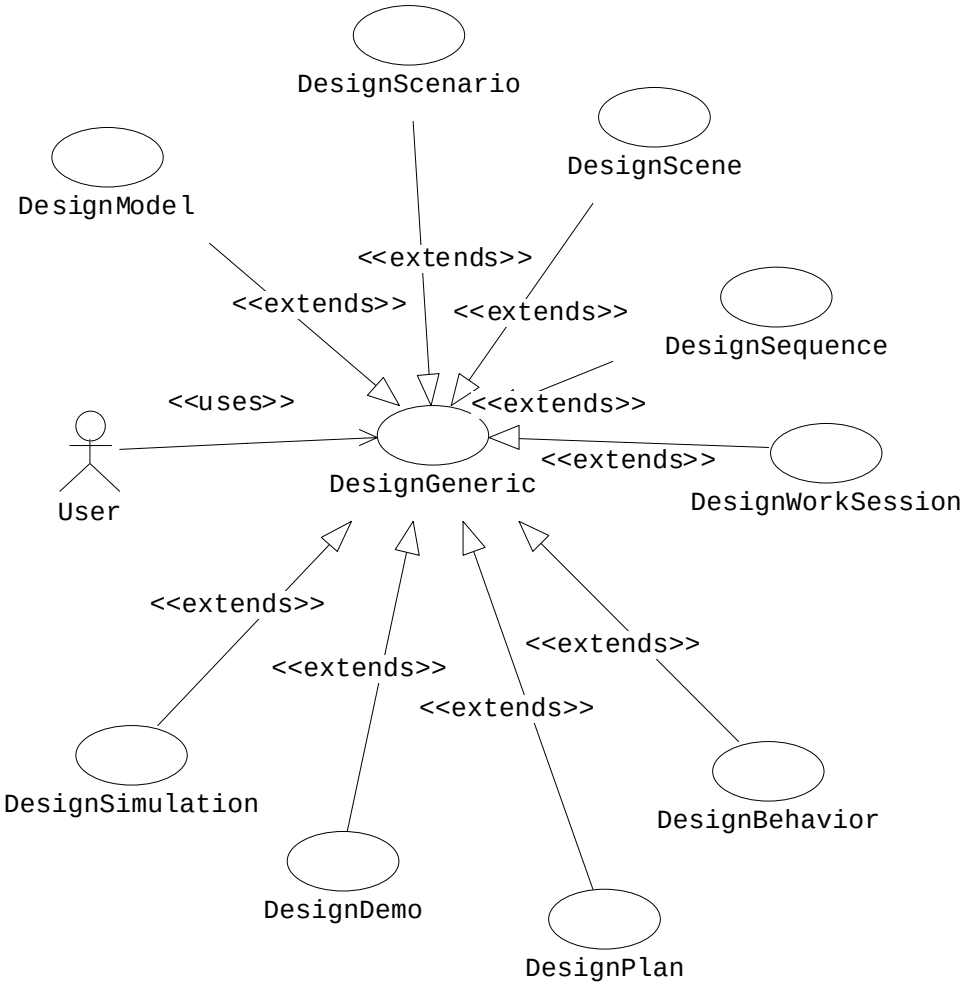
Use Case name:
VisualizeGeneric

Category: Use Case View
Documentation:
Generic use case for visualizing elements in VRES/Vertex.

External Documents:
Abstract: No
State machine: No
Associations:

4.16 Design Use Cases - Complete and Detailed Description

These use cases are concerned with the design of elements in Vertex



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
DesignScenario

Category: Use Case View
Documentation:
This use case is for designing scenarios in VRES/Vertex.

To be discussed with team.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)
Associations:

<no rolename> : ModifyScenario in association <unnamed> (uses)

Use Case name:
DesignModel

Category: Use Case View
Documentation:
This use case is for designing models in VRES/Vertex.

In design mode, the user wants to use raw data in order to build models of the world into which a simulation intervention will take place.

Feedback from Dion, Cote, Bernier, Houde, Poussart, Bergevin, Laurendeau, Ferrie, Simoneau, Borgeat, Vromet and Frey will be useful in this use case (as well as other team members

Flow of events
A- Preconditions
The execution of the appropriate Create use case must have been executed prior the related Design use case.
B- Main flow of events
This use case begins when the user selects the Design mode in VRES/Vertex. The systems enters the Model Design mode and prompts the user for the model he wants to design. A list of "designable" models could be provided in a browsing window in order to assist the user in his choice, or the user could select a previously created model with the mouse. The user chooses a model. The system enters in the appropriate design mode for this model. The user performs the required design operations on the data and exits the design mode (he can save his work with the Save use cases). The operations in designing a model include: build geometric structure (for objects and tools), map texture on model, associate behavior with model, etc...To develop more deeply with team members as research progresses in each area. The use cases for each different type of operations in the design mode for models will have to be defined and documented when the research has reached more accurate results (under the form of prototypes).
C- Subflows
None
D- Alternative flows
The user should be allowed to exit the design mode if he changes his mind and return to the environment.

Use Case name:
DesignScene

Category: Use Case View
Documentation:
This use case is for designing a scene in VRES/Vertex. It is more relevant to VRES.

Feedback from Dion, Cote, Bernier, Houde, Poussart, Bergevin, Laurendeau, Ferrie, Simoneau, Borgeat, Vromet and Frey will be useful in this use case.
Flow of events
A- Preconditions
The execution of the appropriate Create use case must have been executed prior the related Design use case. Furthermore, since a scene is composed of several models (with associated behaviors), the Create/Design/Modify use cases for models and behaviors should have been executed previously.
B- Main flow of events
This use case begins when the user selects the Design mode in VRES/Vertex. The systems enters the Scene Design mode and prompts the user for the scene he wants to design. The user enters the name of the scene he wants to design. The system displays the set of elements that can compose a scene (list of models, behaviors, tools, etc) so the user can chose elements from these lists and associate them to include them in the scene. The user performs the required design operations on the data and exits the design mode (he can save his work with the Save use cases). The operations in designing a scene include: choose a geometic model and associate a behavior and position / orientation in the scene, choose tools and add them to models, etc...This DesignScene use case should be developed more deeply with team members as research progresses in each area. The use cases for each different type of operations in the design mode for scenes will have to be defined and documented when the research has reached more accurate results (under the form of prototypes).
C- Subflows
None
D- Alternative flows
The user should be allowed to exit the design mode if he changes his mind and return to the environment.

External Documents:
Abstract: No
State machine: No

```
Generalization:
   DesignGeneric (extends)
Associations:

         <no rolename> : ModifyScene in association <unnamed> (uses)
```

Use Case name:
DesignSequence

Category: Use Case View
Documentation:
This use case is for designing a sequence in VRES/Vertex.
To be discussed with team members. Feedback from Zaccarin and Mehran will be useful in this use case.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)
Associations:

<no rolename> : ModifySequence in association <unnamed> (uses)

Use Case name:
DesignWorkSession

Category: Use Case View
Documentation:
Allows the user to design a work session in VRES/Vertex.

To be discussed with team.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)
Associations:

<no rolename> : ModifyWorkSession in association <unnamed> (uses)

Use Case name:
DesignBehavior

Category: Use Case View
Documentation:
Allows the user to design a behavior in VRES/Vertex. In design mode, the user wants to use available physical models stored in a database of behaviors in order to build complex composite behaviors that can be linked to geometric models. These models will then be included in a scene for running a simulation.

Feedback from Frey and Laurendeau will be useful in this use case.

Flow of events
A- Preconditions
The execution of the appropriate Create use case must have been executed prior the related Design use case.
B- Main flow of events
This use case begins when the user selects the Design mode in VRES/Vertex. The systems enters the Design Behavior mode and prompts the user for the behavior he wants to design. A list of available behaviors could be provided in a browsing window in order to assist the user in his choice. The user chooses behaviors in the list of available behaviors (stored in the database) and combines them into a mode complex behavior. He sets the different parameters for the complex behavior using the HMI. The system checks whether the chosen parameters are valid or not and verify that the behaviors that are coupled are compatible.When his work is completed, the user exits the design mode (he can save his work with the Save use cases). This use case should be developed more deeply with team members as research progresses in each area. The use cases for each different type of operations in the design mode for behaviors will have to be defined and documented when the research has reached more accurate results (under the form of prototypes).
C- Subflows
None
D- Alternative flows
The user should be allowed to exit the design mode if he changes his mind and return to the environment.

External Documents:
Abstract: No

```
State machine:   No
Generalization:
   DesignGeneric (extends)
Associations:

         <no rolename> : ModifyBehavior in association <unnamed> (uses)
```

Use Case name:
DesignPlan

Category: Use Case View
Documentation:
This use case allows the user to design a plan in VRES/Vertex.

To be discussed with team. Feedback from Montreuil will be useful.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)
Associations:

<no rolename> : ModifyPlan in association <unnamed> (uses)

Use Case name:
DesignDemo

Category: Use Case View
Documentation:
This use case allows the user to design a demo in VRES/Vertex.

To be discussed with team.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)
Associations:

<no rolename> : ModifyDemo in association <unnamed> (uses)

Use Case name:
DesignSimulation

Category: Use Case View
Documentation:
This use case allows the user to design a simulation in VRES/Vertex.

To be discussed with team.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DesignGeneric (extends)

Use Case name:
DesignGeneric

Category: Use Case View
Documentation:
Generic use case for designing VRES/Vertex elements. In design mode, the user wants to use raw data in order to build components of the world into which a simulated intervention will take place. Most of the specific use cases in the Design category are still vague and need more interaction with team members in order to get a more precise description.They will be discussed and documented in the upcoming months

Flow of events
A- Preconditions
The execution of the appropriate Create use case must have been executed prior the related Design use case.
B- Main flow of events
This use case begins when the user selects the Design mode in VRES/Vertex. The system enters the generic Design mode and prompts the user for the element he wants to design. A list of "designable" elements could be provided in a browsing window in order to assist the user in his choice. The user chooses an element. The system enters in the appropriate design mode for this element (design modes may be different for models, behaviors, etc. See corresponding use cases for the design of each element category). The user performs the required design operations on the data and exits the design mode (he can save his work with

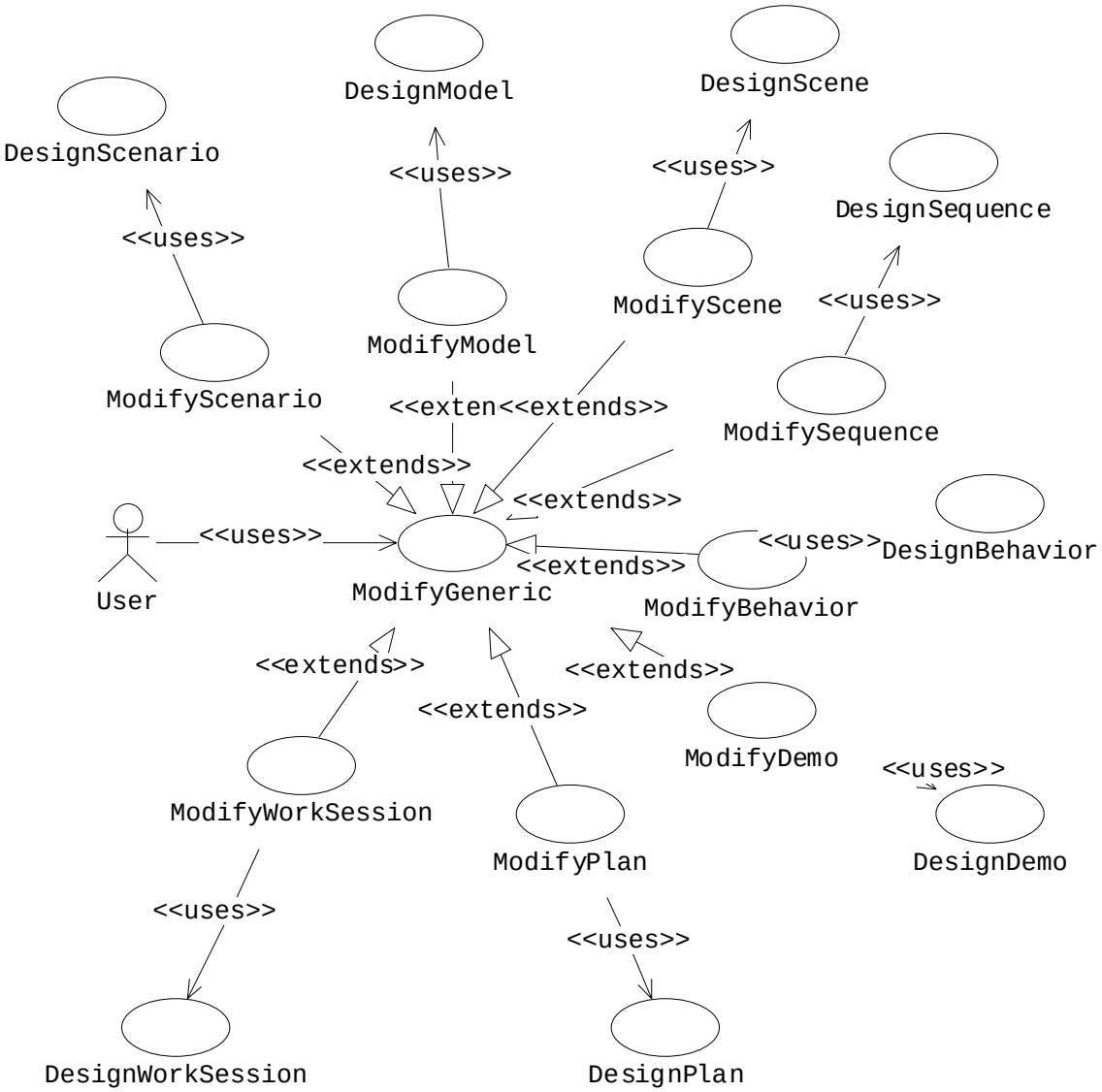
the Save use cases).
C- Subflows
None
D- Alternative flows
The user should be allowed to exit the design mode if he changes his mind and return to the environment.

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses}

4.17 Modify Use Cases - Complete and Detailed description

These use cases are concerned with the modification of elements that were created-designed in Vertex



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
ModifyGeneric

Category: Use Case View
Documentation:
Generic use case for modification of actions/operations in VRES/VERTEX.

This use case is very general and should be used for modifying existing elements of the environment. See specific use cases for details on each different type of modification.

Flow of events
A- Preconditions
In order to be modified, an element must have been created and designed. It can be loaded in the environment or may be stored in the appropriate database. If it is stored in the database, the element must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Generic mode in the HMI. The element is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesigGeneric use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
ModifyModel

Category: Use Case View
Documentation:
Use case for modifying a model created / designed / saved by another use case.

Flow of events

A- Preconditions
In order to be modified, a model must have been created and designed. It can be loaded in the environment or may be stored in the models database. If it is stored in the database, the model must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Model mode in the HMI. The model is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignModel use case.
C- Subflows
D- Alternative flows
External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignModel in association <unnamed> (uses)

Use Case name:
ModifyScenario

Category: Use Case View
Documentation:
Use case for modifying a scenario created by another use case.

Flow of events

A- Preconditions
In order to be modified, a scenario must have been created and designed. It can be loaded in the environment or may be stored in the scenarios database. If it is stored in the database, the scenario must be loaded into the environment before being modified.
B- Main flow of events
To be discussed by team. Uses the DesignScenario use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignScenario in association <unnamed> (uses)

Use Case name:
ModifyScene

Category: Use Case View
Documentation:
Use case for modifying a scene created by another use case.

Flow of events

A- Preconditions
In order to be modified, a scenbe must have been created and designed. It can be loaded in the environment or may be stored in the scenes database. If it is stored in the database, the scene must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Scene mode in the HMI. The scene is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignScene use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignScene in association <unnamed> (uses)

Use Case name:
ModifySequence

Category: Use Case View
Documentation:
Use case for modifying a sequence (see generic sequence generation use case) created by another use case.

Flow of events

A- Preconditions
In order to be modified, a sequence must have been created and designed. It can be loaded in the environment or may be stored in the sequences database. If it is stored in the database, the sequence must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Sequence mode in the HMI. The sequence is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignSequence use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignSequence in association <unnamed> (uses)

Use Case name:
ModifyBehavior

Category: Use Case View
Documentation:
Use case for modifying a behavior created by another use case.

Flow of events

A- Preconditions
In order to be modified, a behavior must have been created and designed. It can be loaded in the environment or may be stored in the behaviors database. If it is stored in the database, the behavior must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Behavior mode in the HMI. The behavior is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignBehavior use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignBehavior in association <unnamed> (uses)

Use Case name:
ModifyDemo

Category: Use Case View
Documentation:
Use case for the modification of a demo created by another use case.

To be discussed by team.

Flow of events

A- Preconditions
In order to be modified, a Demo must have been created and designed. It can be loaded in the environment or may be stored in the demos database. If it is stored in the database, the demo must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Demo mode in the HMI. The demo is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignDemo use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignDemo in association <unnamed> (uses)

Use Case name:
ModifyPlan

Category: Use Case View
Documentation:
Use case for the modification of a plan created by another use case.

Flow of events

A- Preconditions
In order to be modified, a plan must have been created and designed. It can be loaded in the environment or may be stored in the plans database. If it is stored in the database, the plan must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify Plan mode in the HMI. The plan is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the DesignPlan use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignPlan in association <unnamed> (uses)

Use Case name:
ModifyWorkSession

Category: Use Case View
Documentation:
Use case for the modification of a work session created by another use case.

To be discussed by team.

Flow of events

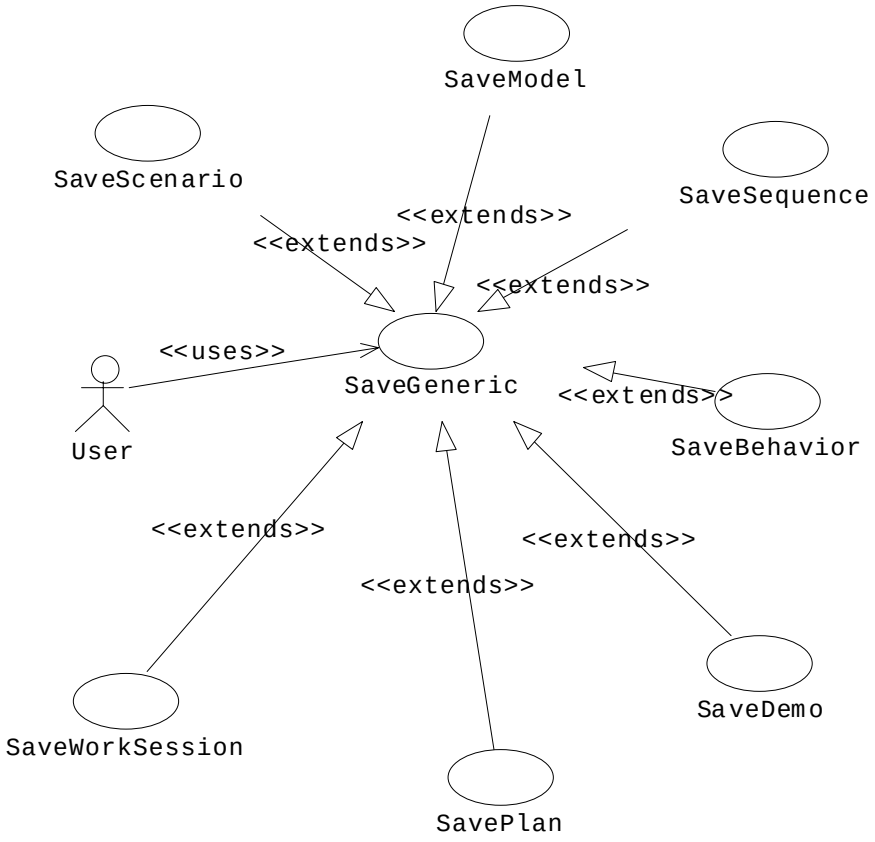
A- Preconditions
In order to be modified, a WorkSession must have been created and designed. It can be loaded in the environment or may be stored in the Worksession of the user. If it is stored in the user's account the worksession must be loaded into the environment before being modified.
B- Main Flow of events
This use case begins when the user selects the Modify WorkSession mode in the HMI. The Worksession is brought into the environment and displayed on the screen. After this step, the remaining operations are the same as the ones that are encountered in the Design WorkSession use case.
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
ModifyGeneric (extends)
Associations:

<no rolename> : DesignWorkSession in association <unnamed> (uses)

4.18 Save Use Cases - Complete and Detailed Description

These use cases deal with the action of saving work performedduring a Vertex Session.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
SaveScenario

Category: Use Case View
Documentation:
Use case for saving a scenario for further use.
Flow of events
A- Preconditions
An scenario should reside in the VRES/Vertex environment before it can be saved into the scenario database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the scenario he wants to save and the database into which the scenario should be saved. A browsing window then displays all the available databases for saving the scenarios. A new database could be created (see the CreateDatabase use case). The user selects the database and scenario he wants to save. The scenario is saved by the system and informs the user that the scenario has been saved correctly. The systems checks whether a scenario with the same name is already saved in the database and asks the user to confirm that the scenario should be replaced by the current one.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the scenario he wished to save has been processed correctly.

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveModel

Category: Use Case View
Documentation:
Use case for saving a model created/modified by another use case.

Flow of events
A- Preconditions
An model should reside in the VRES/Vertex environment before it can be saved into the models database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the model he wants to save and the database into which the model should be saved. A browsing window then displays all the available databases for saving the models. A new database could be created (see the CreateDatabase use case). The user selects the database and model he wants to save. The model is saved by the system and informs the user that the model has been saved correctly. The systems checks whether a model with the same name is already saved in the database and asks the user to confirm that the model should be replaced by the current one.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the model he wished to save has been processed correctly.

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveSequence

Category: Use Case View
Documentation:
Use case for saving a sequence created/modified by another use case.

Flow of events
A- Preconditions
An sequence should reside in the VRES/Vertex environment before it can be saved into the sequence database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the sequence he wants to save and the database into which the sequence should be saved. A browsing window then displays all the available databases for saving the sequences A new database could be created (see the CreateDatabase use case). The user selects the database and sequence he wants to save. The sequence is saved by the system and informs the user that the sequence has been saved correctly. The systems checks whether a sequence with the same name is already saved in the database and asks the user to confirm that the sequence should be replaced by the current one.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the sequence he wished to save has been processed correctly.

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveBehavior

Category: Use Case View
Documentation:
Use case for saving a behavior created/modified by another use case.

Flow of events
A- Preconditions
A behavior should reside in the VRES/Vertex environment before it can be saved into the behavior database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the behavior he wants to save and the database into which the behavior should be saved. A browsing window then displays all the available databases for saving the behavior. A new database could be created (see the CreateDatabase use case). The user selects the database and behavior he wants to save. The behavior is saved by the system and informs the user that the behavior has been saved correctly. The systems checks whether a behavior with the same name is already saved in the database and asks the user to confirm that the behavior should be replaced by the current one.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the behavior he wished to save has been processed correctly.

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveDemo

Category: Use Case View
Documentation:
Use case for saving a demo created/modified by another use case.

Flow of events
A- Preconditions
B. Main flow of events
C. Subflows
D. Alternative flows

External Documents:

Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SavePlan

Category: Use Case View
Documentation:
Use case for saving a plan created/modified by another use case.

Flow of events
A- Preconditions
An plan should reside in the VRES/Vertex environment before it can be saved into the plan database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the plan he wants to save and the database into which the plan should be saved. A browsing window then displays all the available databases for saving the plans. A new database could be created (see the CreateDatabase use case). The user selects the database and plan he wants to save. The plan is saved by the system and informs the user that the plan has been saved correctly. The systems checks whether a plan with the same name is already saved in the database and asks the user to confirm that the plan should be replaced by the current one.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the plan he wished to save has been processed correctly.

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveWorkSession

Category: Use Case View
Documentation:
Use case for saving a work session created/modified by another use case.
Flow of events
A- Preconditions
B. Main flow of events
C. Subflows
D. Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
SaveGeneric (extends)

Use Case name:
SaveGeneric

Category: Use Case View
Documentation:
Generic use case for saving VRES/VERTEX elements. It is a very general and abstract use case that should capture the general operations that are relevant to the saving of an element into a VRES/Vertex database. The database the element is saved into depends on the specific use case that is considered.

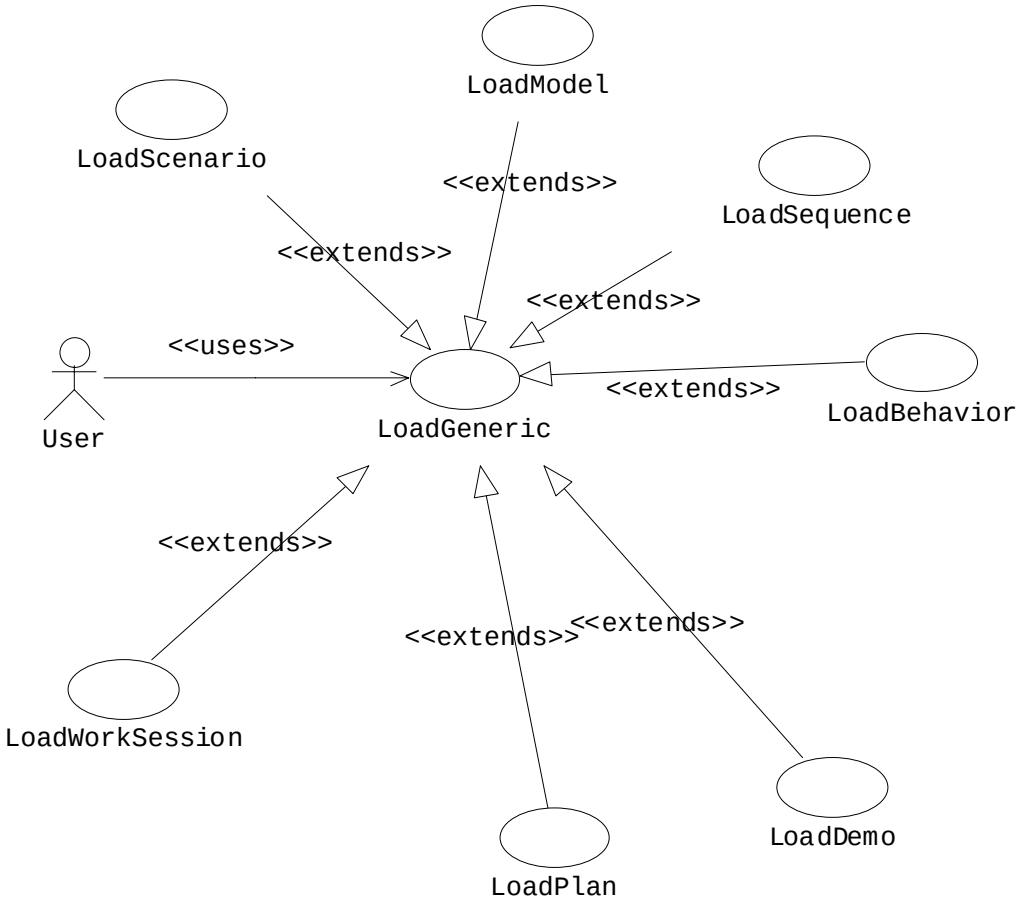
Flow of events
A- Preconditions
An element should reside in a the environment before it can be saved into a database.
B. Main flow of events
This use case begins when the user selects the save command in the VRES/Vertex menu. Following the activation of the save command, a menu should prompt the user for the name of the element he wants to save. A browsing window then displays the databases into which the element could be saved (a new database could also be created at this moment (see the CreateDatabase use case)) and a list of the elements that can be saved at this moment. The user selects one element or type the name of the element he wants to save in a text field and selects a database into which the elements should be saved. The element is saved by the system and informs the user that the element has been saved correctly (a message displayed in a pop-up window would provide enough feedback.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the element he wished to save is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several elements simultaneously and to have been saved in sequence. In that case, it would be necessary to provide the means for associating several elements to several databases...(to develop later).

External Documents:
Abstract: No
State machine: No
Associations:

```
<no rolename> : User in association <unnamed> (uses)
```

4.19 Load Use Cases - Complete and Detailed Description

These use cases are concerned with the action of loading previously saved operations in Vertex.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
LoadScenario

Category: Use Case View
Documentation:
Use case for loading a (previously saved) scenario into VRES/VERTEX.

Flow of events
A- Preconditions
An scenario should reside in a database before it can be loaded into the environment.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the scenario he wants to load into the environmen. A browsing window then displays all the available scenarios in the database. The user selects one scenario or type the name of the scenario he wants to load in a text field. The scenario is loaded by the system and informs the user that the scenario has been loaded correctly.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the scenario he wished to load is not available or that the name given in the text field is incorrect. At this time, it seems irrelevant to allow the user to load more than one scenario simultaneously.

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadModel

Category: Use Case View
Documentation:
Use case for loading a (previously saved) model into VRES/VERTEX.

Flow of events
A- Preconditions
A model should reside in a database before it can be loaded into the environment.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the element he wants to load into the environment (type: model in this use case). A browsing window then displays all the available documents of this type in the database. The user selects one model or type the name of the model he wants to load in a text field. The model is loaded by the system and informs the user that the model has been loaded correctly. In the case of a model, a graphic rendering of the model would be an adequate feedback.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the model he wished to load is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several models simultaneously and to have been loaded in sequence.

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadSequence

Category: Use Case View
Documentation:
Use case for loading a (previously saved) sequence into VRES/VERTEX. A sequence is a group of images (2d and/or 3D) that were acquired sequentially and that can be associated to a same trajectory of the acquisition sensor. The images in a sequence are usually in overlap.

Flow of events
A- Preconditions
A sequence should reside in a database before it can be loaded into the environment.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the element he wants to load into the environment (type: a sequence in this use case). A browsing window then displays all the available documents of this type in the database. The user selects one sequence or type the name of the sequence he wants to load in a text field. The sequence is loaded by the system and informs the user that the element has been loaded correctly. A display of the images into the VRES/Vertex environment would be an adequate feedback.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the sequence he wished to load is not available or that the name given in the text field is incorrect. At this time, it does not seems necessary to have several sequences loaded simultaneously.

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadBehavior

Category: Use Case View
Documentation:
Use case for loading a (previously saved) behavior into VRES/VERTEX.

Flow of events
A- Preconditions
A behavior should reside in a database before it can be loaded into the environment. Models should have been loaded in the environment since a behavior must be attached to a model.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the element he wants to load into the environment (type: a behavior in this use case). A browsing window then displays all the available documents of this type in the database. The user selects one behavior or type the name of the behavior he wants to load in a text field. The behavior is loaded by the system and informs the user that the behavior has been loaded correctly. The user then attach a behavior to a model previously loaded in the environment (see the LoadModel use case). The user is prompted if the association is invalid.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the behavior he wished to load is not available or that the name given in the text field is incorrect. At this time, it would be interesting if several behaviors could be loaded simultaneously and attached to models one after the other by prompting the user for an association between models and behaviors.
Note: more than one behavior can be attached to a model: for instance, a mechanical behavior and a thermic behavior could be attached to the same geometric model.

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadDemo

Category: Use Case View
Documentation:
Use case for loading a (previously saved) demo into VRES/VERTEX.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadPlan

Category: Use Case View
Documentation:
Use case for loading a (previously saved) plan into VRES/VERTEX.

Flow of events
A- Preconditions
A plan should reside in a database before it can be loaded into the environment.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the element he wants to load into the environment (type: a plan in this use case). A browsing window then displays all the available documents of this type in the database. The user selects one plan or type the name of the plan he wants to load in a text field. The plan is loaded by the system and informs the user that the plan has been loaded correctly. The type of feedback that is provided to the user remains to be decided.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the plan he wished to load is not available or that the name given in the text field is incorrect. At this time, it does not seems necessary to have several plans loaded simultaneously.

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadWorkSession

Category: Use Case View
Documentation:
Use case for loading a (previously saved) work session into VRES/VERTEX.

Flow of events
A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
LoadGeneric (extends)

Use Case name:
LoadGeneric

Category: Use Case View
Documentation:
Generic use case for loading elements in VRES/VERTEX. It is a very general and abstract use case that should capture the general operations that are relevant to the loading of an element into the VRES/Vertex environment.

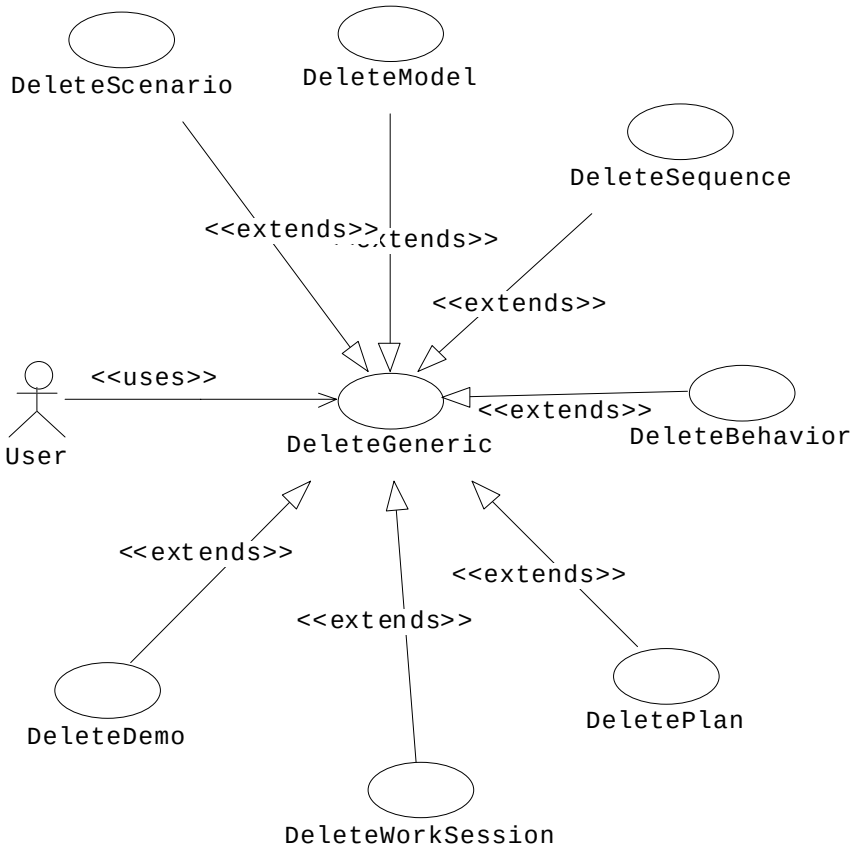
Flow of events
A- Preconditions
An element should reside in a database before it can be loaded into the environment.
B. Main flow of events
This use case begins when the user selects the load command in the VRES/Vertex menu. Following the activation of the load command, a menu should prompt the user for the type and name of the element he wants to load into the environment (type: model, scenario, etc). A browsing window then displays all the available documents of this type in the database. The user selects one element or type the name of the element he wants to load in a text field. The element is loaded by the system and informs the user that the element has been loaded correctly.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the element he wished to load is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several elements simultaneously and to have been loaded in sequence.

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

4.20 Delete Use Cases - Complete ande detailed Description

These use cases are coincerned with the deletion of elements that were previously created-designed-modified in Vertex.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
DeleteScenario

Category: Use Case View
Documentation:
Use case for deleting a scenario from VRES/VERTEX.
The scenario is deleted from the current environment.
A special use case describes the procedure for
deleting a scenario from a database.

Flow of events
A- Preconditions
A scenario should reside in a the environment before
it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete
command in the VRES/Vertex menu. Following the
activation of the delete command, a menu should prompt
the user for the name of the scenario he wants to
delete (for the first phases of the project the
environment should contain only one scenario). A
browsing window then displays a list of the scenarios
that can be deleted at this moment. The user selects
one scenario or type the name of the scenario he wants
to delete in a text field. The scenario is deleted by
the system and informs the user that the scenario has
been deleted correctly (a message displayed in a
pop-up window would provide enough feedback.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the scenario
he wished to delete is not available or that the name
given in the text field is incorrect. It would be
interesting to allow the user to select several
scenarios (later in the project when several scenarios
may be reside simultaneously in the environment)
simultaneously and to have them deleted in sequence.

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteModel

Category: Use Case View
Documentation:
Use case for deleting a model from VRES/VERTEX.

The model is deleted from the current environment. A special use case describes the procedure for deleting a model from a database.

Flow of events
A- Preconditions
A model should reside in the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, a menu should prompt the user for the name of the model he wants to delete or the user could point on a model that is currently being displayed in the work environement and then issue the delete command on this model. The model is deleted by the system and informs the user that the element has been deleted correctly (a message displayed in a pop-up window would provide enough feedback. The system should remove the link(s) between the model and its associated behaviors and refresh the display of the environment.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the model he wished to delete is not available or that the name given in the text field is incorrect (when the model is selected through a menu). It would be interesting to allow the user to select several models simultaneously and to have them deleted in sequence (still removing links with associated behaviors and performing a refresh on the display).

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteSequence

Category: Use Case View
Documentation:
Use case for deleting a sequence from VRES/VERTEX.

The sequence is deleted from the current environment. A special use case described the procedure for deleting a sequence from a database.

Flow of events
A- Preconditions
A sequence should reside in a the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, a menu should prompt the user for the name of the sequence he wants to delete. A browsing window then displays a list of sequences that can be deleted at this moment. The user selects one sequence or type the name of the sequence he wants to delete in a text field. The sequence is deleted by the system and informs the user that the sequence has been deleted correctly (a message displayed in a pop-up window would provide enough feedback).
C. Subflows
None
D. Alternative flows
The user is informed by the system that the sequence he wished to delete is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several sequences simultaneously and to have them deleted.

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteBehavior

Category: Use Case View
Documentation:
Use case for deleting a behavior from VRES/VERTEX.

The behavior is deleted from the current environment. A special use case described the procedure for deleting a behavior from a database.

Flow of events
A- Preconditions
A behavior should reside in the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, a menu should prompt the user for the name of the behavior he wants to delete. A browsing window then displays a list of the behaviors that can be deleted at this moment. The user selects one behavior or type the name of the behavior he wants to delete in a text field. The behavior is deleted by the system and informs the user that the behavior has been deleted correctly (a message displayed in a pop-up window would provide enough feedback. The system should remove the link(s) between the behavior and its associated models and update the status of each model previously linked with this behavior.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the behavior he wished to delete is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several behaviors simultaneously and to have them deleted in sequence (still removing links with associated models and updating the models previously linked with these behaviors.
External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteDemo

Category: Use Case View
Documentation:
Use case for deleting a demo from VRES/VERTEX.

Flow of events
A- Preconditions
B. Main flow of events
C. Subflows
D. Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeletePlan

Category: Use Case View
Documentation:
Use case for deleting a plan from VRES/VERTEX.

The plan is deleted from the current environment. A special use case described the procedure for deleting a plan from a database.

Flow of events
A- Preconditions
A plan should reside in a the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, a menu should prompt the user for the name of the plan he wants to delete. A browsing window then displays a list of plans that can be deleted at this moment. The user selects one plan or type the name of the plan he wants to delete in a text field. The plan is deleted by the system and informs the user that the plan has been deleted correctly (a message displayed in a pop-up window would provide enough feedback).
C. Subflows
None
D. Alternative flows
The user is informed by the system that the plan he wished to delete is not available or that the name given in the text field is incorrect. It would be interesting to allow the user to select several plans simultaneously and to have them deleted.

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteWorkSession

Category: Use Case View
Documentation:
Use case for deleting a work session from VRES/VERTEX.

Flow of events
A- Preconditions
B. Main flow of events
C. Subflows
D. Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
DeleteGeneric (extends)

Use Case name:
DeleteGeneric

Category: Use Case View
Documentation:
Generic use case for deleting previously created/modified VRES/VERTEX elements.

It is a very general and abstract use case that should capture the general operations that are relevant to the deletion of an element into a VRES/Vertex database. The element is deleted from the current environment. A special use case described the procedure for deleting an element from a database.

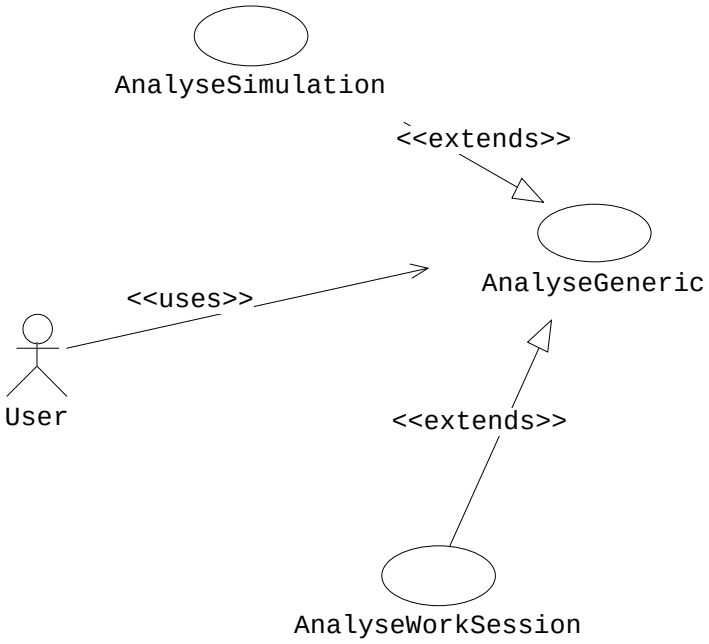
Flow of events
A- Preconditions
An element should have been previously loaded in the environment before it can be deleted.
B. Main flow of events
This use case begins when the user selects the delete command in the VRES/Vertex menu. Following the activation of the delete command, the user selects (through the HMI) the element he wants to delete. A browsing window also displays a list of the elements that can be deleted at this moment. The user selects one element or type the name of the element he wants to delete. The element is deleted by the system and informs the user that the element has been deleted correctly.
C. Subflows
None
D. Alternative flows
The user is informed by the system that the element he wished to delete is not available or that the name given in a text field is incorrect (assuming the user can choose objects with the mouse or with menus and textfields). It would be interesting to allow the user to select several elements simultaneously and to have them deleted in sequence.

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

4.21 Analyse Use Cases - Complete and Detailed Description

These use cases are concerned with the analysis of asequence of operations in Vertex.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
AnalyseSimulation

Category: Use Case View
Documentation:
Use case for analysing the results of a simulation in VRES/VERTEX.

Flow of events

A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows
External Documents:
Abstract: No
State machine: No
Generalization:
AnalyseGeneric (extends)

Use Case name:
AnalyseWorkSession

Category: Use Case View
Documentation:
Use case for the analysis/assessment of a work session in VRES/VERTEX.

Flow of events

A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Generalization:
AnalyseGeneric (extends)

Use Case name:
AnalyseGeneric

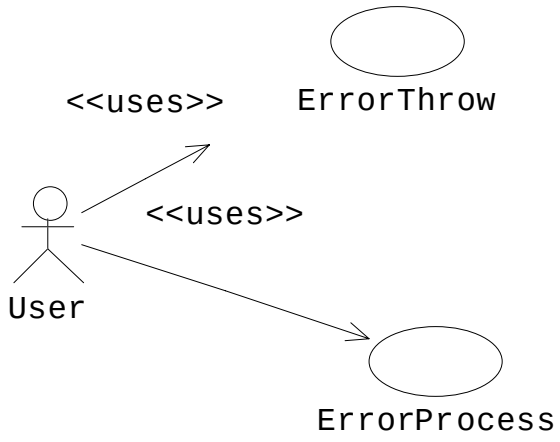
Category: Use Case View
Documentation:
Use case for the analysis of actions/results/etc in VRES/VERTEX.
To be defined...

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

4.22 Error Use Cases - Complete and Detailed Descitption

These use cases are concerned with the processing of errors / exceptions in the course of a Vertex manipulation.



Class name:
User

Category: Use Case View
Documentation:
User of the VRES/Vertex system. The user can interact with the system during a simulation or a real-time run.

We may have to refine our description of the user as the project evolves. To be more specific, the VRES/Vertex system may be used by several types of users: professionnal computer scientists, professionals in site maintenance, operators with little knowledge of computers...

Stereotype: Actor
External Documents:
Export Control: Public
Cardinality: n
Hierarchy:
Superclasses: none
Associations:

<no rolename> : CreateGeneric in association <unnamed> (uses)
<no rolename> : AbortGeneric in association <unnamed> (uses)
<no rolename> : VisualizeGeneric in association <unnamed> (uses)
<no rolename> : DesignGeneric in association <unnamed> (uses)
<no rolename> : ModifyGeneric in association <unnamed> (uses)
<no rolename> : SaveGeneric in association <unnamed> (uses)
<no rolename> : LoadGeneric in association <unnamed> (uses)
<no rolename> : DeleteGeneric in association <unnamed> (uses)
<no rolename> : AnalyseGeneric in association <unnamed> (uses)
<no rolename> : ErrorProcess in association <unnamed> (uses)
<no rolename> : ErrorThrow in association <unnamed> (uses)

State machine: No
Concurrency: Sequential
Persistence: Transient

Use Case name:
ErrorProcess

Category: Use Case View
Documentation:
Use case for the processing of an error that was previously thrown. The mechanism for error processing is still to be defined.

Flow of events

A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows
External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

Use Case name:
ErrorThrow

Category: Use Case View
Documentation:
Use case for throwing an error in VRES/VERTEX. We will have to define the classes for the hierarchy of errors that can occur in the different use cases.

Flow of events

A- Preconditions
B- Main flow of events
C- Subflows
D- Alternative flows

External Documents:
Abstract: No
State machine: No
Associations:

<no rolename> : User in association <unnamed> (uses)

CHAPTER 5 References

[1] G. Booch, "*Object-Oriented Analysis and Design With Applications*," Addison-Wesley, 1994