# An ontology-based approach to knowledge representation for Computer-Aided Control System Design

Carmen Benavides[a,*], Isaías García[a], Héctor Alaiz[a], Luis Quesada[b],
[a]*Dept. of Electrical and Systems Engineering. Universidad de León. Escuela de Ingenierías. Campus de Vegazana, 24071 León, Spain*
[b]*Insight Centre for Data Analytics. University College Cork.Cork, Ireland*

**Abstract.** Different approaches have been used in order to represent and build control engineering concepts for the computer. Software applications for these fields are becoming more and more demanding each day, and new representation schemas are continuously being developed. This paper describes a study of the use of knowledge models represented in ontologies for building Computer Aided Control Systems Design (CACSD) tools. The use of this approach allows the construction of formal conceptual structures that can be stated independently of any software application and be used in many different ones. In order to show the advantages of this approach, an ontology and an application have been built for the domain of design of lead/lag controllers with the root locus method, presenting the results and benefits found.

Keywords: Conceptual modeling, Data and knowledge visualization, Ontologies, Computer-Aided Control System Design

[*] Corresponding author, Escuela de Ingenierías, Universidad de León, Campus de Vegazana, s/n 24071 León, Spain. E-mail: carmen.benavides@unileon.es. Tlf +34987291000 ext. 5395. Fax: +34987291790

## 1. Introduction

Representing control engineering data and knowledge in the computer has always been a concern in CACSD (Computer- Aided Control System Design) software [1], [2]. The growing complexity of the systems, projects and tools related to control engineering makes it necessary to study and find knowledge representation schemas that can face the current software demands and the future trends of the discipline. This work studies the application of knowledge modelling techniques, for addressing these challenges. This approach is based on the use of explicit, computable knowledge models called ontologies [3] which are conceptual structures existing on their own, independently of any specific application.

In order to clarify and study the benefits of this approach, an ontology and a practical application have been developed. The application is devoted to the design of lead/lag controllers by using the root locus method. This application is able to perform a design based on the knowledge that has been previously built into the ontology, storing a data record containing all the concepts, tasks and values involved in the design process carried out, in such a way that it is able to present this information to the user with a level of detail that has not been achieved in any previous application of this kind. The tool is also capable of reacting to user interaction, offering explanations about any of the concepts or steps involved in the process.

The remainder of this paper is organized as follows. Section 2 reviews the different activities within control and systems engineering where data and knowledge representation is a concern, showing the usual approaches and developments that have been used to cope with this problem. Section 3 introduces the knowledge modelling techniques based on the construction of ontologies, showing how they are already being used in many fields relating to systems engineering. Section 4 briefly describes the domain being modelled, while section 5 gives an overview of the general architecture of the system. Section 6 presents the ontology about the design of lead/lag controllers that has been developed, including the rationales of its design and giving examples of some the conceptual structures that are built inside. Finally, section 7 is devoted to show and describe the graphical CACSD application by which the user is able to interact with the application and section 8 contains some discussion relating this work to previously existing ones and stressing the advantages of the approach which has been presented here.

## 2. Evolution of the knowledge representation in CACSD software

Data and knowledge representation in CACSD software is a crucial concern for obtaining better software tools. Some of the motivating issues and problems guiding the research in this field will be briefly discussed here. The aim of this exposition is to show how knowledge representation schemes are becoming more and more complex as the field of CACSD evolves and how the approach presented in this paper relates to the mentioned techniques.

### *2.1. Languages and tools for modeling and simulation*

Finding convenient ways of representing control systems allows an easier development and use of modelling and simulation software. With the experience obtained in the first, structured-language-based tools like CSSL (Continuous System Simulation Language) or ACSL (Advanced Continuous Simulation Language) [4], some advances were made in order to reuse previously programmed models without having to build them from scratch each time a simulation was built. DYMOLA [5] is an early example of this reusing approach, and its successor, OMOLA [6], is one of the first applications using the object orientation paradigm for representing components. MODELICA [7] is based on the use of model libraries able to represent systems of different nature (electrical, mechanical, hydraulic, etc) by means of reusable components.

As these tools have shown, the more powerful the knowledge representation schema is, the more reusable the models are. Reusability and encapsulation are two of the key concepts guiding the research in modelling and simulation languages.

### *2.2. Interaction between humans and machines*

The interaction between the user and a CACSD application is another one of the issues guiding the evolution of this kind of software. Though different types of users need different levels of interaction, the development of these tools over the years shows an evolution from the initial command-line-based ones to the complex graphical user interfaces with a high level of abstraction.

Another great influence in the evolution of CACSD software is the development of the UML language (Unified Modeling Language) [8] and its adaptation to the systems engineering field: SysML (Systems Modeling Language) [9] [10]. By using these languages, it is easier to describe the systems, the design goals, the project characteristics and its evolution, etc. It also eases the interaction among machines as well as the automatic translation of the representing structures to other ones (executable code, documentation, etc). Object orientation and SysML are today widely adopted tools to represent systems and control engineering knowledge.

### *2.3. Integration of applications*

Within the life-cycle of a systems or control engineering project many different computer applications are involved. All these applications use different representations for their data and this, often, makes them incompatible, making it necessary to perform a translation whenever data pass from one application to the next. With this issue in mind, different efforts have been carried out in order to get completely integrated architectures.

The first attempts devoted to building software platforms for integrating systems and control engineering tools date back to the middle 1980. Some examples of these platforms are ECSTASY (Environment for Control System Theory and Synthesis) [11], GE-MEAD (General Electric - Multidisciplinary Expert-aided Analysis and Design) [12] or ANDECS [13]. All these systems are based on the existence of a central repository where the data of a given project is stored in a standardized format. ANDECS is the system that has the most complex data representation structure, consisting of a database where objects representing the involved elements are stored [14]. Data from other applications and manufacturers is translated into the internal format of the repository.

As pointed out in [2], the use of a centralized repository for the data proved not to be an efficient approach: the different manufacturers could change their data format without prior warning, making the programming of wrappers to make the translation to the centralized format very difficult. With this in mind, in the 1990's new efforts were carried out in order to find languages and tools that would facilitate the translation process and that could even be promoted as representation standards. The EXPRESS language [15], developed by ISO (International Organization for Standardization) was introduced in the CACSD domain as a common high level language for the integration of applications [16], however it did not gain wide acceptation due to its inherent complexity and the lack of usable tools to manage it.

The protocol AP-233, devoted to "Systems Engineering Data Representation" [17] is another effort to describe generic data structures for the design and fabrication processes. AP-233 uses both EXPRESS and UML for representing the data structures [18]. AP-233 overlaps, in some of its functionalities, with SysUML [19]

## 3. Ontologies as knowledge representation entities for systems engineering

As was stated in section 2, the evolution of CACSD tools and techniques has pursued the creation of more and more elaborated data and knowledge representation schemas. From libraries of models and components to the use of object orientation techniques or the development of standardized schemas and languages (UML, SysML, EXPRESS, AP-233, etc), the effort is aimed at easing the design, development and reuse of software units for building systems and control engineering tools.

Most of these approaches were based on the application of the developments achieved in the field of software engineering, conveniently applied or adapted to the systems and control engineering field. The aim of this work is to show how the research and developments from field of knowledge representation can be applied to further improve the mentioned techniques and tools, showing how knowledge representation with ontologies enhances the capabilities of the object orientation paradigm, easing the reuse of software models.

### 3.1. Ontologies

Starting in the early 1990's, the idea of ontologies as computer-based knowledge representation schemas appeared and evolved, soon becoming an appealing choice for representing knowledge in the computer, especially in complex domains. Ontologies use to be defined as "a formal and explicit representation of a conceptualization" [20]. One can think of these conceptual structures as if they were a dictionary or a thesaurus containing the terms and concepts appearing in a given domain but augmented with a number of formally specified relationships and axioms holding among those concepts. The mentioned formalization allows the automatic processing of the conceptual structure by a computer. Ontological engineering is a very active research field today, most of this research is aimed at the application of these ideas in the Internet, in what is called "the semantic web" [21], but many other fields of application are also being explored. The impact of ontologies for e-Science applications is remarkable [22], including systems engineering as will be seen in the next section.

### 3.2. Ontologies for systems engineering

The use of ontologies in the systems engineering domain has gained popularity in recent years. Research aimed at practical applications of ontologies in this field can be found in many areas like advising systems [23], product assembly [24], process engineering [25], [26], automation and robotics [27] aeronautics [28], earth sciences (SWEET (The Semantic Web for Earth and Environmental Terminology) [29], automation engineering [30] supply chain management [31], [32] manufacturing systems engineering [33] [34], design [35], system configuration [36], engineering information management [37], decision making [38] simulation [39], [40], etc.

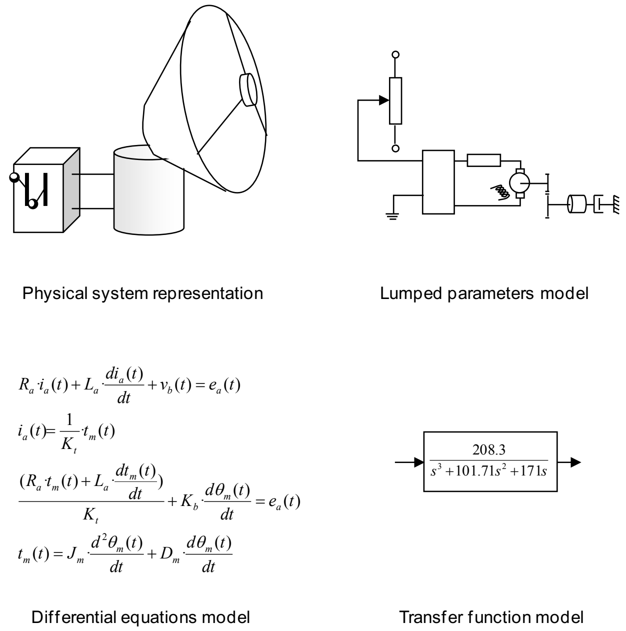
Ontologies are also being used today in substitution or combination with other representation schemas that have been previously mentioned in section 2. In [41] a study is presented comparing the use of what are called "lightweight ontologies" (developed with UML or similar formalisms) and "heavyweight ontologies" (that include complex reasoning processes based on some sort of logic) for manufacturing knowledge representation. Another example is the use of ontology defining languages instead of EXPRESS [42], [43] for describing systems engineering models in the context of the project STEP [44], [45]. A third example could be the integration of STEP, UML and the web ontology language (OWL) (a language for the definition of ontologies developed by the Wide World Web Consortium, W3C) in the Pan Galactic Engineering Framework [46]. Industry Foundation Classes (IFC) are also being represented by ontologies as is the case of the ifcOWL project [47], [48]. A semantic interoperability framework for Computer Aided Design (CAD) and related ontologies are described in [49] where not only product shape, but parameters, constraints, features or design history for the product is captured.

## 4. The domain of study: lead lag controller design with the root locus method

This section is an introduction to the domain of control engineering and also to the design technique that was chosen to be modelled and represented in the ontology: the root locus method. Among the many different design procedures and techniques that can be found in the field, the root locus method was chosen because of the conceptual challenges that it posed from the knowledge representation point of view. Compared to other, more modern methods, the root locus is a graphical one involving heuristic knowledge and is widely used as an educational tool for undergraduate students.

### 4.1. Overview of control engineering

The purpose of control engineering is to modify the dynamical behaviour of a given system in order to make it operate under some desired working conditions. A typical example is the control of an antenna positioning system (fig. 1). Some conditions may be imposed at design time for the characteristics of the response of the antenna to a change in the reference (desired) angular position (e.g. the time the antenna takes to reach its final position, the difference between the desired and the actual final position, etc.).



Physical system representation

Lumped parameters model

$$R_a \cdot i_a(t) + L_a \cdot \frac{di_a(t)}{dt} + v_b(t) = e_a(t)$$

$$i_a(t) = \frac{1}{K_t} \cdot t_m(t)$$

$$\frac{(R_a \cdot t_m(t) + L_a \cdot \frac{dt_m(t)}{dt})}{K_t} + K_b \cdot \frac{d\theta_m(t)}{dt} = e_a(t)$$

$$t_m(t) = J_m \cdot \frac{d^2\theta_m(t)}{dt} + D_m \cdot \frac{d\theta_m(t)}{dt}$$

Differential equations model

$$\boxed{\frac{208.3}{s^3 + 101.71s^2 + 171s}}$$

Transfer function model

Figure 1. Antenna positioning system with its lumped parameters, differential equations and transfer function models

In order to achieve the design goals, some analysis and design processes must be carried out by using mathematical models of the physical systems involved. Figure 2 shows two possible example response curves for the controlled system (the curve represent the angular position of the antenna vs time): The first (fig 2a), slower one, has no oscillatory response but lasts longer to reach its final position. The second response (fig 2b) is faster, but the antenna surpasses the intended final angular position and presents some oscillations until it reaches the final value.

Control engineering uses a number of mathematical models in order to study the behaviour of systems. Classical control theory, in particular, uses the so-called Laplace transformation that, conveniently applied to the set of differential equations describing the system, produces a mathematical representation consisting in a polynomial quotient that reflects the relationship between the output and the input signals to the system. This polynomial quotient is called the "transfer function model" of the corresponding system (see figure 1). The power of this representation lies in the fact that the response of the system to a given input can be derived by studying the position of the roots of the polynomials in the numerator and denominator of the transfer function, that are called, respectively, the zeros and poles of the system.

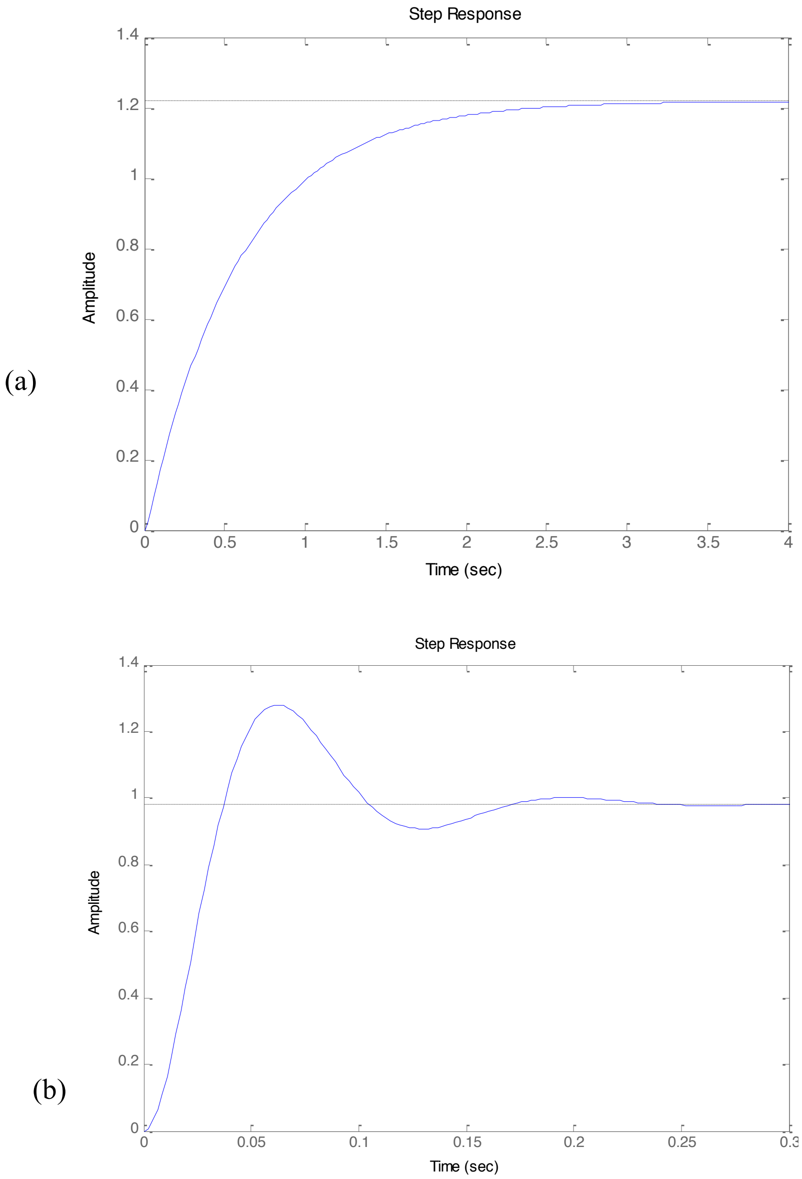


Figure. 2. Different responses for two control strategies of an antenna positioning system.

Any subsystem is graphically represented by a transfer function block. These blocks are interconnected in a diagram representing the physical interconnection of the real subsystems. The typical control schema, shown in figure 3, consists of a feedback loop that measures the current position of the antenna, a comparator producing an error signal (difference between the reference and the current position) and a controller, which is a component that, according to the error signal, drives the input to the antenna in order to have it working under the desired conditions.

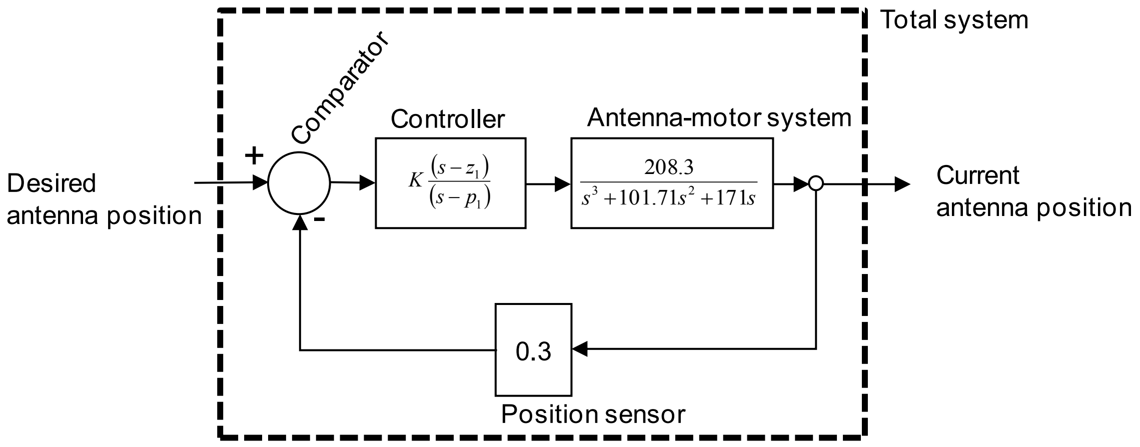


Figure 3. Antenna positioning system with feedback loop and controller.

The aim of the control design process is to find the appropriate parameters for the controller ($K$, $p_1$ and $z_1$ in figure 3) that will make the total system (the one including all the subsystems having, as the input, the required position of the antenna and, as the output, the actual position of the antenna) work under the desired conditions.

### *4.2. Controller design with the root locus method*

Controller design with the root locus method is a graphical process that tries to place the poles of the total system according to its desired dynamical behaviour. First, the pole ($p_1$) and zero ($z_1$) positions of the controller are calculated, according to different placement algorithms, keeping the parameter $K$ variable (see figure 3). Varying the value of K from zero to infinity allows the sketch of the path that the poles of the total system draw in the argand diagram (real vs imaginary axis). This graph is called the root locus (curves in figure 4). This way, the most suitable value for K can be chosen (small squares in figure 4 represent the position of the poles of the total system for a given value of K). The controller must then be tested by running simulations on the transfer function model and also in the real system. If the behaviour does not meet the initial performance criteria, a re-design process must be performed.

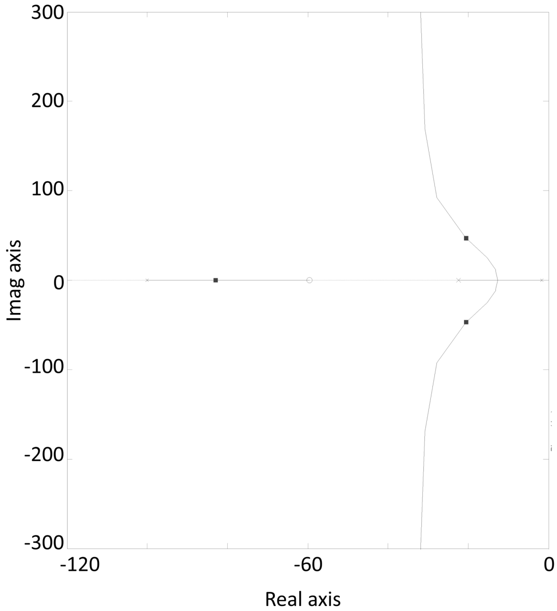


Figure. 4. Root locus of the total system in figure 3. Squared dots shows the positions of the poles of the total system for a given value of the parameter K in the controller.

## 5. General architecture of the system

Figure 5 shows the architecture and building blocks of the software that has been developed. The first stage is the construction of the ontology, that defines the concepts and knowledge used to implement the design of lead-lag controllers with the root locus method. This stage needs the collaboration of a knowledge engineer and a control engineer. Once the ontology is built, a software module translates the structures in the ontology into production rules that execute the described design procedure.

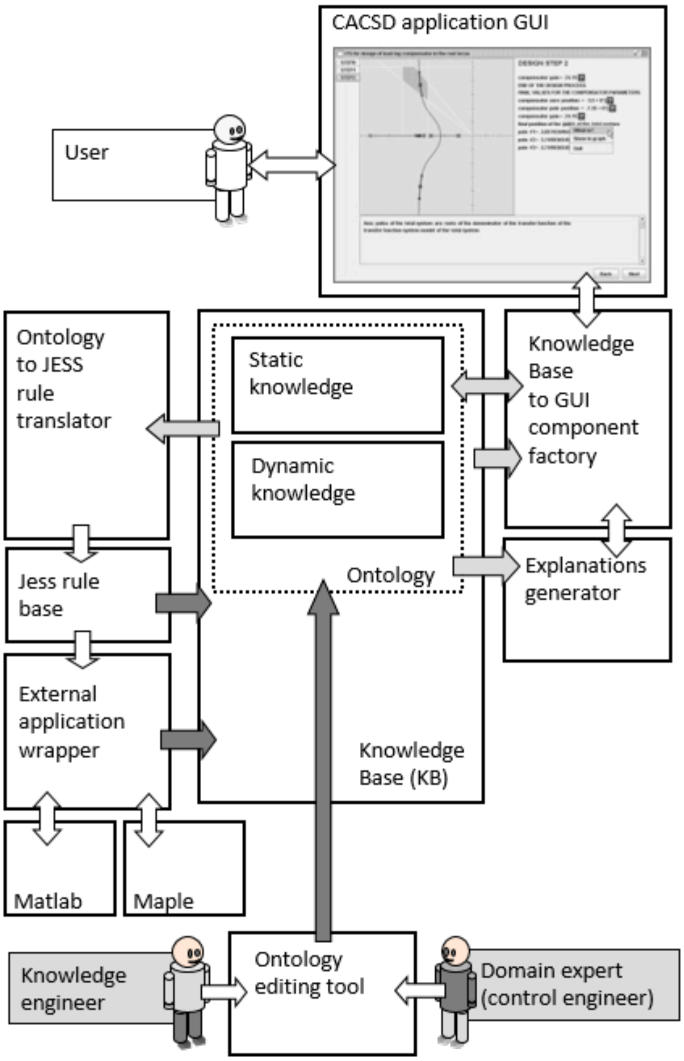


Figure 5. Architecture and building blocks

With the ontology and the corresponding rules, the system is ready to be entered some design problem data. With these particular data the rules will be fired until a solution is found. Every single calculation and design decision will be stored along with the ontology forming a knowledge base Then the user can interact with this solved problem by using a graphical user interface where all the information will be presented. By using this interface, the user is guided through the design process, and the knowledge model serves as a source of explanations for any question the user may pose.

Each of the different modules and subsystems are presented and described in the following sections.

## 6. Conceptual structures in the ontology for root-locus controller design

This section contains the description of the structure of the ontology describing the controller design process by the root locus method. There are many ontology building methodologies [50], most of them based on well-known software engineering methodologies. In this work, METHONTOLOGY [51] was chosen as a guiding methodology, and a bottom-up strategy was used to build the conceptual model, this means that the bottom level concepts are conceptualized first, while the top level (more complex) ones are built based on the concepts already existing in the model. This decision was taken on the basis of a study about the characterization of the knowledge of the domain: As pointed out in [52] one of the main characteristics of control engineering is the way it emerged as a differentiated discipline from other previously existing ones (like electronics or communication engineering) by creating and developing a language of its own. The concepts of this language are built on top of other, lower level ones, starting with those from mathematics, and, therefore, bottom-up strategy seems to be adequate.

Before presenting the ontology, it is useful to point out that two conceptually different types of knowledge to be modelled can be found: the one dealing with the static knowledge and the one dealing with the dynamic knowledge. The first one describes and defines static concepts, properties, characteristics, components, etc; while the second describes actions, tasks, design decisions, iterations, etc. This distinction is useful both for building and understanding the structure of the ontology. The conceptualization in the ontology of both types of knowledge is described next.

### *6.1. Conceptualization of the static knowledge*

Control engineering analysis and design methods are based on the use of mathematical models describing the dynamical behaviour of the systems. For this reason, it is reasonable to start the conceptualization at the level of mathematical structures. The ontology, in fact, is grounded on the conceptualization of the mathematical concept "real number".

On top of the concept "real number", other concepts are represented: complex number (consisting of a pair of real numbers), polynomial (consisting of a series of coefficients or a series of roots plus a main coefficient), quotient (consisting of a numerator and a denominator), polynomial quotient (a specialization of a quotient where both numerator and denominator are polynomials), and so on. Each transfer function representing the involved systems is described by one of these polynomial quotients.

Figure 6 shows four different conceptual levels, where the concepts belonging to each of the levels are defined by using and combining the concepts from the levels below, and so the higher the level, the higher the abstraction of the concept. The basic mathematical concepts belong to “Level 1”. In this level, classes (dotted squares) represent mathematical concepts with their corresponding attributes (labelled arrows). Level 2 contains instances (rounded rectangles) that are used to give a suitable name to a mathematical concept from level 1. For example, the polynomial appearing in the “hasDenominator” attribute of a PolynomialQuotient element is named the “denominator” of the given quotient. This is important in order to later build mathematical expressions by using these names instead of the name of the attribute. Level 3 shows an instance “poles”, that is defined by concatenating the names of the concepts created in level 2. This way, the “poles” of a given TransferFunction are defined as the “roots” of the “denominator” of the “transfer function” (the dotted arrow shows this ordered concatenation). This way, the control engineer uses the names of the mathematical concepts he is used to handle while, internally, this expression is translated into a property (attribute) chaining in the ontology descriptions. In a fourth level the instance concept “realPoles” is built by stablishing a condition on the concept in the level below. In this example, “realPoles” are those “poles” (complex numbers) having its imaginary part equal to zero. With all this set of concepts, the control engineer has a suitable conceptual toolkit to ease the knowledge acquisition phase when describing the design process of a compensator. He can refer to “the real poles of a given system” without the need to worry about the internal structure of the ontology because the definition for “real poles” is conveniently represented, as has been seen.

Level 1 contains raw data structures, while level 2 contains mathematical concepts. Finally, levels 3 and 4 contain control engineering concepts. This stratified structure reflects the fact that control engineering concepts are built on top of other ones, and allows the generation of definitions and explanations both for the control engineer describing the design of compensators and for the final user of the application.

### *6.1.1. Mathematical concepts*

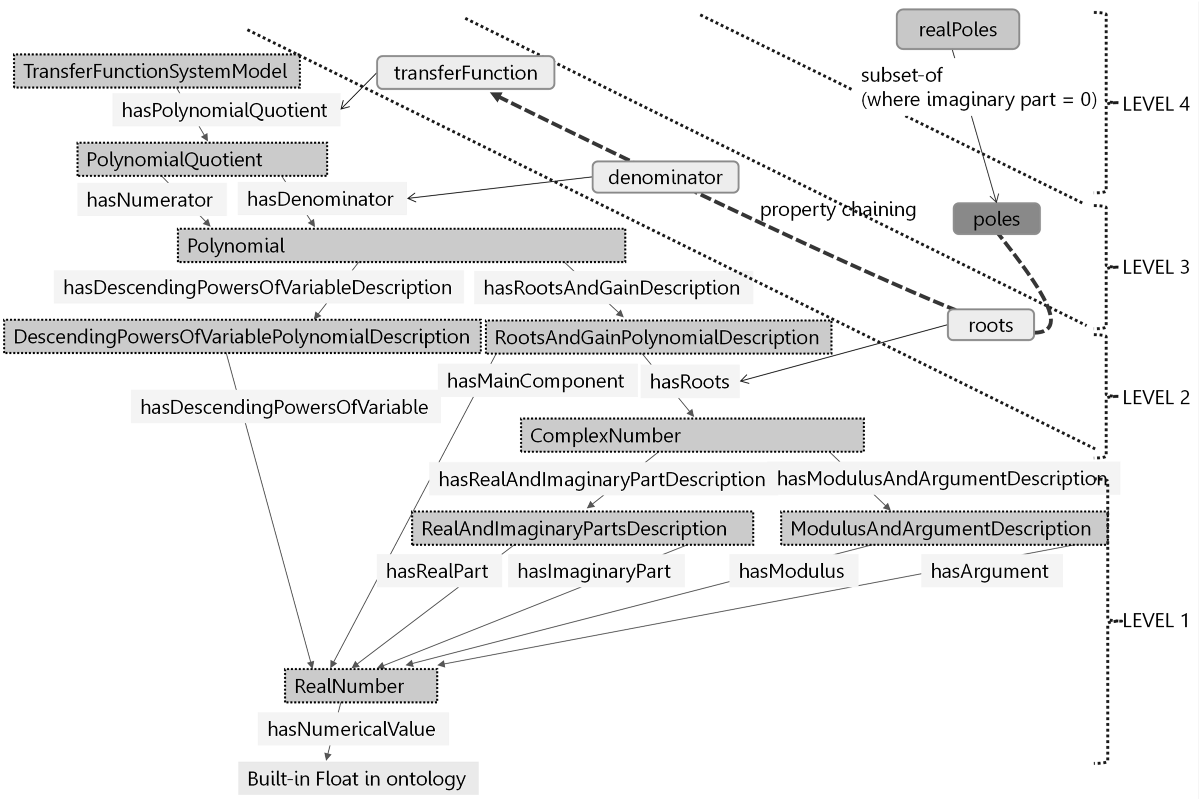

realPoles
TransferFunctionSystemModel
transferFunction
subset-of
(where imaginary part = 0)
LEVEL 4
hasPolynomialQuotient
PolynomialQuotient
denominator
hasNumerator
hasDenominator
property chaining
poles
Polynomial
LEVEL 3
hasDescendingPowersOfVariableDescription
hasRootsAndGainDescription
DescendingPowersOfVariablePolynomialDescription
RootsAndGainPolynomialDescription
roots
hasMainComponent
hasRoots
LEVEL 2
hasDescendingPowersOfVariable
ComplexNumber
hasRealAndImaginaryPartDescription
hasModulusAndArgumentDescription
RealAndImaginaryPartsDescription
ModulusAndArgumentDescription
hasRealPart
hasImaginaryPart
hasModulus
hasArgument
LEVEL 1
RealNumber
hasNumericalValue
Built-in Float in ontology

Figure 6. Basic and derived concepts for the mathematical structures representing the models of the dynamical systems.

Characteristics, also called properties, are a very important ontological construct. As an example we have: *real part*, *imaginary part*, *modulus* and *argument* as characteristics of complex numbers; *order*, *roots* or *main coefficient*, etc, for polynomials, and so on. All these properties are part of the definition of the concepts or can be calculated by some mathematical expression ("mathematical expression" is also a concept in the ontology). Both quantitative and qualitative characteristics can be found. Quantitative characteristics can be calculated by a formula or take their value from external applications in case that there is no direct mathematical expression to be applied for its calculation. In this later case, the interaction between the ontology and the external applications (Matlab and Maple have been used in this study) is also conceptualized in the ontology (representing the functions to be called and the parameters and their corresponding data types to be passed to these functions). Qualitative characteristics are also defined, usually by establishing different intervals over quantitative conditions.

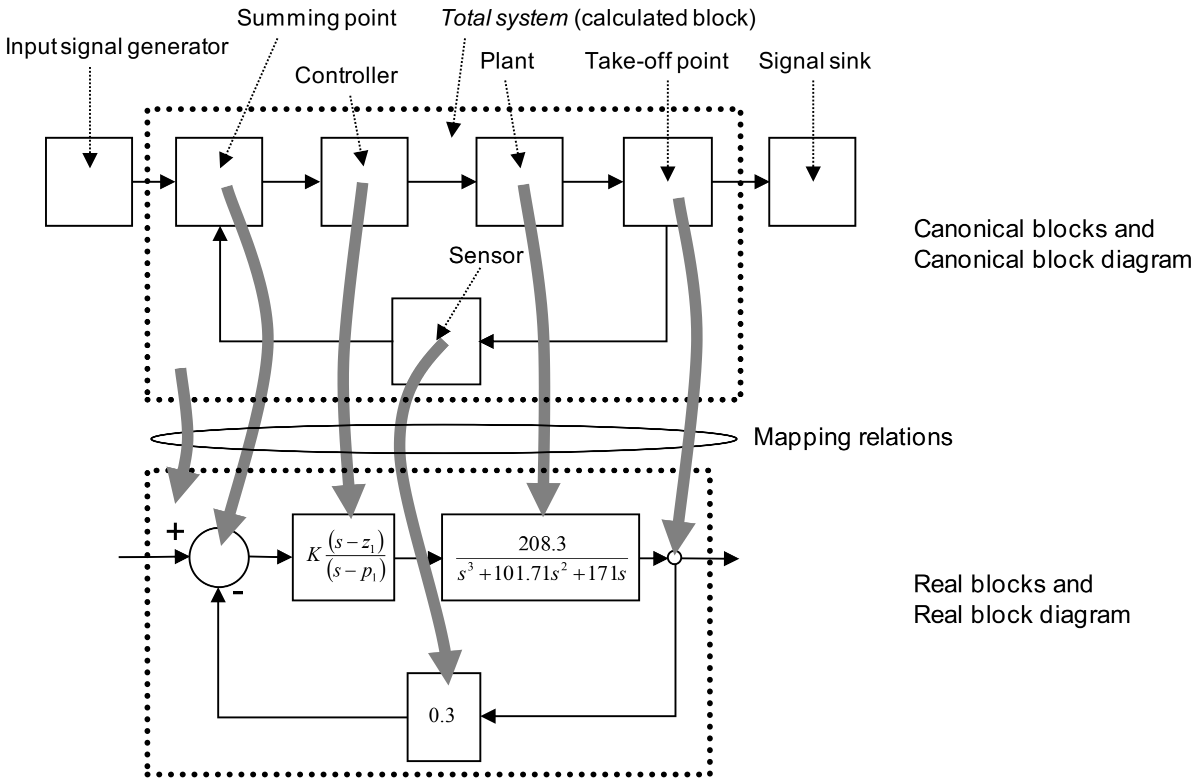


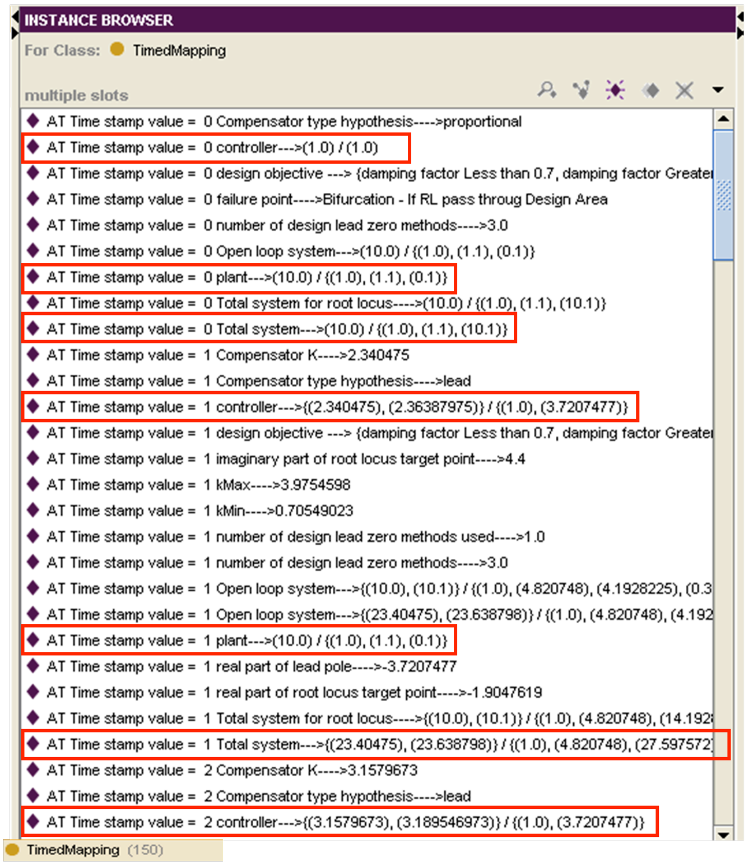
Figure 7. Canonical blocks and mapping to real blocks

### *6.1.2. Subsystem interconnection*

The interconnection of transfer functions is represented in the ontology by means of a conceptualization of the block diagram, as shown in figure 3. Blocks are named according to their function in the diagram. Connections among blocks are also conceptualized, in order to being able to define calculated blocks, as is the case of the total system in figure 7. The blocks represented in the top diagram in figure 7 are called "canonical blocks" because they represent the systems involved independently of the actual different physical elements that may play these roles.

The name of the canonical blocks is useful to be able to refer to them in order to stablish the rationale of the design process in an abstract way. The design process will be described by referring to the characteristics of the canonical blocks ("the Plant", "the Controller", "the Total system", etc.) in general. Later, when this design process is applied to a real problem, the canonical blocks will be mapped to the real mathematical models in order to accomplish the design. Moreover, within a given design process, some canonical blocks, as is the case of "the Controller", will have different mappings in the different design steps because the parameters defining it will change as the design evolves.

Canonical blocks are represented in the ontology as instances of a class where the main property is the name of the canonical block, because it is the way they will be referred. The mappings consist on a reference to the canonical block and a reference to a given instance of the mapped object (in this case, an instance of TransferFunctionSystemModel). Timed mappings also have account of the design phase (represented with an integer reflecting the design step).



```
AT Time stamp value =  0 Compensator type hypothesis---->proportional
AT Time stamp value =  0 controller--->(1.0) / (1.0)
AT Time stamp value =  0 design objective ---> {damping factor Less than 0.7, damping factor Greater
AT Time stamp value =  0 failure point---->Bifurcation - If RL pass throug Design Area
AT Time stamp value =  0 number of design lead zero methods---->3.0
AT Time stamp value =  0 Open loop system--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  0 plant--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  0 Total system for root locus---->(10.0) / {(1.0), (1.1), (10.1)}
AT Time stamp value =  0 Total system--->(10.0) / {(1.0), (1.1), (10.1)}
AT Time stamp value =  1 Compensator K---->2.340475
AT Time stamp value =  1 Compensator type hypothesis---->lead
AT Time stamp value =  1 controller--->{(2.340475), (2.36387975)} / {(1.0), (3.7207477)}
AT Time stamp value =  1 design objective ---> {damping factor Less than 0.7, damping factor Greater
AT Time stamp value =  1 imaginary part of root locus target point---->4.4
AT Time stamp value =  1 kMax---->3.9754598
AT Time stamp value =  1 kMin---->0.70549023
AT Time stamp value =  1 number of design lead zero methods used---->1.0
AT Time stamp value =  1 number of design lead zero methods---->3.0
AT Time stamp value =  1 Open loop system--->{(10.0), (10.1)} / {(1.0), (4.820748), (4.1928225), (0.3
AT Time stamp value =  1 Open loop system--->{(23.40475), (23.638798)} / {(1.0), (4.820748), (4.192
AT Time stamp value =  1 plant--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  1 real part of lead pole---->-3.7207477
AT Time stamp value =  1 real part of root locus target point---->-1.9047619
AT Time stamp value =  1 Total system for root locus---->{(10.0), (10.1)} / {(1.0), (4.820748), (14.192
AT Time stamp value =  1 Total system--->{(23.40475), (23.638798)} / {(1.0), (4.820748), (27.597572
AT Time stamp value =  2 Compensator K---->3.1579673
AT Time stamp value =  2 Compensator type hypothesis---->lead
AT Time stamp value =  2 controller--->{(3.1579673), (3.189546973)} / {(1.0), (3.7207477)}
```

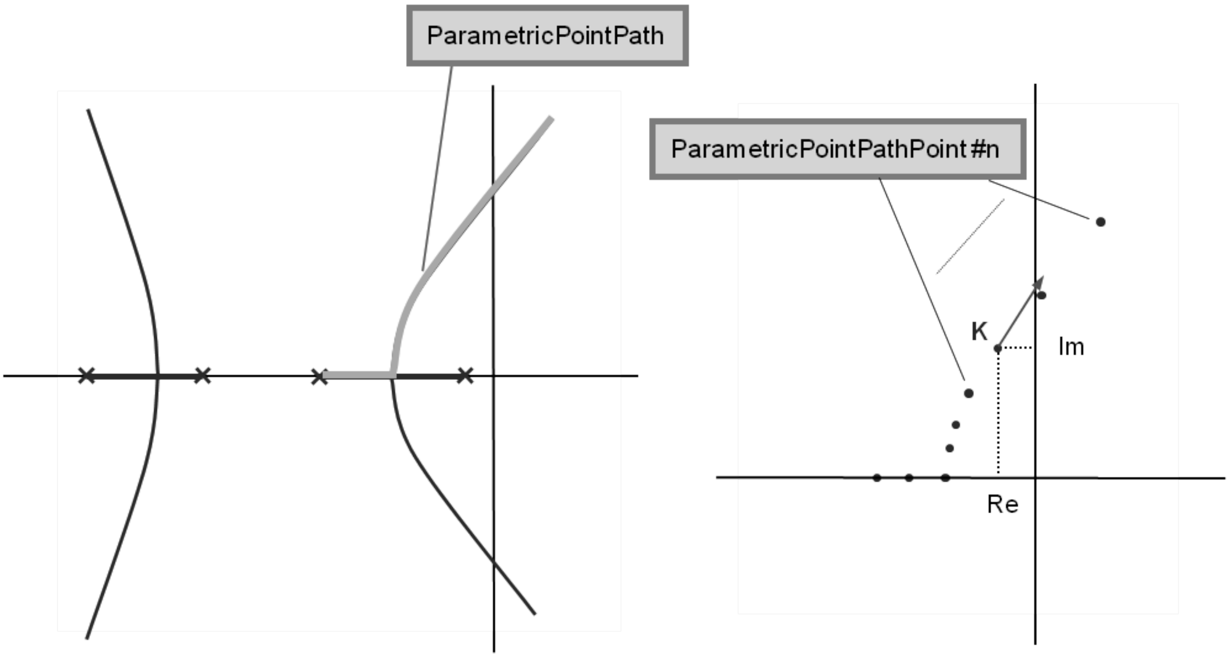
Figure 8. Canonical blocks with timed mappings in the ontology.

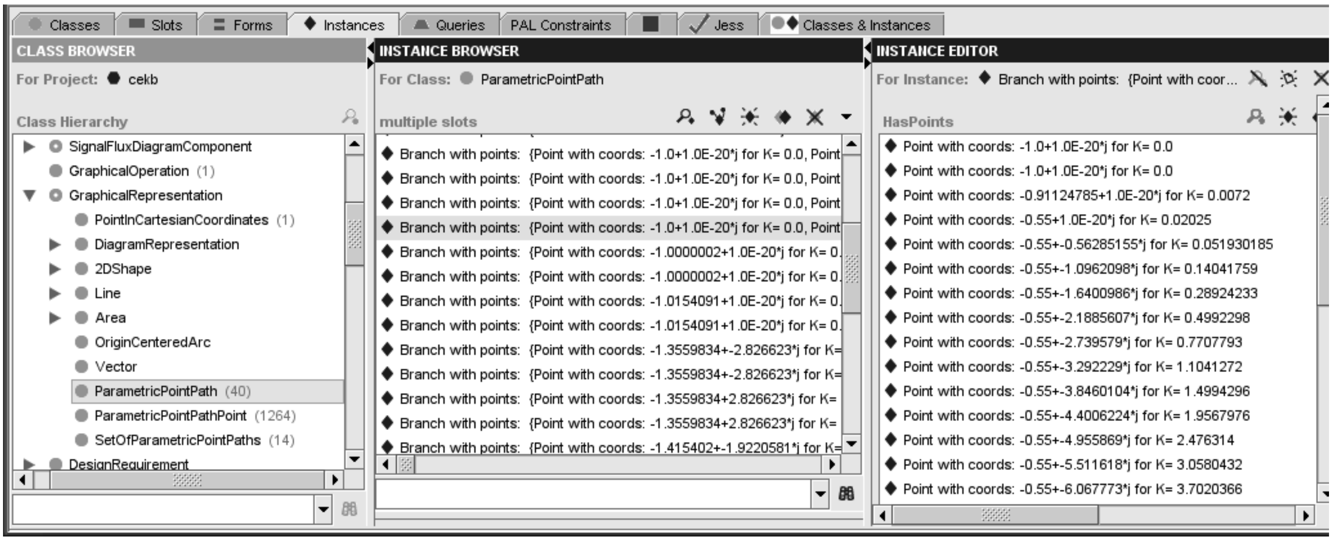
Figure 8 shows some instances of the concept TimedMapping. There are mappings referring to the design steps number 0, 1 and 2. Mappings related to the canonical blocks are highlighted with a rectangle. For each mapping there is a reference to the design step (the integer name representing the step is called a “time stamp”), then it appears the name of the canonical block, and finally the actual transfer function is associated. In the figure, the canonical blocks that appear are “plant”, “Total system” and “controller”. As can be seen, the plant is always mapped to the same transfer function, because the plant is a problem data and does not change during the design. The controller is the element being designed and so each design stage has a different value. The total system is also different for each time stamp because its transfer function depends on the one of the controller.

### *6.1.3. Conceptualization of the root locus graph*

Figure 9 shows some examples of root locus graphical objects and their corresponding ontology representation. The root locus is the path that each of the poles of the total system travel when the parameter K of the compensator varies from zero to infinity. Each pole, then, describes a parametric curve (usually called “a branch” of the root locus). In the ontology, each of these branches is called a “ParametricPointPath”, and the points that build them are represented by concepts called “ParametricPointPathPoint”.

The root locus is an important structure to be modeled. All the graphical elements constituting the locus must be included in the conceptualization, and therefore points (with their real and imaginary part, value of the parameter K, partial derivative at the point, etc), branches (composed of a series of points), etc. must be represented. Derived concepts are also needed like "stable portion of a branch" or "stable portion of the root locus" (defined as the interval of K values where all the corresponding branches are in the left-hand-side plane).





CLASS BROWSER

For Project: cekb

Class Hierarchy

- SignalFluxDiagramComponent
- GraphicalOperation (1)
- GraphicalRepresentation
  - PointInCartesianCoordinates (1)
  - DiagramRepresentation
  - 2DShape
  - Line
  - Area
  - OriginCenteredArc
  - Vector
  - ParametricPointPath (40)
  - ParametricPointPathPoint (1264)
  - SetOfParametricPointPaths (14)
- DesignRequirement

INSTANCE BROWSER

For Class: ParametricPointPath

multiple slots

```
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0000002+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0000002+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0154091+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0154091+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.3559834+-2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+-2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+2.826623*j for K=
Branch with points:  {Point with coords: -1.415402+-1.9220581*j for K=
```

INSTANCE EDITOR

For Instance: Branch with points: {Point with coor...

HasPoints

```
Point with coords: -1.0+1.0E-20*j for K= 0.0
Point with coords: -1.0+1.0E-20*j for K= 0.0
Point with coords: -0.91124785+1.0E-20*j for K= 0.0072
Point with coords: -0.55+1.0E-20*j for K= 0.02025
Point with coords: -0.55+-0.56285155*j for K= 0.051930185
Point with coords: -0.55+-1.0962098*j for K= 0.14041759
Point with coords: -0.55+-1.6400986*j for K= 0.28924233
Point with coords: -0.55+-2.1885607*j for K= 0.4992298
Point with coords: -0.55+-2.739579*j for K= 0.7707793
Point with coords: -0.55+-3.292229*j for K= 1.1041272
Point with coords: -0.55+-3.8460104*j for K= 1.4994296
Point with coords: -0.55+-4.4006224*j for K= 1.9567976
Point with coords: -0.55+-4.955869*j for K= 2.476314
Point with coords: -0.55+-5.511618*j for K= 3.0580432
Point with coords: -0.55+-6.067773*j for K= 3.7020366
```

Figure 9. Representation of the root locus

### *6.2. Conceptualization of the dynamic knowledge*

The process of compensator design in control engineering is iterative in nature, that is, different compensators can be designed until a given one makes the total system accomplish the original performance objectives.

In this research, a typical configuration is assumed with a lead-lag compensator in cascade with the plant, unitary and negative feedback and designing for tracking of the reference signal. The design parameters, under these assumptions, are the type of controller (proportional, lead or lag), the positions of poles and zeroes of the controller ($p_1$ and $z_1$ in figure 3), and its proportional constant $K$.

The design process is conceptualized by dividing it into a number of tasks and subtasks. Each of these tasks is represented by a rule-like structure where the following components can be distinguished:

- An antecedent, consisting of a series of objects that must be present in the knowledge base and a number of conditions that must be accomplished in order to perform the task.

- The consequent, consisting on a number of actions to be executed when the task is performed and the next task to be executed (if any) when the present one ends.

Rules are stated by using the constructs available in the ontology and, consequently, there is no need to know a particular rule language. The ontology, in this regard, acts as convenient knowledge acquisition tool, letting the control engineer focus in the design method and not in the implementation.

Actions, eventually, can contain design decisions, that are statements assigning a given value to one of the design parameters (for example, “the real part of the lead zero”). The dynamic knowledge model is modular, allowing the use and interchange of different strategies for a single design task, easing the maintainability and extensibility of the ontology, and the use of different design strategies and algorithms.

Almost all of the sentences describing the design process are referred, in the end, to a given transfer function system model of the ones participating in the design (that is: the plant, the controller, the "open loop system" and the total system, see figure 3). But these references must be stated independently of any particular value that the systems may have in a given design iteration. This makes it necessary to build a mapping (for each of the design stages) from the names of the participating blocks to the corresponding real transfer functions, as represented in figure 7 and 8.

Time is not treated as a continuous measure in the design process, but as a discrete series of intervals where the design is performed. During one of these design intervals, all the necessary design parameters are given a value and so the design interval will end by testing the current design against the original performance specifications. The ontology has a conceptualization of the current design stage and also a set of operators in order to refer to the "previous", "previous of previous", etc., design intervals. This way, any reference to any point of the design process can be easily stated in relation to the current interval. For simplicity, when an expression makes reference to "the type of the controller" or to "the real part of the lead zero", it is assumed that this means "the type of the current controller" and "the real part of the current lead zero". Whenever a mention is made to a previous design, the time stamp reference is calculated to tackle with this issue.

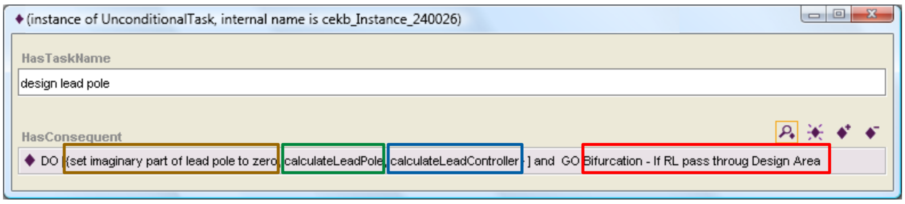
With all these considerations, a design decision about a redesign activity could be stated as "set the real part of the lead zero equal to 0.75 times its previous value", as use to be the case in real scenarios.

Once the ontology reflecting the design process is built, the conceptual structures must be "executed" against some particular problem data. In order to accomplish this, the conceptual structures in the ontology are first automatically translated into a series of production rules.

#### *6.2.1. Some examples about the conceptualization of the dynamic knowledge*

Some examples of dynamical knowledge structures and their corresponding translations into rules are given next in order to clarify this type of conceptualization.

Figure 10 shows an example of design conceptualization for a given sub-design problem: assigning a value to the lead pole of the compensator. This is one of the simplest tasks that can be found in the ontology. It is a task with no preconditions (is called an “UnconditionalTask”). It has only a consequent where a “DO” clause and a “GO” clause are stated. The “DO” clause contains a set of actions to be performed while the “GO” clause states the next task to be achieved. Below the ontology snapshot, the corresponding (automatically generated) translation to a production rule is also shown.



(instance of UnconditionalTask, internal name is cekb_Instance_240026)

HasTaskName

design lead pole

HasConsequent

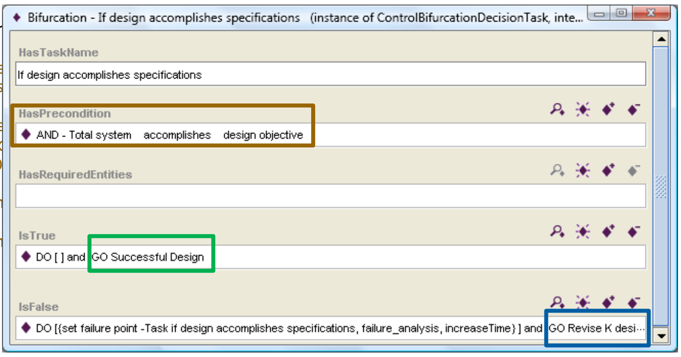
DO {set imaginary part of lead pole to zero, calculateLeadPole, calculateLeadController } and GO Bifurcation - If RL pass throug Design Area

```
      (defrule UnconditionalTask-design_lead_pole
      (object (is-a NamedTimeStamp)(hasTimeStampName  "CURRENT")(hasTimeStamp  ?currentTimeStamp))
      ?idActivator <- (design lead pole)
      =>
      (removeDesignParameterTimedMapping  cekb_Instance_170010  ?currentTimeStamp  )
      (bind ?mapping00  (make-instance  of  DesignParameterMapping
      (hasDesignParameter  cekb_Instance_170010)
      (hasMappedDesignParameterValue   cekb_Instance_12)
       map))
DO    (make-instance  of  TimedMapping(hasMapping  ?mapping00)
      (hasDesignTimeStamp   ?currentTimeStamp)
       map)
      (calculateLeadPole          (create$ cekb_Instance_170009  cekb_Instance_560006
      cekb_Instance_100011  cekb_Instance_150002  cekb_Instance_560001  )   )
      (calculateLeadController    (create$ )   )
GO    (assert  (If  RL  pass  throug  Design  Area))
      (retract ?idActivator)
      )
```

Figure 10. Representation of a simple task for assigning the lead pole.

Another example of conceptualized task is the one that tests if the compensator design achieved within a design stage leads to the fulfillment of the initial design requirements. This conceptual structure, and its corresponding generated rules can be seen in figure 11. In this case, a different set of actions are stated depending whether the design is successful (the execution ends successfully) or not (a complete re-design process must be achieved, trying first to choose a different value for the parameter K of the compensator). In this case, the translation into rules results in two different ones: one for the case of a successful design, and one for the case of an unsuccessful design.

```
(defrule ControlBifurcationDecisionTask-If_design_accomplishes_specifications-TRUE
?idActivator <- (If design accomplishes  specifications)
(object (is-a NamedTimeStamp)(hasTimeStampName  "CURRENT")(h
;The precondition  patterns:
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTime
(object (is-a CanonicalToRealMapping)(OBJECT  ?mapping0)(has
cekb_Instance_220000))  (hasRealInstance  ?realInstance0))
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTime
(object (is-a DesignConceptMapping)(OBJECT  ?mapping1)  (has(
cekb_Instance_410019))(hasRealDesignConcept  ?realInstance1)
;The accomplishes  function patterns (ts, %OS)
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCh
"settling time 2 percent"))  (hasElementTowhichCharacteristi
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCh
"damping factor"))  (hasElementTowhichCharacteristicApplies
(test (accomplishes ?realInstance0 ?realInstance1 ))
;The required  Entity patterns:
=>
(assert (Successful  Design))
(retract ?idActivator)
)
```



Bifurcation - If design accomplishes specifications   (instance of ControlBifurcationDecisionTask, inte...

HasTaskName

If design accomplishes specifications

HasPrecondition

AND - Total system   accomplishes   design objective

HasRequiredEntities

IsTrue

DO [ ] and GO Successful Design

IsFalse

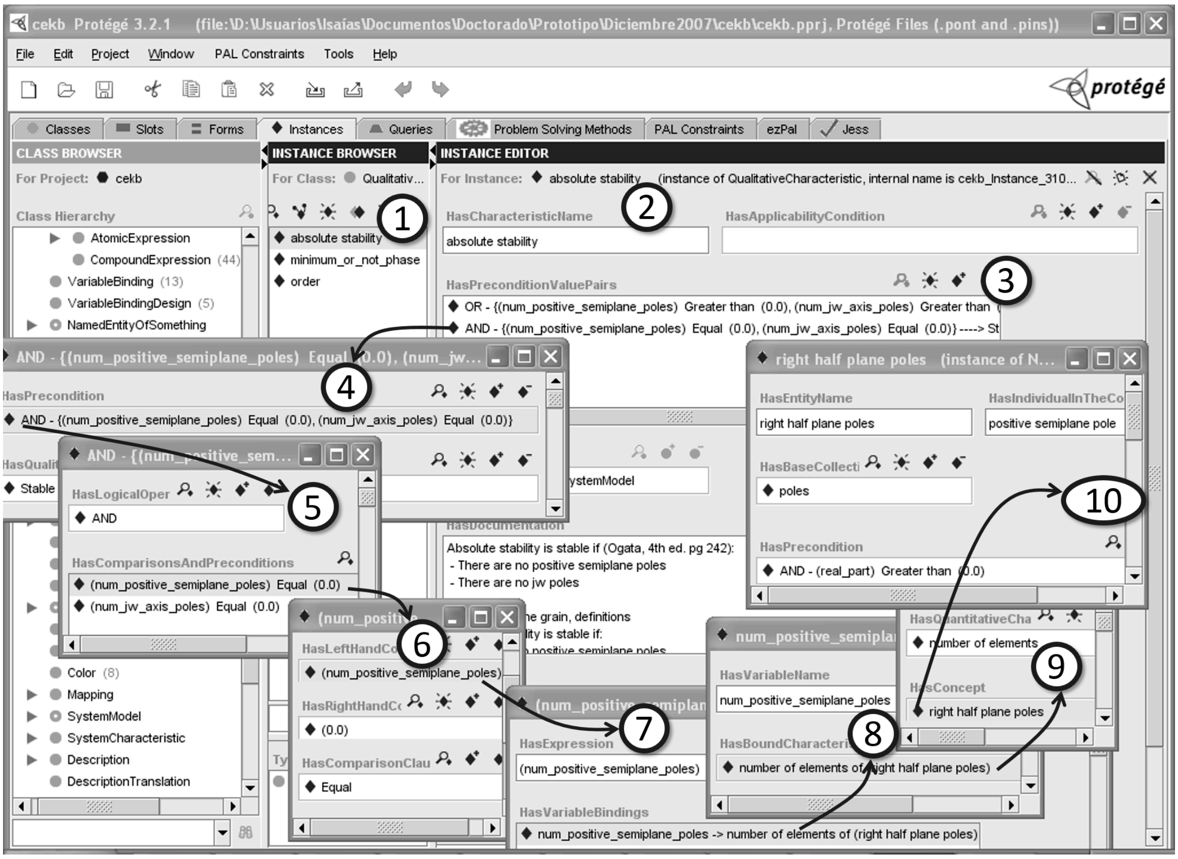
DO [{set failure point -Task if design accomplishes specifications, failure_analysis, increaseTime} ] and GO Revise K desi...

```
(defrule ControlBifurcationDecisionTask-If_design_accomplishes_specifications-FALSE
?idActivator <- (If design accomplishes  specifications)
(object (is-a NamedTimeStamp)(hasTimeStampName  "CURRENT")(hasTimeStamp  ?currentTimeStamp))
;The re;The precondition  patterns:
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTimeStamp)(hasMapping  ?mapping0 ))
(object (is-a CanonicalToRealMapping)(OBJECT  ?mapping0)(hasCanonicalInstance  ?canonical0&:(=  (instance-name  ?canonical0)
cekb_Instance_220000))  (hasRealInstance  ?realInstance0))
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTimeStamp)(hasMapping  ?mapping1 ))
(object (is-a DesignConceptMapping)(OBJECT  ?mapping1)  (hasCanonicalDesignConcept  ?canonical1&:(=  (instance-name  ?canonical1)
cekb_Instance_410019))(hasRealDesignConcept  ?realInstance1))
;The accomplishes  function patterns (ts, %OS)
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCharacteristic  ?charac1&:(eq  (slot-get  ?charac1  hasCharacteristicName)
"settling time 2 percent"))  (hasElementTowhichCharacteristicApplies  ?realInstance0)  )
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCharacteristic  ?charac2&:(eq  (slot-get  ?charac2  hasCharacteristicName)
"damping factor"))  (hasElementTowhichCharacteristicApplies  ?realInstance0)  )
(not(test (accomplishes ?realInstance0 ?realInstance1 )))
;The required  Entity patterns:
                                                              =>
.......DO ACTIONS………..
(assert (Revise  K design))
(retract ?idActivator)
)
```

Figure 11. Representation of the task testing if the current design is successful or not.

### *6.3. Some examples and metrics from the ontology*

The next two paragraphs show two examples of conceptualizations in the ontology. The objective is to show the high modularity of concepts that give a great level of flexibility for building the conceptual structures and easing the knowledge acquisition phase. The first one corresponds to the so-called static structure of the ontology and the second one to its dynamic structure.


cekb Protégé 3.2.1 (file:\D:\Usuarios\Isaías\Documentos\Doctorado\Prototipo\Diciembre2007\cekb\cekb.pprj, Protégé Files (.pont and .pins))
File Edit Project Window PAL Constraints Tools Help
protégé
Classes Slots Forms Instances Queries Problem Solving Methods PAL Constraints ezPal Jess
CLASS BROWSER
INSTANCE BROWSER
INSTANCE EDITOR
For Project: cekb
For Class: Qualitativ...
Class Hierarchy
AtomicExpression
CompoundExpression (44)
VariableBinding (13)
VariableBindingDesign (5)
NamedEntityOfSomething
Color (8)
Mapping
SystemModel
SystemCharacteristic
Description
DescriptionTranslation
1
2
3
4
5
6
7
8
9
10

For Instance: absolute stability (instance of QualitativeCharacteristic, internal name is cekb_Instance_310...

Instances: absolute stability; minimum_or_not_phase; order

HasCharacteristicName: absolute stability

HasApplicabilityCondition:

HasPreconditionValuePairs:
- OR - {(num_positive_semiplane_poles) Greater than (0.0), (num_jw_axis_poles) Greater than
- AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw_axis_poles) Equal (0.0)} ----> St

AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw...

HasPrecondition: AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw_axis_poles) Equal (0.0)}

HasQualit...: Stable

...ystemModel

HasDocumentation:
Absolute stability is stable if (Ogata, 4th ed. pg 242):
- There are no positive semiplane poles
- There are no jw poles
...e grain, definitions
...ity is stable if:
...positive semiplane poles

AND - {(num_positive_sem...

HasLogicalOper: AND

HasComparisonsAndPreconditions:
- (num_positive_semiplane_poles) Equal (0.0)
- (num_jw_axis_poles) Equal (0.0)

(num_positi...

HasLeftHandCo...: (num_positive_semiplane_poles)

HasRightHandC...: (0.0)

HasComparisonClau...: Equal

(num_positive_semiplan...

HasExpression: (num_positive_semiplane_poles)

HasVariableBindings: num_positive_semiplane_poles -> number of elements of (right half plane poles)

num_positive_semipla...

HasVariableName: num_positive_semiplane_poles

HasBoundCharacteri...: number of elements of (right half plane poles)

HasQuantitativeCha...: number of elements

HasConcept: right half plane poles

right half plane poles (instance of N...

HasEntityName: right half plane poles

HasIndividualInTheCo...: positive semiplane pole

HasBaseCollecti...: poles

HasPrecondition: AND - (real_part) Greater than (0.0)

Figure 12. Part of the conceptualization of the concept "absolute stability"

Figure 12 shows a qualitative characteristic of a dynamical system as described in classical control theory: absolute stability. The figure shows the complexity of the definition that is represented in the ontology (smaller windows show some of concepts that are used for the definition of the main one). This representation allows the automatic generation of user friendly definitions of the concept if the user asks for them, and also the automatic calculation of the characteristic value.

The absolute stability of a dynamical system is a qualitative value taking the following values:

- "Stable" – when all the poles of the transfer function model of the system have a negative real part.
- "Not Stable" – when at least one of the poles of the transfer function model of the system have a positive real part or is located in the imaginary axis.

In figure 12, the following structures can be observed:

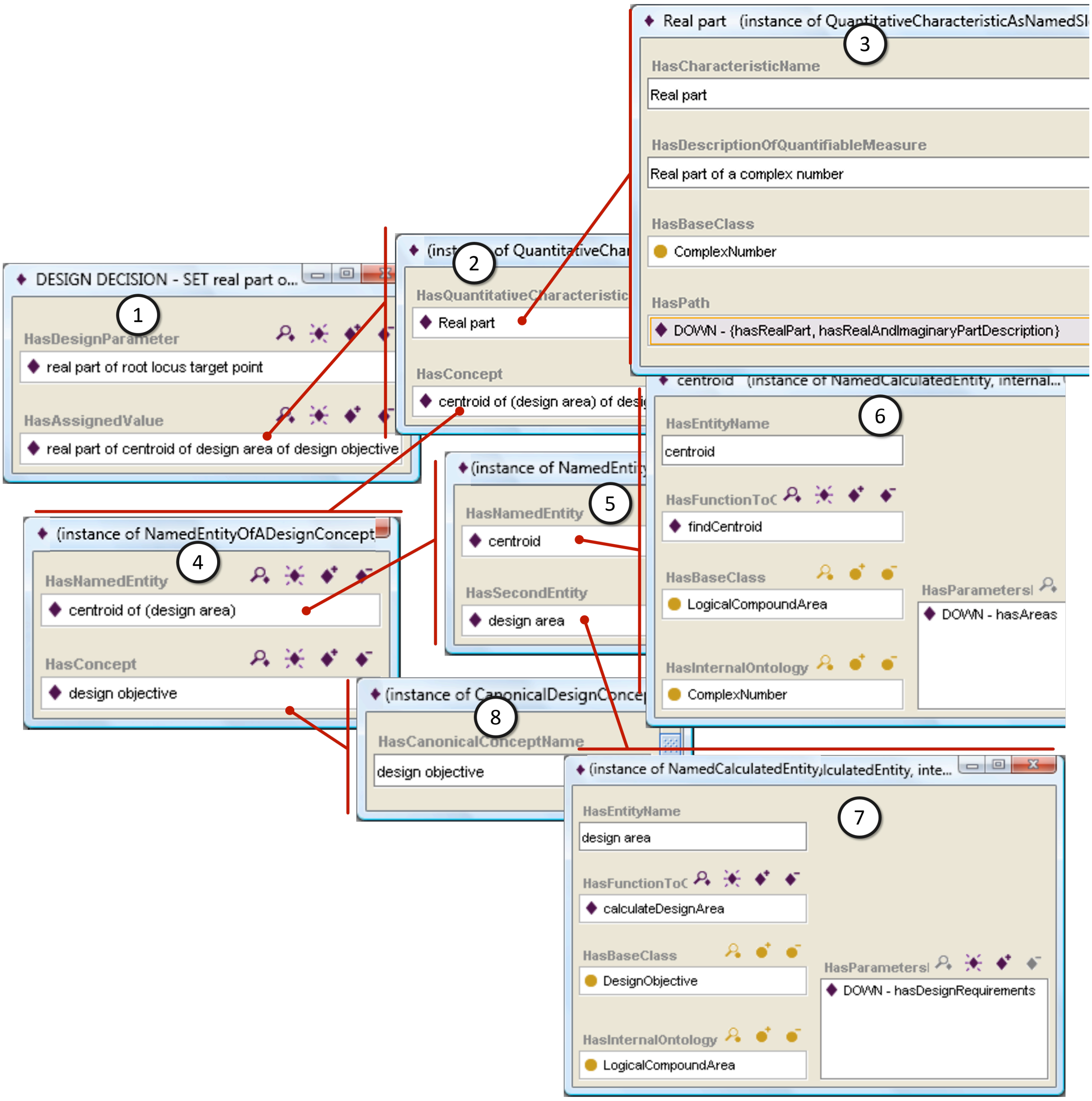
- Number 1 shows the instance defining the absolute stability.
- Number 2 is the window where the definition of this instance is stated.
- Number 3 shows the two conditions corresponding to the "Stable" and "Not Stable" qualitative values, as stated previously.
- Number 4 expands the second of the conditions, showing how the boolean condition is mapped to the qualitative value "Stable".
- Number 5 is the instance where the boolean condition is stated. A logical operator AND is stablished, and the conditions affected are shown below.
- Number 6 shows the instance where the first condition is created. There is a quantitative characteristic ("num_positive_semiplane_poles"), a value ("0.0"), and a comparator ("Equal").
- Number 7 shows how the quantitative value "num_positive_semiplane_poles" is defined. In this case, it corresponds to the expression "number of elements of (right half plane poles)".

- Number 8 shows how the mapping from the expression “number of elements of (right half plane poles) to the value “num_positive_semiplane_poles” (more adequate for handling by humans) is stated in the ontology.
- Number 9 shows how the expression “number of elements of (right half plane poles)” is composed of the operator “number of elements” and the collection of objects “right half plane poles”.
- Number 10 shows how the concept “right half plane poles” is defined. The definition is built by using a condition on the set of poles, testing which of these poles have their real part greater than zero.
- The definition of the concept “poles” could also be shown, but would make the figure unreadable. But it is interesting to refer to figure 6 where this concept appears in the “level 3”, defined as the “roots” of the “denominator” of the “transferFunction”. Also, the concepts from level 2 “roots”, “denominator” and “transferFunction” refer to concepts in level 1 that, in the end, are based on the basic “RealNumber” concept, which is the simplest concept in the ontology.

The second example is the conceptualization of part of a task corresponding to the dynamic knowledge. One of the possible actions when designing a lead/lag compensator consists of locating a point called "the root locus target point" (the point through which the root locus of the total system should ideally pass) in the Argand diagram. Figure 13 shows part of the conceptualization regarding a design decision: the assignation of a value to the abscissa (real part) of this point. As can be seen, every single concept is represented in the ontology and decomposed into simpler concepts down to the mathematical ones. In the figure, some conceptual constructs can be distinguished:

- Number 1 corresponds to the instance representing the design decision, that is the assignation of a value (in this case the real part of the centroid of the design area built taking into account the design objectives) to a parameter (the real part of the root locus target point).
- Number 2 is the knowledge representation for “the real part” of an element, in this case “the centroid of the design area”. The instance has a quantitative characteristic (the real part in this example) that is applied to a complex number (the centroid of the design area).
- Number 3 is the conceptualization of the quantitative characteristic “real part”. As it is shown, it can be applied to instances of the class “ComplexNumber” and is defined as the attribute “hasRealPart” of the instance appearing in the attribute hasRealAndImaginaryPartDescription of the corresponding instance of the class ComplexNumber. As can be seen, this definition is based in a property (attribute) chaining. Figure 6 shows how this attributes and classes are related.
- Number 4 shows the concept “centroid of design area” applied to “design objective”, that is, it refers to the centroid of an area that results from the graphical representation of the design objectives (requirements) for the controlled system when the problem is stated.
- Number 5 is the concept representing the centroid of a design area. In number 6, the centroid calculation is conceptualized, in this case by calling an external function (“findCentroid”) to which the parameter obtained from the attribute “hasAreas” is passed. The returning value will have to be parsed and stored as a “ComplexNumber” ontology concept. The instance describing the function “findCentroid” is not included here. It is a call to a Maple function that calculates the centroid of a given 2-D area. Number 6 is the instance representing the calculation of the design area by passing the design requirements to a given function. As a result, a LogicalCompoundArea instance is obtained and stored in the ontology.
- Number 8 is a concept representing the “design objective”. In a similar way as with the canonical blocks, the “design objective” is mapped to the particular design requirements stablished when the problem is stated (see figure 8). In this case, the “design objective” is always mapped to the same set of requirements, because these are initial and fixed desired performance criteria for the controlled system.

With the help of this whole conceptual structure, the user can ask, at any time, for the definition of a concept, for the description of a task, for the function to be called and the parameters to be passed, etc.



DESIGN DECISION - SET real part o...
HasDesignParameter
real part of root locus target point
HasAssignedValue
real part of centroid of design area of design objective

(inst... of QuantitativeChar
HasQuantitativeCharacteristic
Real part
HasConcept
centroid of (design area) of desi

Real part (instance of QuantitativeCharacteristicAsNamedSI
HasCharacteristicName
Real part
HasDescriptionOfQuantifiableMeasure
Real part of a complex number
HasBaseClass
ComplexNumber
HasPath
DOWN - {hasRealPart, hasRealAndImaginaryPartDescription}

(instance of NamedEntityOfADesignConcept
HasNamedEntity
centroid of (design area)
HasConcept
design objective

(instance of NamedEnti
HasNamedEntity
centroid
HasSecondEntity
design area

centroid (instance of NamedCalculatedEntity, internal...
HasEntityName
centroid
HasFunctionToC
findCentroid
HasBaseClass
LogicalCompoundArea
HasInternalOntology
ComplexNumber
HasParametersI
DOWN - hasAreas

(instance of CanonicalDesignConce
HasCanonicalConceptName
design objective

(instance of NamedCalculatedEntity,lculatedEntity, inte...
HasEntityName
design area
HasFunctionToC
calculateDesignArea
HasBaseClass
DesignObjective
HasInternalOntology
LogicalCompoundArea
HasParametersI
DOWN - hasDesignRequirements

Figure 13. Part of the conceptualization of a design decision

The ontology has been built by using a frame-like formalism with the tool Protégé [53], resulting in more than a thousand concepts. The knowledge structures are automatically translated into more than a hundred rules with JESS [54] syntax for executing the dynamic knowledge. The result of the execution of the ontology structures for a given problem is a very rich knowledge base describing the design process, with thousands of new instances introduced as the result of executing an average design activity. This knowledge base is the core and origin of the data for displaying and implementing interaction with the user in the graphical application, which is going to be described next.

### *6.4. Evaluation of the knowledge model*

The development of the ontology was carried out using the knowledge from experts in the field and a number of textbooks. Each of the modelled design procedures was tested with particular problem data sets in order to evaluate the validity of the results. Each set of problem data was carefully chosen in order to force the system to make use of all the different conceptual structures represented in the ontology. Each of the results obtained with the tool were compared to the ones got by manual calculation, with the aid of the "SISO tool" from Matlab (now called Control System Designer App). In all the cases, choosing a similar design point for the controlled system in the Argand diagram, the results obtained were similar.

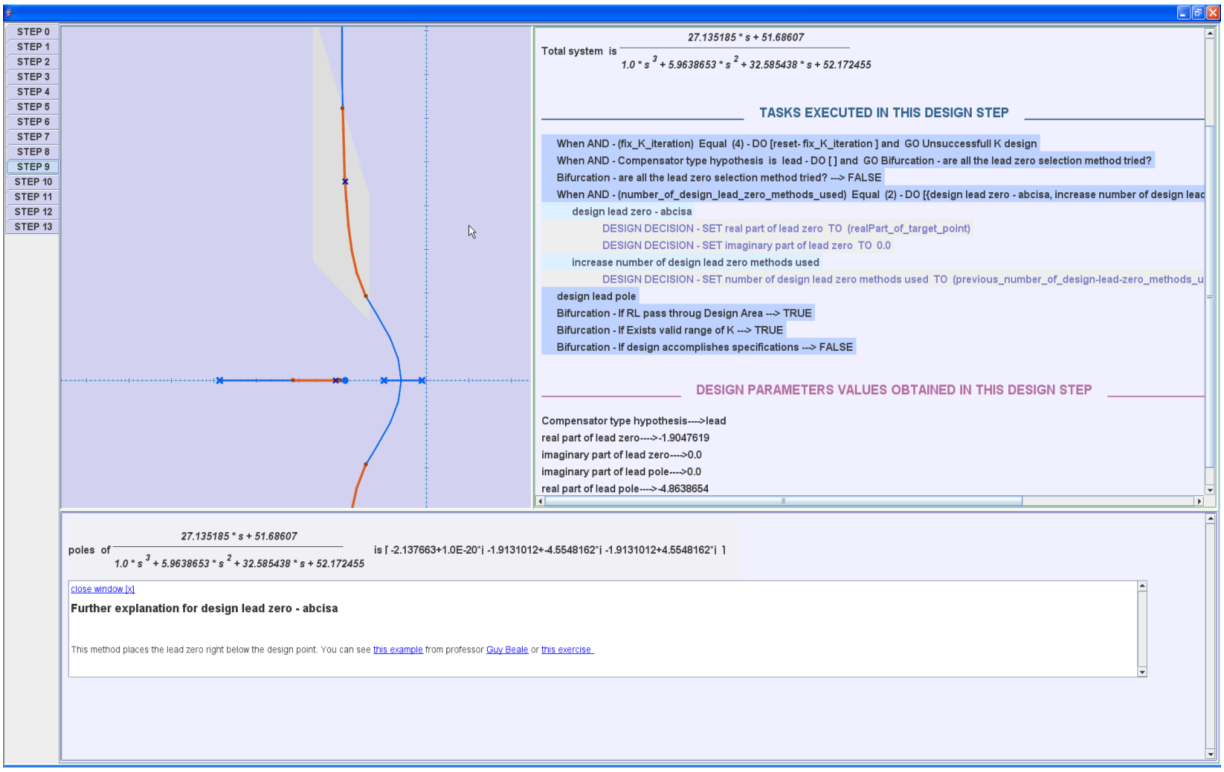
## 7. The application for data presentation and user interaction

Three different levels can be distinguished in the architecture of the system (see figure 5). The first one is the knowledge acquisition level, based on the use of the concepts in the ontology for creating the design procedures. The second one would be an automatic mechanism for transforming the conceptual structures into production rules in order to run the previously defined procedures. The third one would be the rule base once all the rules have been generated.

When some problem data is inserted into the system, the rules will fire until a solution is reached. All the intermediate steps, along with the data that is generated in this process, are stored in a knowledge base. There is a huge amount of data generated for each execution that will be later used to show the process, answer questions and give explanations to the user.

The final result is a knowledge model with a level of description that has not been previously reached for this kind of applications. This knowledge model represents the design process including the design rationale (telling why some tasks were performed but also why some others were not) and also the redesign activities (that is, new designs performed on the basis of previous, unsuccessful results).

Once a given problem has been solved by the system, a user can use the graphical user interface (GUI) of the application in order to study how the problem was solved and the reasons behind each of the design decisions. With this interface, the user can interact with the elements in the screen, posing questions about concepts or design procedures and tasks. Figure 14 shows the graphical application showing the data corresponding to the execution of a given design process. As can be seen, there is a tab for each design iteration (labeled STEPn). Each of these labels has three different windows. The top left shows the graphical elements of the root locus. The top right shows the values of the different transfer functions, the tasks performed during that design iteration and the values of the design parameters. The bottom panel is where the answers to the questions of the user are displayed.



Total system is $\dfrac{27.135185 * s + 51.68607}{1.0 * s^{3} + 5.9638653 * s^{2} + 32.585438 * s + 52.172455}$

TASKS EXECUTED IN THIS DESIGN STEP

When AND - (fix_K_iteration) Equal (4) - DO [reset- fix_K_iteration ] and GO Unsuccessfull K design
When AND - Compensator type hypothesis is lead - DO [ ] and GO Bifurcation - are all the lead zero selection method tried?
Bifurcation - are all the lead zero selection method tried? ---> FALSE
When AND - (number_of_design_lead_zero_methods_used) Equal (2) - DO [{design lead zero - abcisa, increase number of design leac
design lead zero - abcisa
DESIGN DECISION - SET real part of lead zero TO (realPart_of_target_point)
DESIGN DECISION - SET imaginary part of lead zero TO 0.0
increase number of design lead zero methods used
DESIGN DECISION - SET number of design lead zero methods used TO (previous_number_of_design-lead-zero_methods_u
design lead pole
Bifurcation - If RL pass throug Design Area ---> TRUE
Bifurcation - If Exists valid range of K ---> TRUE
Bifurcation - If design accomplishes specifications ---> FALSE

DESIGN PARAMETERS VALUES OBTAINED IN THIS DESIGN STEP

Compensator type hypothesis---->lead
real part of lead zero---->-1.9047619
imaginary part of lead zero---->0.0
imaginary part of lead pole---->0.0
real part of lead pole---->-4.8638654

poles of $\dfrac{27.135185 * s + 51.68607}{1.0 * s^{3} + 5.9638653 * s^{2} + 32.585438 * s + 52.172455}$ is [ -2.137663+1.0E-20*i -1.9131012+-4.5548162*i -1.9131012+4.5548162*i ]

close window [x]

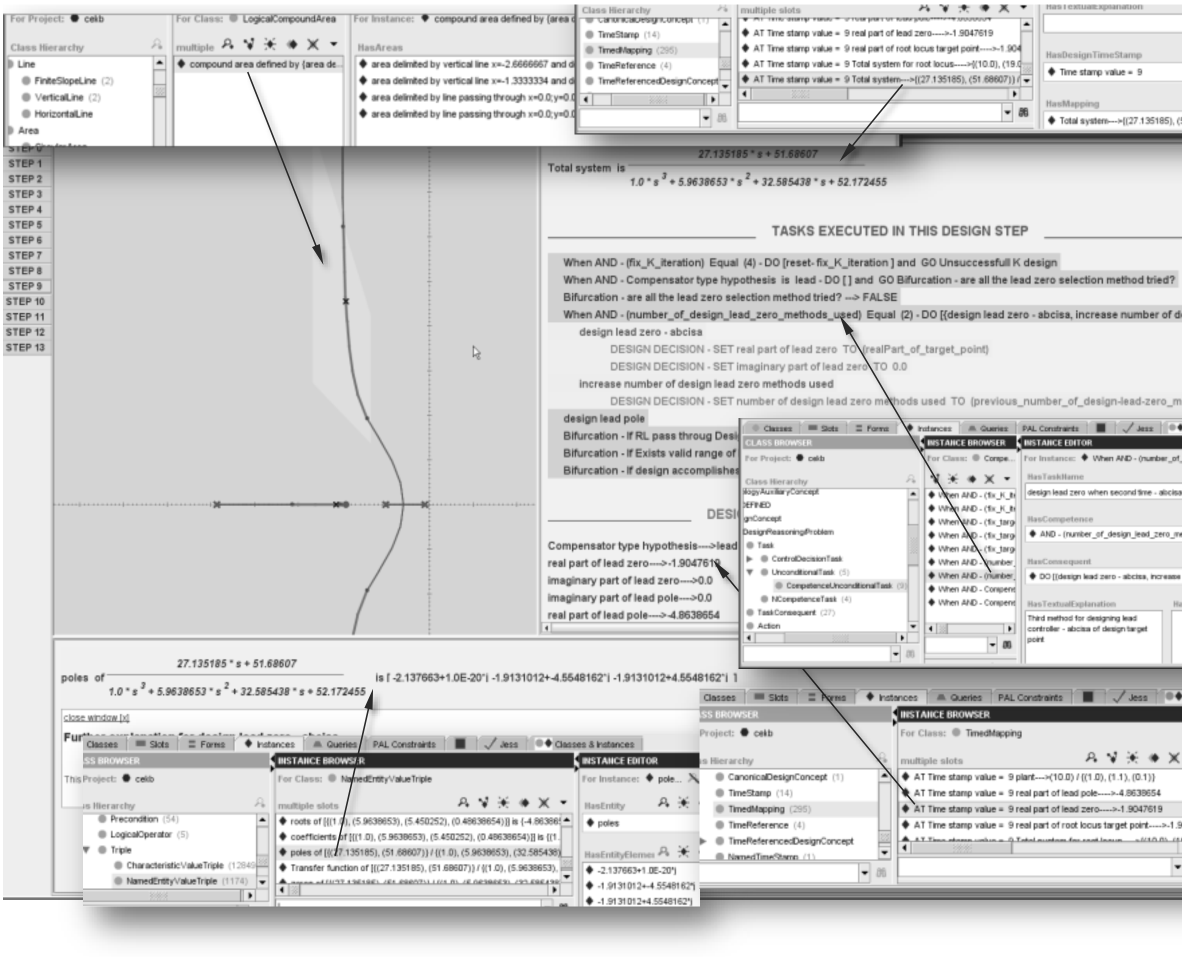
**Further explanation for design lead zero - abcisa**

This method places the lead zero right below the design point. You can see this example from professor Guy Beale or this exercise

Figure 14. Main window of the application

Every element in the graphical user interface is generated from the elements in the ontology and the knowledge base (figure 15 gives some examples of concepts in the ontology and their corresponding graphical representation), moreover, the user can freely interact with them. This approach is an implementation of the Model-Based User Interface paradigm[55]. A similar, though web-oriented, approach, also using ontologies as knowledge models, is presented in [56].

Each time the user right-clicks on an element, a contextual menu is generated on the fly by querying the ontology and retrieving the different menu items according to the things that are stored in the ontology regarding the corresponding concept. In addition, any explanation about the concepts and the tasks executed during the design process is also automatically obtained from the formal structures of the ontology according to the user needs.



Total system is $\frac{27.135185 * s + 51.68607}{1.0 * s^3 + 5.9638653 * s^2 + 32.585438 * s + 52.172455}$

TASKS EXECUTED IN THIS DESIGN STEP

When AND - (fix_K_iteration) Equal (4) - DO [reset- fix_K_iteration ] and GO Unsuccessfull K design
When AND - Compensator type hypothesis is lead - DO [ ] and GO Bifurcation - are all the lead zero selection method tried?
Bifurcation - are all the lead zero selection method tried? --> FALSE
When AND - (number_of_design_lead_zero_methods_used) Equal (2) - DO [{design lead zero - abcisa, increase number of d
design lead zero - abcisa
DESIGN DECISION - SET real part of lead zero TO (realPart_of_target_point)
DESIGN DECISION - SET imaginary part of lead zero TO 0.0
increase number of design lead zero methods used
DESIGN DECISION - SET number of design lead zero methods used TO (previous_number_of_design-lead-zero_m
design lead pole
Bifurcation - If RL pass throug Desi
Bifurcation - If Exists valid range of
Bifurcation - If design accomplishes

Compensator type hypothesis---->lead
real part of lead zero---->-1.9047619
imaginary part of lead zero---->0.0
imaginary part of lead pole---->0.0
real part of lead pole---->-4.8638654

poles of $\frac{27.135185 * s + 51.68607}{1.0 * s^3 + 5.9638653 * s^2 + 32.585438 * s + 52.172455}$ is [ -2.137663+1.0E-20*i -1.9131012+-4.5548162*i -1.9131012+4.5548162*i ]

AT Time stamp value = 9 real part of lead zero---->-1.9047619
AT Time stamp value = 9 real part of root locus target point---->-1.904
AT Time stamp value = 9 Total system for root locus---->[(10.0), (19.0
AT Time stamp value = 9 Total system--->[(27.135185), (51.68607)] /
HasDesignTimeStamp: Time stamp value = 9
HasMapping: Total system--->[(27.135185), (5

HasTaskName: design lead zero when second time - abcisa
HasCompetence: AND - (number_of_design_lead_zero_m
HasConsequent: DO [{design lead zero - abcisa, increase
HasTextualExplanation: Third method for designing lead controller - abcisa of design target point

AT Time stamp value = 9 plant--->(10.0) / [(1.0), (1.1), (0.1)]
AT Time stamp value = 9 real part of lead pole---->-4.8638654
AT Time stamp value = 9 real part of lead zero---->-1.9047619
AT Time stamp value = 9 real part of root locus target point---->-1.9

roots of [{(1.0), (5.9638653), (5.450252), (0.48638654)}] is {-4.86386
coefficients of [{(1.0), (5.9638653), (5.450252), (0.48638654)}] is {(1.
poles of [{(27.135185), (51.68607)} / {(1.0), (5.9638653), (32.585438)
Transfer function of [{(27.135185), (51.68607)} / {(1.0), (5.9638653),
HasEntity: poles
-2.137663+1.0E-20*i
-1.9131012+-4.5548162*i
-1.9131012+4.5548162*i

Figure 15. Main window of the application, showing the relationship to concepts in the ontology.

## 8. Discussion

The approach described in this paper focuses and stresses the representation (in the computer) of the control engineering concepts themselves, independently of any software component or any particular CACSD application. The use of ontologies as the representing formalism allows the resulting conceptualization to be both computable and independent of any actual realization in software. This conceptual structure is highly reusable and may later be used within a number of software applications or for building other ontologies.

Maintainability of the system is greatly eased with this approach. The roles of the knowledge engineer and the control engineer are clearly separated. Once the ontology is built, a control engineer can modify or insert new design methods by only knowing about a high-level, human-like syntax, language. Also, the same ontology can be reused for a number of different CACSD applications without the need to acquire knowledge engineering skills or with much less effort than if making it from scratch. This helps to solve what formerly was found to be one of the biggest drawbacks when building CACSD software with knowledge modeling techniques, as stated in [57].

The ontology, as well as being reusable, also acts as a knowledge acquisition tool [58], helping to manage the so-called knowledge acquisition bottleneck [59], one of the biggest problems when creating a knowledge model for a complex domain.

As mentioned earlier, in order to solve a given design problem by using the knowledge model, the knowledge inside the ontology must be somehow "executed". In the case of this study, a rule-based system is automatically generated from the conceptual structures. This rule-based system must be generated only once, unless there is a change in the knowledge inside the ontology. In addition to the separation between knowledge and control engineer roles, also the cumbersome structure of the rule based system is separated from the much more convenient syntax of the ontology constructs.

All these facts suppose a number of advantages compared to some of the CACSD systems of the 1980's and 1990's (see, for example, [60] or [61]). There, the roles of the knowledge and control engineer were hard so separate and the resulting rule based systems were almost unmanageable. In those systems, any explanation facility had to be hardcoded, while with ontologies these explanations can be automatically generated from the formal structure of the concept definitions. As an example, in [61] the concept "poles" was a property of the software object "system". Poles got calculated somehow and set as the value of this property. If one wanted a definition of poles, it had to be hardcoded elsewhere. With the ontology approach, the concept of poles is formally defined and this definition is the base to (automatically) generate the code that calculates them, while being also the source for the explanation of the concept itself.

A comparison with CACSD software developed from the 1990's is also possible, but difficult to achieve and of limited use because of the different objective that guide the approach described in this paper. Current, evolved, CACSD tools are focused on taking advantage of the graphical and numerical manipulation capacities of the modern computers [62, 63] while the work presented here puts the focus on taking advantage of the knowledge representation capabilities of the computers, that have been greatly improved during the las two decades of evolution of this discipline. This way, tools like MATRIXx, Matlab Control System Design App, or Labview Control Design and Simulation assume that the user has a good knowledge about the controller design methods, and the tools are focused on offering a good user interface and presenting the graphics that show the behavior of the system while being designed. Other tools are specifically oriented to education and offer a great level of interactivity with the user, showing how the variations of a given design parameter affects the behavior of the system [64, 65]. These tools may include additional, text or web-based material for explaining the design, but in a non-structured plain-text or html format. In contrast, the tool described in this paper tries to adapt to any level of user knowledge about the design procedure. As the design is performed automatically (without user intervention), it will reflect the expertise of the control engineer that initially created the knowledge model describing the process. By storing all the design procedure at a great level of detail, the user can interact with the system (once the controller is designed) and obtain the information he may need of any design step achieved. Different users, according to their knowledge, may require different set of explanations from the system. In this sense, the tool can be a good educational and/or training platform. For example, a collection of solved exercises could be generated, validated and stored, and later put at the disposal of any user that may want to learn about the given compensator design process.

The formal structure of ontologies makes it easier to translate among different representation formalisms that can be used by different software components. As an example, a wrapper was built in order to automate the communication with numerical calculus applications like Matlab or Maple.

As regards the usability of the tool, the fact that every element in the graphical user interface is generated from the ontology has a number of advantages. The application can show a glimpse of the design procedure and the user can ask any question about any of the design iterations. Every user has a different experience of the application depending on the kind of interaction he demands. The user can concentrate in what he wants to explore, he can skip the known facts and concentrate and interact with the unknown ones. No explanation is hardcoded anywhere; they are automatically generated from the conceptual structure whenever they are needed.

The kind of problems used for building the ontology and the knowledge-based system described in this paper are quite simple from the point of view of current, real controller design scenarios. This was an initial decision made because the main objective was to build a proof-of-concept application in order to find a suitable ontology-based representation for every conceptual structure involved in the root locus design method, that is, representation of transfer functions, block diagrams, mathematical formulae, quantitative and qualitative characteristics, the root locus graph components, etc. Choosing a more complex design problem (a multivariable problem, using different control topologies, controller formats or design methods) would have taken the research apart from its main initial objective, because it would have needed extra time to develop a bigger ontology, while the set of conceptual structures to be represented would have been, in essence, the same. Once this first approach has proven to be functional, the extension of the idea to more complex scenarios is feasible.

It would only take some extra time, and the help of an expert on the given design method/s, to adapt or extend the ontology to suit this higher complexity.

In this sense, future work could include the development of conceptual structures for representing other, different design methods like the frequency-response ones (Bode, Nyquist), other control configurations (more control loops, several inputs and outputs

Other compensator types: PID

## References

[1] Bilqees, A. (1996) Computing environments for control engineering. PhD. Thesis, Cambridge University. Online. Available: http://citeseer.ist.psu.edu/rd/75021831%2C610741%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/29125/http:zSzzSzwww-control.eng.cam.ac.ukzSzabszSzthesis.pdf/computing-environments-for.pdf

[2] T. Varsamidis, S. Hope and C. P. Jobling (1996) Integration using a unified model for CACSD. IEE Colloquium on Advances in Computer-Aided Control System Design (Digest No: 1996/061), pp. 2/1 - 2/4, 10.1049/ic:19960420.

[3] N. Guarino, D. Oberle and S. Staab (2009) What is an ontology? In Staab, S. and Studer, R. (eds.) Ontological Engineering (International Handbooks on Information Systems), Springer, pp. 1 – 20., 10.1007/978-3-540-92673-3

[4] Michel and Gauthier Associates (1996) Advanced Continuous Simulation Language (ACSL), Concord, Massachusetts.

[5] H. Elmqvist (1978) A Structured Model Language for Large Continuous Systems. Phd thesis, Department of Automatic Control, Lund Institute of Technology, Lund.

[6] S. E. Mattson, M. Anderson, and K. J. Åström (1993) Object oriented modeling and simulation. In D. A. Linkens, editor. CAD for Control Systems. Marcel Dekker, Inc.

[7] S. E. Mattson and H. Elmqvist (1997) Modelica - an international effort to design the next generation modeling language. Proceedings of the 7th IFAC Symposium on Computer Aided Control Systems Design, CACSD'97, Gent, Belgium, April 28-30, 1997, 10.1016/S1474-6670(17)43628-7.

[8] Object Management Group (OMG) (1997). Unified Modeling Language: Semantics 1.1 Final Adopted Specification ptc/97-08-04. Online: www.omg.org.

[9] Object Management Group (OMG) - Systems Engineering Domain Special Interest Group (2006) SysML Specification v1.0. Online: http://sysml.org/docs/specs/OMGSysML-v1.0-07-09-01.pdf

[10] B. Willard (2007) UML for systems engineering, Computer Standards & Interfaces, Vol. 29(1), pp. 69 – 81, https://doi.org/10.1016/j.csi.2005.12.006

[11] N. Munro (1990) ECSTASY – A Control System CAD Environment. In Proceedings of the 11th IFAC World Congress on Automatic Control, Tallinn, https://doi.org/10.1016/S1474-6670(17)51760-7

[12] J. H. Taylor and D. K. Frederick (1984) An expert system architecture for computer-aided control engineering. In Proceedings of the IEEE, 72, December 1984, 10.1109/PROC.1984.13087

[13] G. Grübel (1995) The ANDECS CACE Framework. IEEE Control Systems, 15(2), pp. 8 – 13, 10.1109/37.375278.

[14] G. Grübel (1994) The ANDECS-CACE framework A-RSYST for integrated analysis and design of controlled systems. In Proceedings of IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, pp. 389 – 394, 10.1109/CACSD.1994.288900

[15] International Organization for Standarization (ISO) (1992) Language EXPRESS reference manual, ISO 10303, Industrial Automation Systems and Integration – Part 11.

[16] T. Varsamidis, S. Hope and C. P. Jobling, C.P. (1994) Information management for control system designers. In Proceedings of the IEE International Conference on Control (Control '94), 1, pp 13 – 17, 10.1049/cp:19940101

[17] E. Herzog and A. Törne (2001) Information modelling for system specification representation and data exchange. In Proceedings of ECBS 2001, Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 136 – 143, 10.1109/ECBS.2001.922415.

[18] J. Lubell, R. D. Peak, V. Srinivasan and S. Waterbury (2004) STEP, XML, and UML: complementary technologies. Paper DETC2004-57743, American Society of Mechanical Engineers ASME 2004, Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Salt Lake City, 10.1115/1.1818683

[19] E. Herzog (2005) SysML – an assessment, In Proceedings of the 15th International Council on Systems Engineering INCOSE International Symposium, 10.1002/j.2334-5837.2005.tb00670.x

[20] T. R. Gruber (1995) Towards principles for the design of ontologies used for knowledge sharing. International Journal of Human and Computer Studies, vol. 43, pp. 907 – 928, https://doi.org/10.1006/ijhc.1995.1081

[21] T. Berners-Lee, J. Hendler and O. Lassila (2001) The Semantic Web. Scientific American, May 2001.

[22] Brodraric, B. Gahegan, M. (2010) Ontology use for semantic e-Science. Semantic Web 1, IOS Press, pp. 149 – 153, 10.3233/SW-2010-0021

[23] P. Morgan, J. A. Cafeo, K. Godden, R. M. Lesperance, A. M. Simon, D. L. McGuinness and J. L. Benedict (2005) The General Motors' variation-reduction adviser. AI Magazine, 26(2), pp. 269 – 276, 10.1002/isaf.236

[24] K.-Y. Kim, D. G. Manley, and H. Yang (2006) Ontology-Based Assembly Design and Information Sharing for Collaborative Product Development, Computer Aided Design, Vol. 38, pp. 1233-1250, https://doi.org/10.1016/j.cad.2006.08.004

[25] W. Marquardt, J. Morbach, A. Wiesner and A. Yang (2010) OntoCAPE: A Re-Usable Ontology for Chemical Process Engineering. Springer-Verlag Berlin Heidelberg

[26] T. Tudorache (2008) Ontologies in Engineering: Modeling, Consistency and Use Cases, VDM Verlag

[27] Prestes, E.; Carbonera, J. L; Fiorini, S. R.; Jorge, V. A. M.; Abel, M. Madhavan, R; Locoro, A.; Goncalves, P.; Barreto, M. E.; Habib, M.; Chibani, A.; Gérard, S.; Amirat, Y.; Schlenoff, C. (2013) Towards a core ontology for robotics and automation. Robotics and Autonomous Systems, 61(11), pp. 1193-1204, 10.1016/j.robot.2013.04.005

[28] R. Hodgson (2011) TCMX - Ontology-Based Specifications for Space Systems Telemetry, Commanding and Messaging. In Proceedings of The 13th NASA-ESA Workshop on Product Data Exchange, 11 - 13 May 2011

[29] NASA (2006) Semantic web for earth and environmental terminology (SWEET). Jet Propulsión Laboratory, California Institute of Technology, 2006. Online: http://sweet.jpl.nasa.gov/

[30] Z. Hu, E. Kruse and L. Draws (2003) Intelligent binding in the engineering of automation systems using ontology and Web services. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, 33(3), pp 403 – 412, 10.1109/TSMCC.2003.817346

[31] C. Chandra, and A. Tumanyan (2007) Organization and problem ontology for supply chain information support system. Data & Knowledge Engineering, 61(2), pp. 263 – 280, https://doi.org/10.1016/j.datak.2006.06.005

[32] Scheuermann, A. and Leukel, J. (2014) Supply chain management ontology from an ontology engineering perspective. Computers in Industry, 65(6), pp. 913-923, https://doi.org/10.1016/j.compind.2014.02.009

[33] H. K. Lin, and J. A. Harding (2007) A manufacturing system engineering web ontology model on the Semantic Web for inter-enterprise collaboration', Computers in Industry, 58, 2007, pp 428 – 437, https://doi.org/10.1016/j.compind.2006.09.015.

[34] O. Lukibanov (2005) Use of ontologies to support design activities at DaimlerChrysler, In Proceedings of the 8th International Protégé Symposium, Madrid, 2005.

[35] S. Ahmed, S. Kim and K. Wallace (2005). A Methodology for Creating Ontologies for Engineering Design, 2005 ASME IDETC/CIE, 10.1115/1.2720879

[36] V. Liang, and C. Paredis (2004). A Port Ontology for Conceptual Design of Systems. Journal of Computing and Information Science in Engineering, 4 (3), pp. 206 – 217, 10.1115/1.1778191

[37] G. Mocko, D. Rosen and F. Mistree (September 2007). Decision Retrieval and Storage Enabled through Description Logic, 2007 ASME Computers and Information in Engineering Conference. Las Vegas, NV, 10.1115/DETC2007-35644

[38] J. A. Rockwell, I. R. Grosse, S. Krishnamurty and J. C. Wileden (2010) Semantic Information Model for Capturing and Communicating Design Decisions, J. Comput. Inf. Sci. Eng., 10(3), 10.1115/1.3462926

[39] Y. Kitamura and R. Mizoguchi (2010) Characterizing Functions based on Ontological Models from an Engineering Point of View. In Proceedings of the Sixth International Conference on Formal Ontology in Information Systems (FOIS 2010), Toronto, Canada, May 11-14, 2010, pp. 301-314, IOS Press, 10.3233/978-1-60750-535-8-301

[40] R. Mizoguchi and Y. Kitamura (2009) A Functional Ontology of Artifacts. The Monist - An Int'l Quarterly J. of General Philosophical Inquiry, Vol. 92, No.3, pp. 387-402.

[41] N. Chungoora and R.I.M. Young (2011) A Framework to Support Semantic Interoperability in Product Design and Manufacture. Global Product Development, Part 12, Springer, pp. 435-443, https://doi.org/10.1007/978-3-642-15973-2_44

[42] D. Price, M. West, T. Christiansen and J. Kendall (2006) ISO 15926 as OWL - an ontological approach to data warehousing using OWL. In Proceedings of the PDE 2006, 8th NASA-ESA Workshop on Product Data Exchange (PDE).

[43] A. Barnard (2009) Future STEP Project. In Proceedings of The 11th NASA-ESA Workshop on Product Data Exchange, 29 April - 1 May 2009

[44] G. Trapp (1993) The emerging STEP standard for product-model data exchange, Computer, 26(2), pp. 85 – 87, 0.1109/2.192004

[45] Barbau, R.; Krima, S; Sachuri, S.; Narayanan, A.; Fiorentini, X.; Foufou, S. and Sriram, R. D (2012) OntoSTEP: Enriching product model data using ontologies. Comput. Aided Des. 44(6), pp. 575-590, https://doi.org/10.1016/j.cad.2012.01.008

[46] S. Waterbury (2007) The Pan Galactic status report (an update on the Pan Galactic Engineering Framework [PGEF]).in Proceedings of the 9th NASA-ESA Workshop on Product Data Exchange (PDE), Santa Barbara, CA, USA.

[47] Pauwels, P. (2015) ifcOWL: The EXPRESS to OWL Conversion Pattern, online: http//www.w3.org/community/lbd/ifcowl/

[48] Terkaj, W.; Sojic, A. (2015) Ontology-based representation of IFC EXPRESS rules: An enhancement of the ifcOWL ontology, Automation in Construction. Vol 57 . pp 188-201, https://doi.org/10.1016/j.autcon.2015.04.010

[49] Ghafour, S.A., Ghodous, P., Khosrowshahi, F., Perna, E., & Shariat, B. (2014). Semantic interoperability of knowledge in feature-based CAD models. Computer-Aided Design, 56, 45-57, https://doi.org/10.1016/j.cad.2014.06.001

[50] Corcho, Ó., Fernández-López, M., & Gómez-Pérez, A. (2003). Methodologies, tools and languages for building ontologies: Where is their meeting point? Data Knowl. Eng., 46, 41-64, 10.1016/S0169-023X(02)00195-7

[51] M. Fernández-López, A. Gómez-Pérez and N. Juristo (1997) METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, pp 33–40

[52] C. Bissell (1993) A new way of talking: aspects of the creation of the language of control engineering, Faculty of Technology/Systems Architecture Group Internal Report (SAG/1993/RR31/CCB), November 1993

[53] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy and S. W. Tu (2002) The evolution of Protégé: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford Medical Institute, https://doi.org/10.1016/S1071-5819(02)00127-1

[54] E. Friedman-Hill (2011) JESS (Java Expert System Shell) Online: http://herzberg.ca.sandia.gov/jess

[55] Model-Based User Interface paradigm. Online: http://www.w3.org/TR/mbui-intro/

[56] Chavarriaga, E. and Macías, J. A. (2009) A model-driven approach to building modern Semantic Web-Based User Interfaces. Advances in Engineering Software, 40, pp. 1329–1334, https://doi.org/10.1016/j.advengsoft.2009.01.016

[57] B. P. Butz, N. F. Palumbo, R. C. Unterberger and D. G. Miller (1990) An expert system for control system design. Proceedings of the 5th IEEE International Symposium on Intelligent Control (vol.2), pp. 1156 – 1162, 10.1109/ISIC.1990.128600

[58] Shadbolt, Nigel, O'Hara, Kieron and Cottam, Hugh (2004) The Use of Ontologies for Knowledge Acquisition. In, Cuena, Jose, Demazeau, Yves, Garcia Serrano, Ana and Treur, Jan (eds.) Knowledge Engineering and Agent Technology, IOS Press, 19-42.

[59] Buchanan, B. G. and Wilkins, D C. (editors) (1993). Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems. Morgan Kaufmann

[60] P. A. Muha (1991) Expert system for SROOT. IEE Proceedings. - D, 138(4), pp. 381 – 387, 10.1049/ip-d.1991.0051

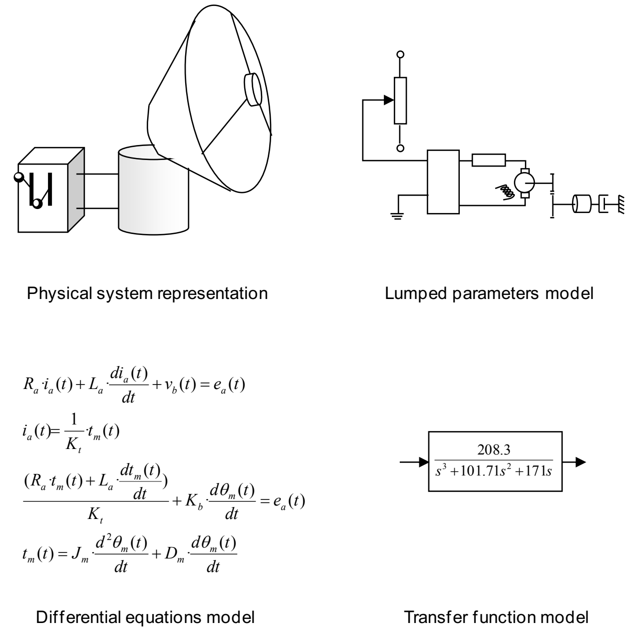
[61] G. Pang (1992) A Matrix and Expert system Development Aid Language, In Proceedings of the IEEE Symposium on CACSD, Napa, pp. 148 – 155, 10.1109/CACSD.1992.274438

[62] Varga, D. A. (2014). Computer-Aided Control Systems Design: Introduction and Historical Overview. In J. Baillieul & T. Samad (Eds.), *Encyclopedia of Systems and Control* (pp. 122–127). London: Springer London. https://doi.org/10.1007/978-1-4471-5102-9_138-2

[63] Sima, I. M. V. (2014). Interactive Environments and Software Tools for CACSD. In J. Baillieul & T. Samad (Eds.), *Encyclopedia of Systems and Control* (pp. 584–590). London: Springer London. https://doi.org/10.1007/978-1-4471-5102-9_139-1

[64] Díaz, J. M., & Dormido, S. (2015). ITADLS: An Interactive Tool for Analysis and Design of Linear Systems. IFAC-PapersOnLine, 48(29), 253-258. https://doi.org/10.1016/j.ifacol.2015.11.245

[65] Guzman JL, Åström KJ, Dormido S, Hägglund T, Berenguel M, Piguet Y (2008) Interactive learning modules for PID control. IEEE Control Syst Mag 28:118–134



Physical system representation

Lumped parameters model

$$R_a \cdot i_a(t) + L_a \cdot \frac{di_a(t)}{dt} + v_b(t) = e_a(t)$$

$$i_a(t) = \frac{1}{K_t} \cdot t_m(t)$$

$$\frac{(R_a \cdot t_m(t) + L_a \cdot \frac{dt_m(t)}{dt})}{K_t} + K_b \cdot \frac{d\theta_m(t)}{dt} = e_a(t)$$

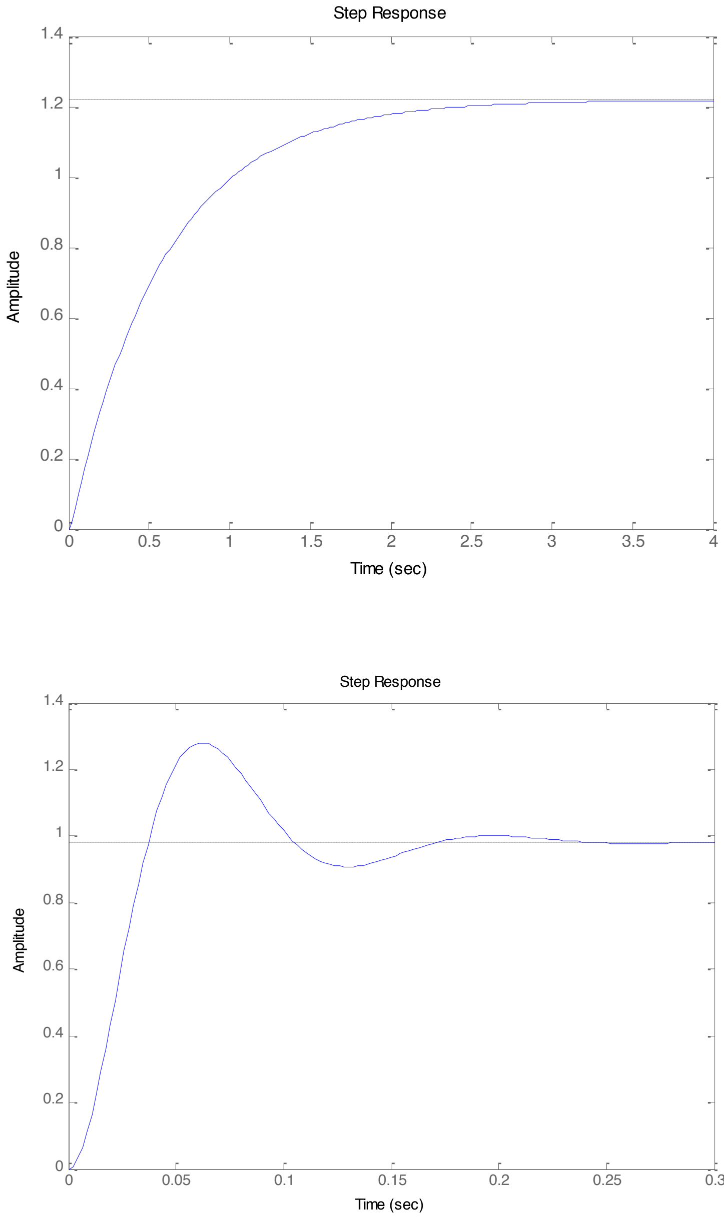
$$t_m(t) = J_m \cdot \frac{d^2\theta_m(t)}{dt} + D_m \cdot \frac{d\theta_m(t)}{dt}$$

$$\boxed{\frac{208.3}{s^3 + 101.71s^2 + 171s}}$$

Differential equations model

Transfer function model

Figure 1. Antenna positioning system with its lumped parameters, differential equations and transfer function models



(a)

(b)

Figure. 2. Different responses for two control strategies of an antenna positioning system.

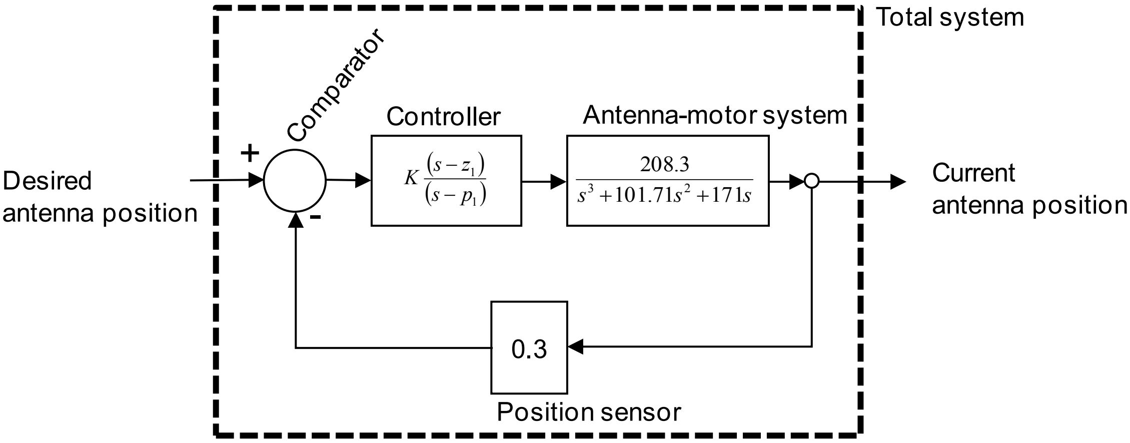


Figure 3. Antenna positioning system with feedback loop and controller.

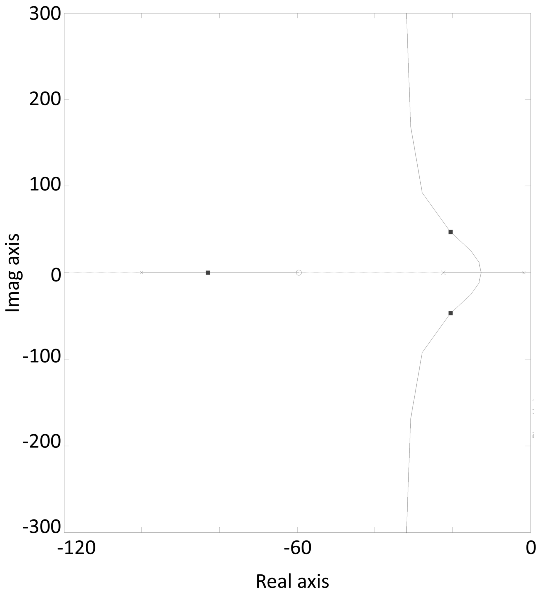


Figure. 4. Root locus of the total system in figure 3. Squared dots shows the positions of the poles of the total system for a given value of the parameter K in the controller.

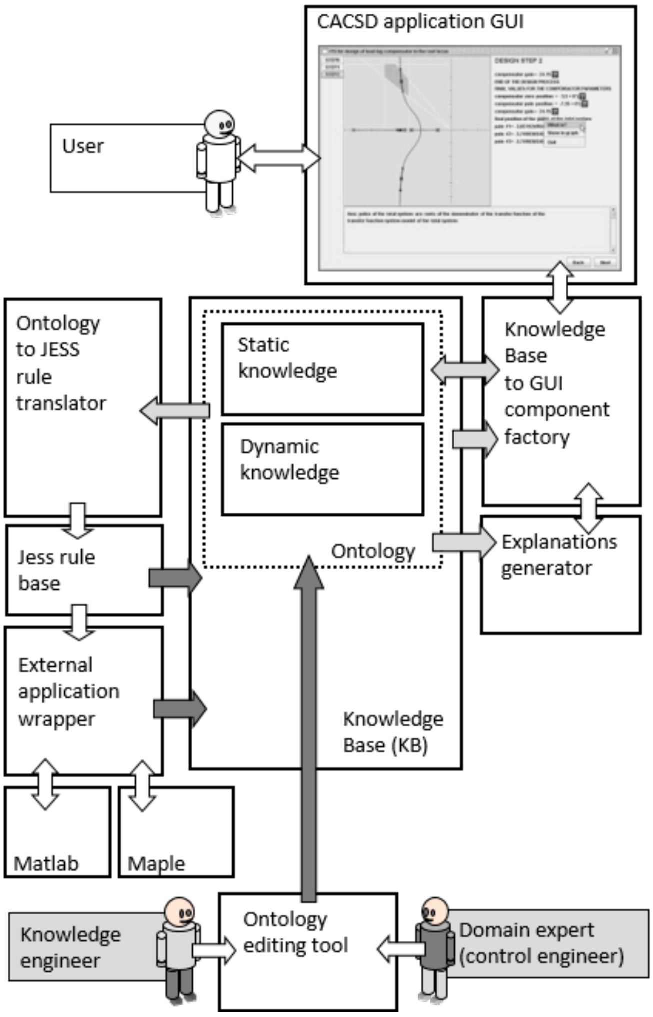


Figure 5. Architecture and building blocks

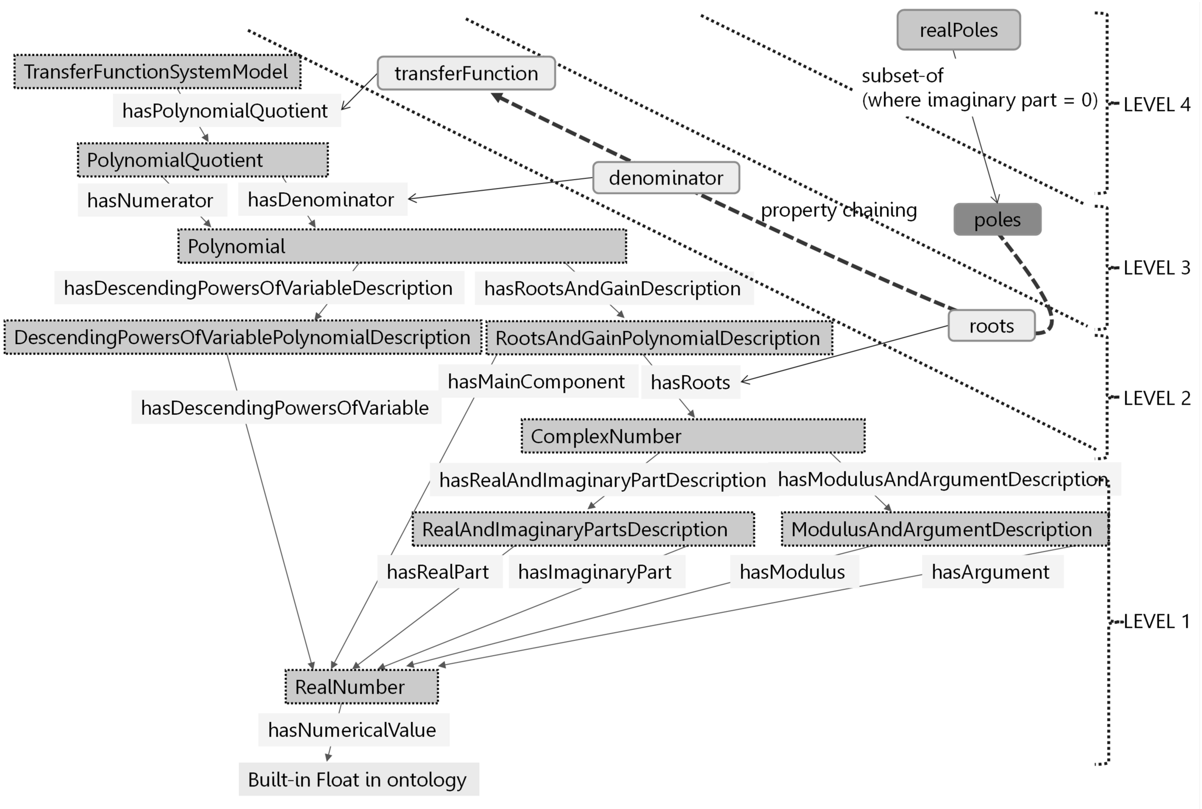

realPoles
LEVEL 4
subset-of
(where imaginary part = 0)
TransferFunctionSystemModel
transferFunction
hasPolynomialQuotient
PolynomialQuotient
denominator
hasNumerator
hasDenominator
property chaining
poles
LEVEL 3
Polynomial
hasDescendingPowersOfVariableDescription
hasRootsAndGainDescription
DescendingPowersOfVariablePolynomialDescription
RootsAndGainPolynomialDescription
roots
hasMainComponent
hasRoots
LEVEL 2
hasDescendingPowersOfVariable
ComplexNumber
hasRealAndImaginaryPartDescription
hasModulusAndArgumentDescription
RealAndImaginaryPartsDescription
ModulusAndArgumentDescription
hasRealPart
hasImaginaryPart
hasModulus
hasArgument
LEVEL 1
RealNumber
hasNumericalValue
Built-in Float in ontology

Figure 6. Basic and derived concepts for the mathematical structures representing the models of the dynamical systems.

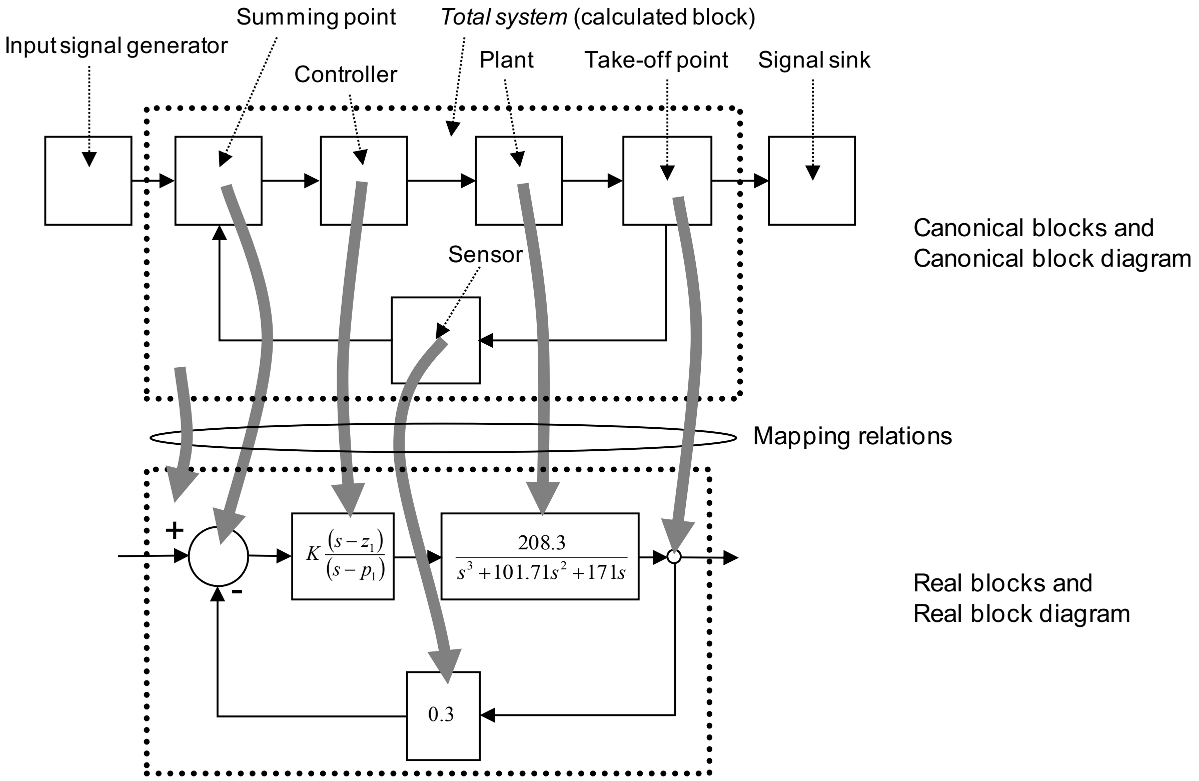
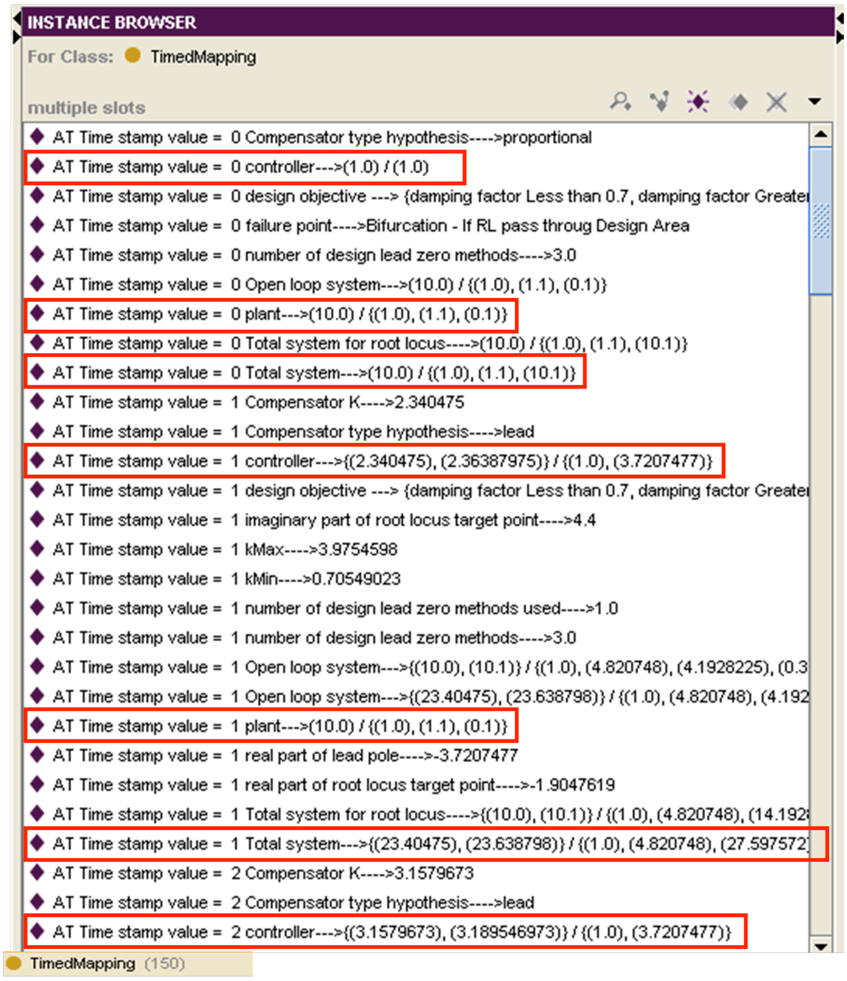
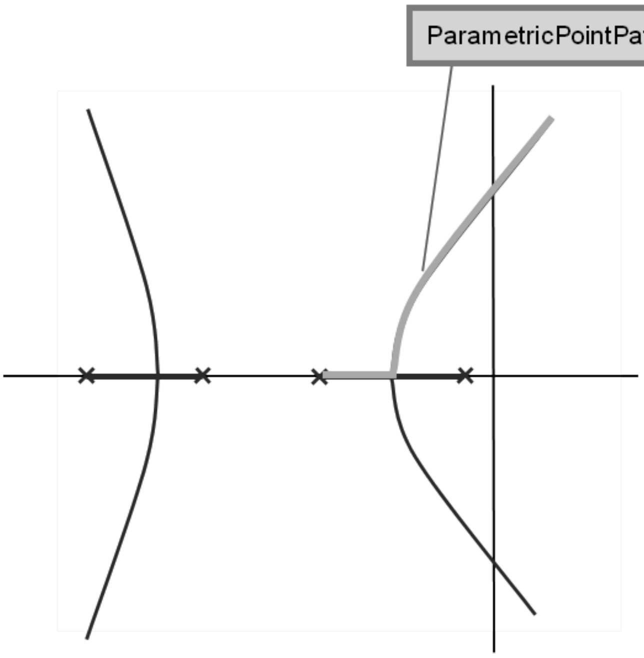


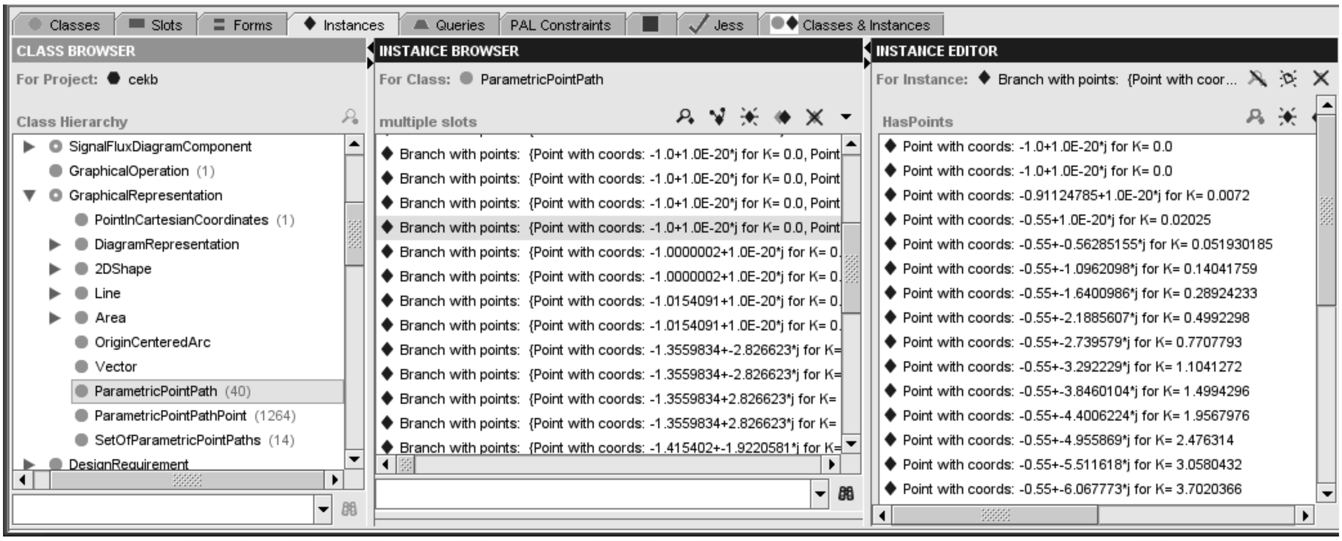
Figure 7. Canonical blocks and mapping to real blocks



```
AT Time stamp value =  0 Compensator type hypothesis---->proportional
AT Time stamp value =  0 controller--->(1.0) / (1.0)
AT Time stamp value =  0 design objective ---> {damping factor Less than 0.7, damping factor Greater
AT Time stamp value =  0 failure point---->Bifurcation - If RL pass throug Design Area
AT Time stamp value =  0 number of design lead zero methods---->3.0
AT Time stamp value =  0 Open loop system--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  0 plant--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  0 Total system for root locus---->(10.0) / {(1.0), (1.1), (10.1)}
AT Time stamp value =  0 Total system--->(10.0) / {(1.0), (1.1), (10.1)}
AT Time stamp value =  1 Compensator K---->2.340475
AT Time stamp value =  1 Compensator type hypothesis---->lead
AT Time stamp value =  1 controller--->{(2.340475), (2.36387975)} / {(1.0), (3.7207477)}
AT Time stamp value =  1 design objective ---> {damping factor Less than 0.7, damping factor Greater
AT Time stamp value =  1 imaginary part of root locus target point---->4.4
AT Time stamp value =  1 kMax---->3.9754598
AT Time stamp value =  1 kMin---->0.70549023
AT Time stamp value =  1 number of design lead zero methods used---->1.0
AT Time stamp value =  1 number of design lead zero methods---->3.0
AT Time stamp value =  1 Open loop system--->{(10.0), (10.1)} / {(1.0), (4.820748), (4.1928225), (0.3
AT Time stamp value =  1 Open loop system--->{(23.40475), (23.638798)} / {(1.0), (4.820748), (4.192
AT Time stamp value =  1 plant--->(10.0) / {(1.0), (1.1), (0.1)}
AT Time stamp value =  1 real part of lead pole---->-3.7207477
AT Time stamp value =  1 real part of root locus target point---->-1.9047619
AT Time stamp value =  1 Total system for root locus---->{(10.0), (10.1)} / {(1.0), (4.820748), (14.192
AT Time stamp value =  1 Total system--->{(23.40475), (23.638798)} / {(1.0), (4.820748), (27.597572
AT Time stamp value =  2 Compensator K---->3.1579673
AT Time stamp value =  2 Compensator type hypothesis---->lead
AT Time stamp value =  2 controller--->{(3.1579673), (3.189546973)} / {(1.0), (3.7207477)}
```

Figure 8. Canonical blocks with timed mappings in the ontology.





CLASS BROWSER

For Project: cekb

Class Hierarchy

```
▶ SignalFluxDiagramComponent
  GraphicalOperation (1)
▼ GraphicalRepresentation
    PointInCartesianCoordinates (1)
  ▶ DiagramRepresentation
  ▶ 2DShape
  ▶ Line
  ▶ Area
    OriginCenteredArc
    Vector
    ParametricPointPath (40)
    ParametricPointPathPoint (1264)
    SetOfParametricPointPaths (14)
▶ DesignRequirement
```

INSTANCE BROWSER

For Class: ParametricPointPath

multiple slots

```
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0+1.0E-20*j for K= 0.0, Point
Branch with points:  {Point with coords: -1.0000002+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0000002+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0154091+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.0154091+1.0E-20*j for K= 0.
Branch with points:  {Point with coords: -1.3559834+-2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+-2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+2.826623*j for K=
Branch with points:  {Point with coords: -1.3559834+2.826623*j for K=
Branch with points:  {Point with coords: -1.415402+-1.9220581*j for K=
```

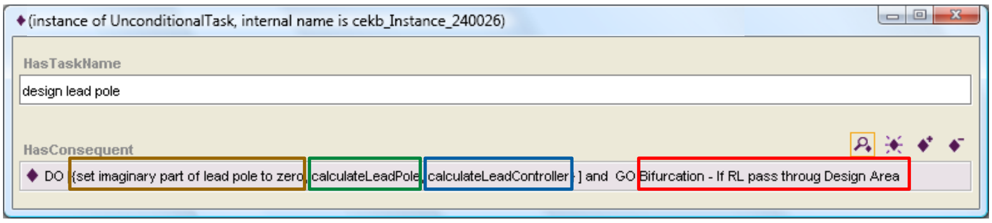
INSTANCE EDITOR

For Instance: Branch with points: {Point with coor...

HasPoints

```
Point with coords: -1.0+1.0E-20*j for K= 0.0
Point with coords: -1.0+1.0E-20*j for K= 0.0
Point with coords: -0.91124785+1.0E-20*j for K= 0.0072
Point with coords: -0.55+1.0E-20*j for K= 0.02025
Point with coords: -0.55+-0.56285155*j for K= 0.051930185
Point with coords: -0.55+-1.0962098*j for K= 0.14041759
Point with coords: -0.55+-1.6400986*j for K= 0.28924233
Point with coords: -0.55+-2.1885607*j for K= 0.4992298
Point with coords: -0.55+-2.739579*j for K= 0.7707793
Point with coords: -0.55+-3.292229*j for K= 1.1041272
Point with coords: -0.55+-3.8460104*j for K= 1.4994296
Point with coords: -0.55+-4.4006224*j for K= 1.9567976
Point with coords: -0.55+-4.955869*j for K= 2.476314
Point with coords: -0.55+-5.511618*j for K= 3.0580432
Point with coords: -0.55+-6.067773*j for K= 3.7020366
```

Figure 9. Representation of the root locus



(instance of UnconditionalTask, internal name is cekb_Instance_240026)

HasTaskName

design lead pole

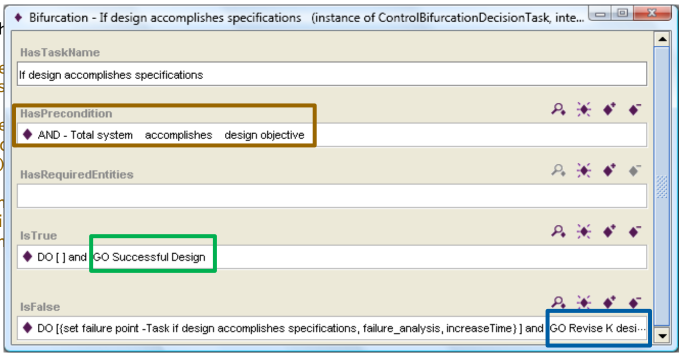
HasConsequent

DO [set imaginary part of lead pole to zero, calculateLeadPole, calculateLeadController ] and GO Bifurcation - If RL pass throug Design Area

```
      (defrule UnconditionalTask-design_lead_pole
      (object (is-a NamedTimeStamp)(hasTimeStampName "CURRENT")(hasTimeStamp ?currentTimeStamp))
      ?idActivator <- (design lead pole)
      =>
      (removeDesignParameterTimedMapping cekb_Instance_170010 ?currentTimeStamp )
      (bind ?mapping00 (make-instance of DesignParameterMapping
      (hasDesignParameter cekb_Instance_170010)
      (hasMappedDesignParameterValue  cekb_Instance_12)
       map))
DO    (make-instance of TimedMapping(hasMapping ?mapping00)
      (hasDesignTimeStamp  ?currentTimeStamp)
       map)
      (calculateLeadPole        (create$ cekb_Instance_170009 cekb_Instance_560006
      cekb_Instance_100011 cekb_Instance_150002 cekb_Instance_560001 )  )
      (calculateLeadController   (create$ )  )
GO    (assert (If RL pass throug Design Area))
      (retract ?idActivator)
      )
```

Figure 10. Representation of a simple task for assigning the lead pole.

```
(defrule ControlBifurcationDecisionTask-If_design_accomplishes_specifications-TRUE
?idActivator <- (If design accomplishes specifications)
(object (is-a NamedTimeStamp)(hasTimeStampName  "CURRENT")(h
;The precondition patterns:
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTime
(object (is-a CanonicalToRealMapping)(OBJECT  ?mapping0)(has
cekb_Instance_220000))  (hasRealInstance  ?realInstance0))
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTime
(object (is-a DesignConceptMapping)(OBJECT  ?mapping1)  (hasC
cekb_Instance_410019))(hasRealDesignConcept  ?realInstance1)
;The accomplishes function patterns (ts, %OS)
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCh
"settling time 2 percent"))  (hasElementToWhichCharacteristi
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCh
"damping factor"))  (hasElementToWhichCharacteristicApplies
(test (accomplishes ?realInstance0 ?realInstance1 ))
;The required Entity patterns:
=>
(assert (Successful Design))
(retract ?idActivator)
)
```



Bifurcation - If design accomplishes specifications   (instance of ControlBifurcationDecisionTask, inte...

HasTaskName
If design accomplishes specifications

HasPrecondition
AND - Total system   accomplishes   design objective

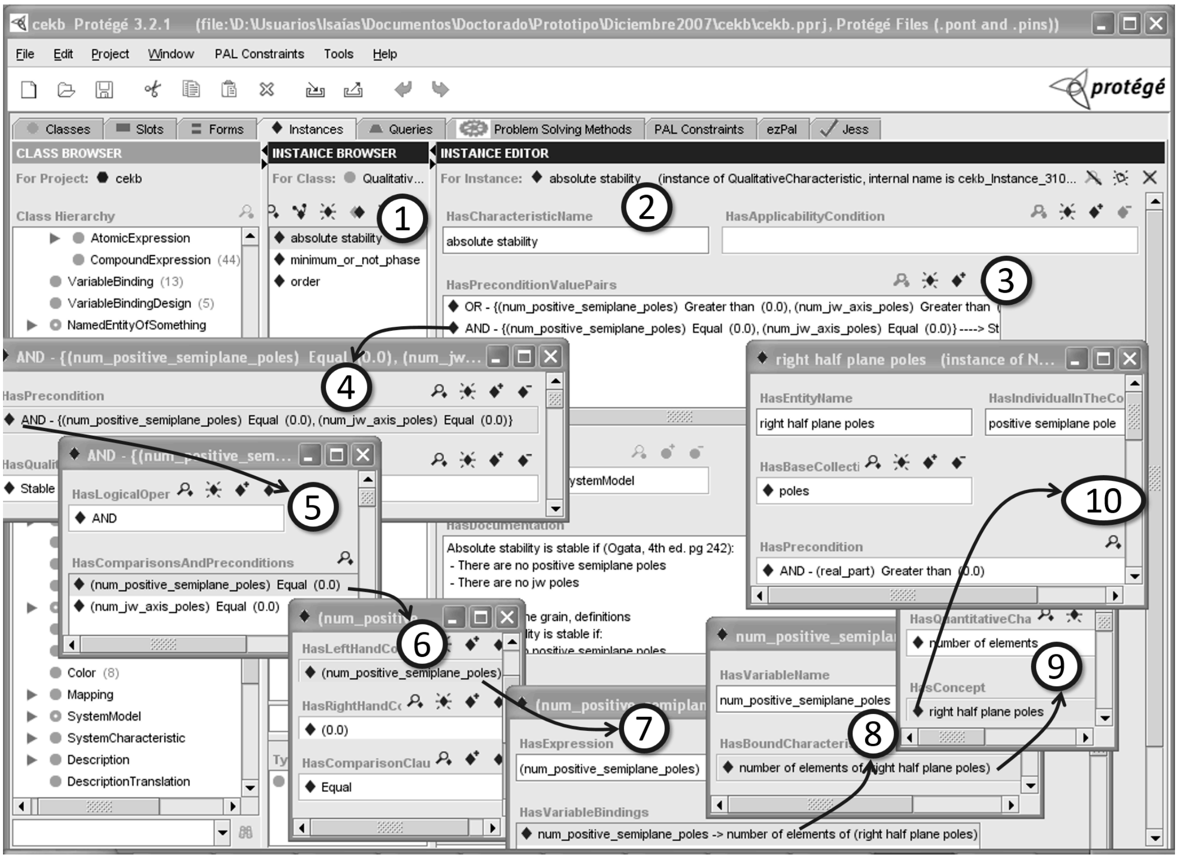
HasRequiredEntities

IsTrue
DO [ ] and GO Successful Design

IsFalse
DO [{set failure point -Task if design accomplishes specifications, failure_analysis, increaseTime} ] and GO Revise K desi...

```
(defrule ControlBifurcationDecisionTask-If_design_accomplishes_specifications-FALSE
?idActivator <- (If design accomplishes specifications)
(object (is-a NamedTimeStamp)(hasTimeStampName  "CURRENT")(hasTimeStamp  ?currentTimeStamp))
;The re;The precondition patterns:
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTimeStamp)(hasMapping  ?mapping0 ))
(object (is-a CanonicalToRealMapping)(OBJECT  ?mapping0)(hasCanonicalInstance  ?canonical0&:(= (instance-name  ?canonical0)
cekb_Instance_220000))  (hasRealInstance  ?realInstance0))
(object (is-a TimedMapping)(hasDesignTimeStamp  ?currentTimeStamp)(hasMapping  ?mapping1 ))
(object (is-a DesignConceptMapping)(OBJECT  ?mapping1)  (hasCanonicalDesignConcept  ?canonical1&:(= (instance-name  ?canonical1)
cekb_Instance_410019))(hasRealDesignConcept  ?realInstance1))
;The accomplishes function patterns (ts, %OS)
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCharacteristic  ?charac1&:(eq  (slot-get  ?charac1 hasCharacteristicName)
"settling time 2 percent"))  (hasElementToWhichCharacteristicApplies  ?realInstance0)  )
(object (is-a CharacteristicValueTriple)  (hasQuantitativeCharacteristic  ?charac2&:(eq  (slot-get  ?charac2  hasCharacteristicName)
"damping factor"))  (hasElementToWhichCharacteristicApplies  ?realInstance0)  )
(not(test (accomplishes ?realInstance0 ?realInstance1 )))
;The required Entity patterns:
                                                              =>
.......DO ACTIONS………..
(assert (Revise K design))
(retract ?idActivator)
)
```

Figure 11. Representation of the task testing if the current design is successful or not.



For Project: cekb

For Class: Qualitativ...

For Instance: absolute stability (instance of QualitativeCharacteristic, internal name is cekb_Instance_310...

HasCharacteristicName: absolute stability

HasApplicabilityCondition:

HasPreconditionValuePairs:
- OR - {(num_positive_semiplane_poles) Greater than (0.0), (num_jw_axis_poles) Greater than
- AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw_axis_poles) Equal (0.0)} ----> St

ystemModel

HasDocumentation:

Absolute stability is stable if (Ogata, 4th ed. pg 242):
- There are no positive semiplane poles
- There are no jw poles

he grain, definitions
lity is stable if:
positive semiplane poles

AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw...

HasPrecondition: AND - {(num_positive_semiplane_poles) Equal (0.0), (num_jw_axis_poles) Equal (0.0)}

HasQualit: Stable

AND - {(num_positive_sem...

HasLogicalOper: AND

HasComparisonsAndPreconditions:
- (num_positive_semiplane_poles) Equal (0.0)
- (num_jw_axis_poles) Equal (0.0)

(num_positive

HasLeftHandCo: (num_positive_semiplane_poles)

HasRightHandCo: (0.0)

HasComparisonClau: Equal

(num_positive_semiplan

HasExpression: (num_positive_semiplane_poles)

HasVariableBindings: num_positive_semiplane_poles -> number of elements of (right half plane poles)

num_positive_semiplan

HasVariableName: num_positive_semiplane_poles

HasBoundCharacteri: number of elements of (right half plane poles)

HasQuantitativeCha: number of elements

HasConcept: right half plane poles

right half plane poles (instance of N...

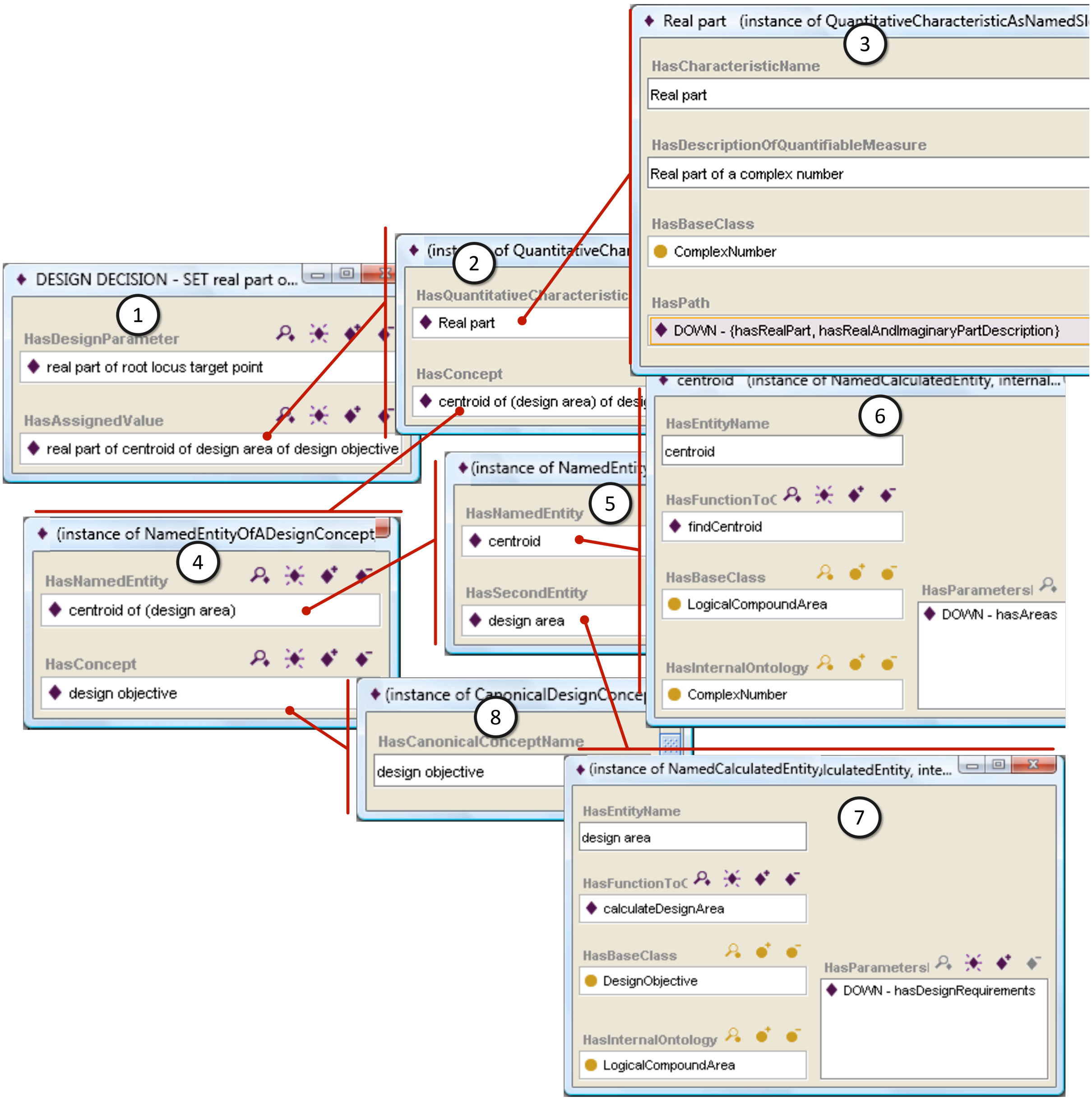
HasEntityName: right half plane poles

HasIndividualInTheCo: positive semiplane pole

HasBaseCollecti: poles

HasPrecondition: AND - (real_part) Greater than (0.0)

Figure 12. Part of the conceptualiztion of the concept "absolute stability"



DESIGN DECISION - SET real part o...

HasDesignParameter: real part of root locus target point

HasAssignedValue: real part of centroid of design area of design objective

(inst... of QuantitativeChar

HasQuantitativeCharacteristic: Real part

HasConcept: centroid of (design area) of desig

Real part (instance of QuantitativeCharacteristicAsNamedSI

HasCharacteristicName: Real part

HasDescriptionOfQuantifiableMeasure: Real part of a complex number

HasBaseClass: ComplexNumber

HasPath: DOWN - {hasRealPart, hasRealAndImaginaryPartDescription}

(instance of NamedEntityOfADesignConcept

HasNamedEntity: centroid of (design area)

HasConcept: design objective

(instance of NamedEntity

HasNamedEntity: centroid

HasSecondEntity: design area

centroid (instance of NamedCalculatedEntity, internal...

HasEntityName: centroid

HasFunctionToC: findCentroid

HasBaseClass: LogicalCompoundArea

HasInternalOntology: ComplexNumber

HasParametersI: DOWN - hasAreas

(instance of CanonicalDesignConce

HasCanonicalConceptName: design objective

(instance of NamedCalculatedEntity,lculatedEntity, inte...

HasEntityName: design area

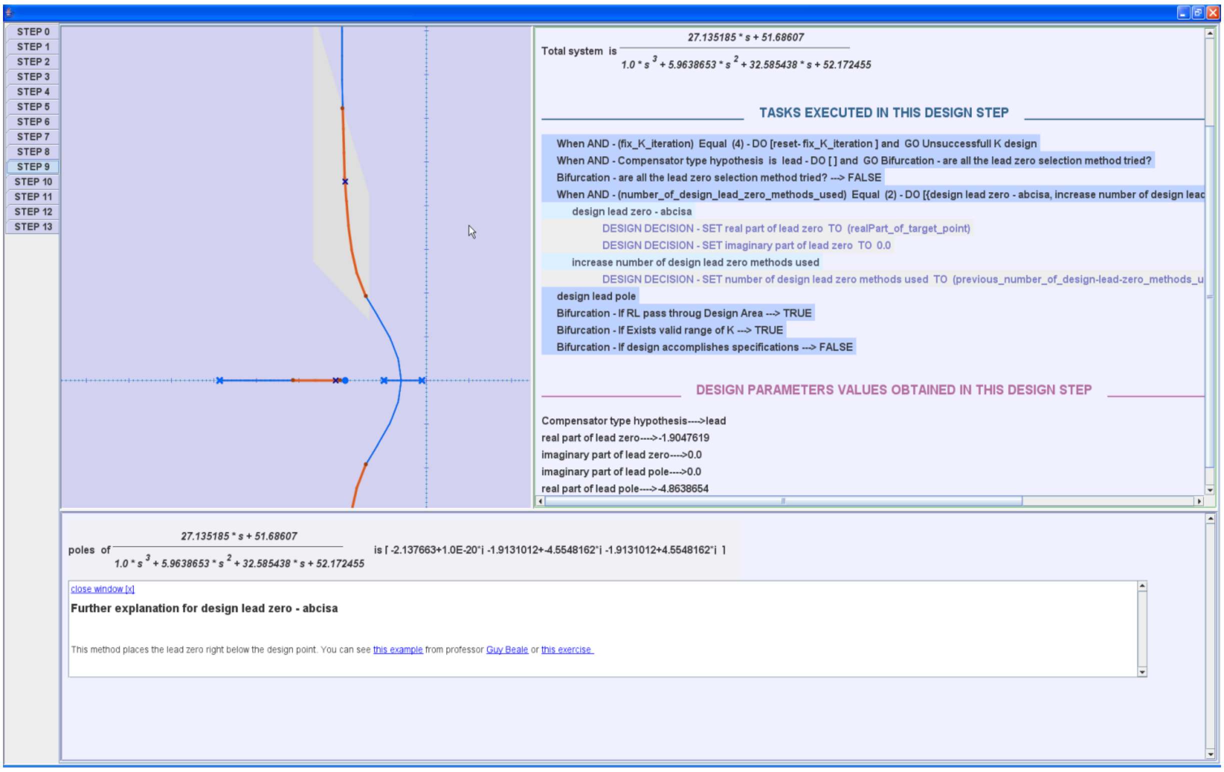
HasFunctionToC: calculateDesignArea

HasBaseClass: DesignObjective

HasInternalOntology: LogicalCompoundArea

HasParametersI: DOWN - hasDesignRequirements

Figure 13. Part of the conceptualization of a design decision



Total system is $\frac{27.135185 * s + 51.68607}{1.0 * s^{3} + 5.9638653 * s^{2} + 32.585438 * s + 52.172455}$

TASKS EXECUTED IN THIS DESIGN STEP

```
When AND - (fix_K_iteration)  Equal  (4) - DO [reset- fix_K_iteration ] and  GO Unsuccessfull K design
When AND - Compensator type hypothesis  is  lead - DO [ ] and  GO Bifurcation - are all the lead zero selection method tried?
Bifurcation - are all the lead zero selection method tried? ---> FALSE
When AND - (number_of_design_lead_zero_methods_used)  Equal  (2) - DO [{design lead zero - abcisa, increase number of design leac
    design lead zero - abcisa
        DESIGN DECISION - SET real part of lead zero  TO  (realPart_of_target_point)
        DESIGN DECISION - SET imaginary part of lead zero  TO  0.0
    increase number of design lead zero methods used
        DESIGN DECISION - SET number of design lead zero methods used  TO  (previous_number_of_design-lead-zero_methods_u
design lead pole
Bifurcation - If RL pass throug Design Area ---> TRUE
Bifurcation - If Exists valid range of K ---> TRUE
Bifurcation - If design accomplishes specifications ---> FALSE
```

DESIGN PARAMETERS VALUES OBTAINED IN THIS DESIGN STEP

```
Compensator type hypothesis---->lead
real part of lead zero---->-1.9047619
imaginary part of lead zero---->0.0
imaginary part of lead pole---->0.0
real part of lead pole---->-4.8638654
```

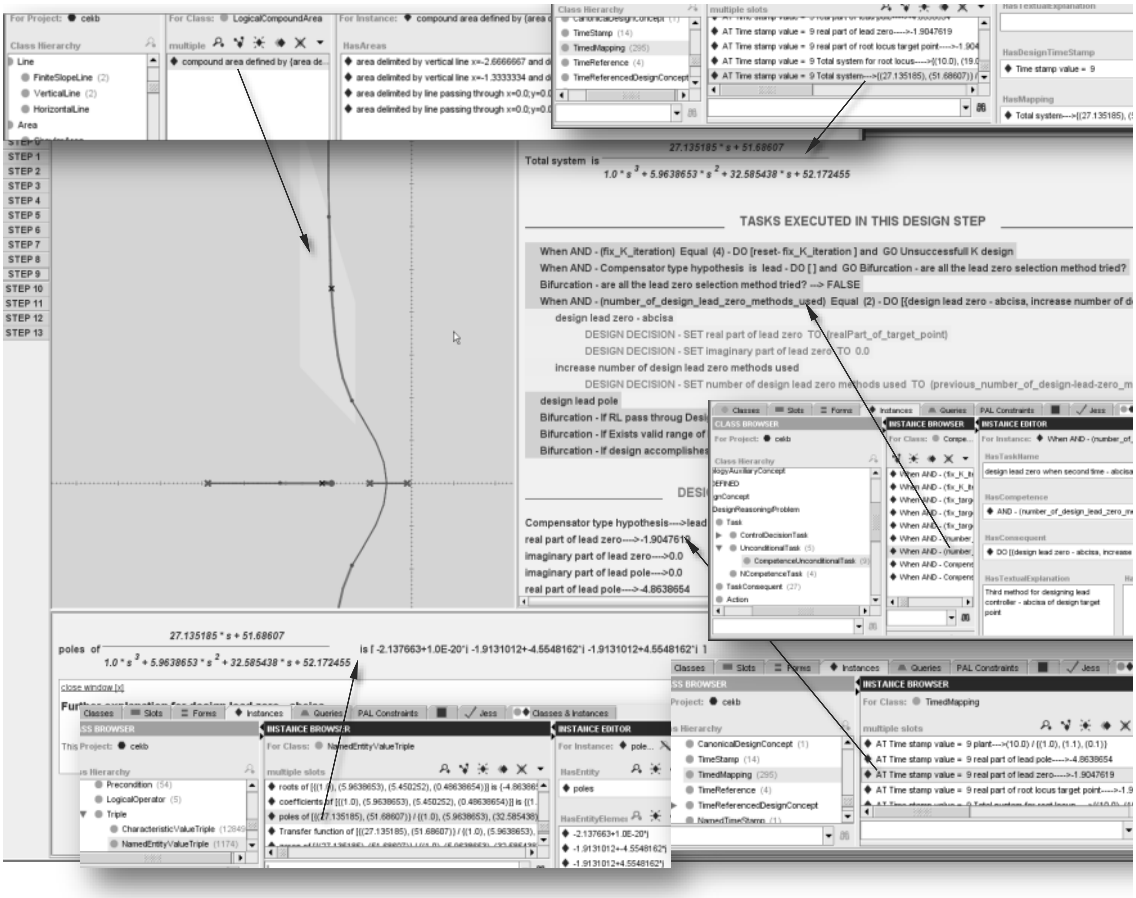
poles of $\frac{27.135185 * s + 51.68607}{1.0 * s^{3} + 5.9638653 * s^{2} + 32.585438 * s + 52.172455}$ is [ -2.137663+1.0E-20*i -1.9131012+-4.5548162*i -1.9131012+4.5548162*i ]

close window [x]

Further explanation for design lead zero - abcisa

This method places the lead zero right below the design point. You can see this example from professor Guy Beale or this exercise

Figure 14. Main window of the application



◆ compound area defined by {area de...

◆ area delimited by vertical line x=-2.6666667 and d

◆ area delimited by vertical line x=-1.3333334 and d

◆ area delimited by line passing through x=0.0;y=0.0

◆ area delimited by line passing through x=0.0;y=0.0

Class Hierarchy: TimeStamp (14), TimedMapping (295), TimeReference (4), TimeReferencedDesignConcept

◆ AT Time stamp value = 9 real part of lead zero---->-1.9047619

◆ AT Time stamp value = 9 real part of root locus target point---->-1.904

◆ AT Time stamp value = 9 Total system for root locus---->[(10.0), (19.0

◆ AT Time stamp value = 9 Total system--->[(27.135185), (51.68607)] /

HasDesignTimeStamp: ◆ Time stamp value = 9

HasMapping: ◆ Total system--->[(27.135185), (5

Total system is $\dfrac{27.135185 * s + 51.68607}{1.0 * s^3 + 5.9638653 * s^2 + 32.585438 * s + 52.172455}$

TASKS EXECUTED IN THIS DESIGN STEP

When AND - (fix_K_iteration) Equal (4) - DO [reset- fix_K_iteration ] and GO Unsuccessfull K design

When AND - Compensator type hypothesis is lead - DO [ ] and GO Bifurcation - are all the lead zero selection method tried?

Bifurcation - are all the lead zero selection method tried? ---> FALSE

When AND - (number_of_design_lead_zero_methods_used) Equal (2) - DO [{design lead zero - abcisa, increase number of d

design lead zero - abcisa

DESIGN DECISION - SET real part of lead zero TO (realPart_of_target_point)

DESIGN DECISION - SET imaginary part of lead zero TO 0.0

increase number of design lead zero methods used

DESIGN DECISION - SET number of design lead zero methods used TO (previous_number_of_design-lead-zero_m

design lead pole

Bifurcation - If RL pass throug Desi

Bifurcation - If Exists valid range of

Bifurcation - If design accomplishes

DESI

Compensator type hypothesis---->lead

real part of lead zero---->-1.9047619

imaginary part of lead zero---->0.0

imaginary part of lead pole---->0.0

real part of lead pole---->-4.8638654

For Project: ● cekb

Class Hierarchy: ologyAuxiliaryConcept, DEFINED, gnConcept, DesignReasoningProblem, Task, ControlDecisionTask, UnconditionalTask (5), CompetenceUnconditionalTask (9), NCompetenceTask (4), TaskConsequent (27), Action

For Class: Compe...

◆ When AND - (fix_K_it

◆ When AND - (fix_K_it

◆ When AND - (fix_targ

◆ When AND - (fix_targ

◆ When AND - (fix_targ

◆ When AND - (number_

◆ When AND - (number_

◆ When AND - Compens

◆ When AND - Compens

For Instance: ◆ When AND - (number_of_

HasTaskName: design lead zero when second time - abcisa

HasCompetence: ◆ AND - (number_of_design_lead_zero_m

HasConsequent: ◆ DO [{design lead zero - abcisa, increase

HasTextualExplanation: Third method for designing lead controller - abcisa of design target point

poles of $\dfrac{27.135185 * s + 51.68607}{1.0 * s^3 + 5.9638653 * s^2 + 32.585438 * s + 52.172455}$ is [ -2.137663+1.0E-20*i -1.9131012+-4.5548162*i -1.9131012+4.5548162*i ]

Fur

This Project: ● cekb

Class Hierarchy: Precondition (54), LogicalOperator (5), Triple, CharacteristicValueTriple (12849), NamedEntityValueTriple (1174)

For Class: NamedEntityValueTriple

◆ roots of [{(1.0), (5.9638653), (5.450252), (0.48638654)}] is {-4.86386

◆ coefficients of [{(1.0), (5.9638653), (5.450252), (0.48638654)}] is {(1.

◆ poles of [{(27.135185), (51.68607)} / {(1.0), (5.9638653), (32.585438)

◆ Transfer function of [{(27.135185), (51.68607)} / {(1.0), (5.9638653),

For Instance: ◆ pole...

HasEntity: ◆ poles

◆ -2.137663+1.0E-20*i

◆ -1.9131012+-4.5548162*i

◆ -1.9131012+4.5548162*i

Project: ● cekb

Class Hierarchy: CanonicalDesignConcept (1), TimeStamp (14), TimedMapping (295), TimeReference (4), TimeReferencedDesignConcept, NamedTimeStamp (1)

For Class: TimedMapping

◆ AT Time stamp value = 9 plant--->(10.0) / {(1.0), (1.1), (0.1)}

◆ AT Time stamp value = 9 real part of lead pole---->-4.8638654

◆ AT Time stamp value = 9 real part of lead zero---->-1.9047619

◆ AT Time stamp value = 9 real part of root locus target point---->-1.9

Figure 15. Main window of the application, showing the relationship to concepts in the ontology.