Design of Fractal-Based Systems within MDA: Platform Independent Modelling

Erika Asnina, Janis Osis, Marite Kirikova

Institute of Applied Computer Systems
Riga Technical University
Meza str. ¼, Riga
LV 1048 Latvia
erika.asnina@cs.rtu.lv
janis.osis@cs.rtu.lv
marite.kirikova@cs.rtu.lv

Abstract: Today's businesses must continuously adapt to changing external environment in accelerated time frames. This requires businesses to decrease costs and time of the very process of change. Therefore, the potentials of changes in software caused by changes in businesses must be addressed during the software modelling. This can be accomplished with a fractal-based architecture. The main properties of fractal-based systems are self-similarity, self-organization, goal-orientation, dynamics, and vitality. This paper discusses how fractal-based nature of the system can be analyzed and modelled in the context of the object-oriented paradigm from the platform-independent viewpoint proposed by Model Driven Architecture (MDA). The main objective of MDA is to make software designs easily portable between different operating platforms, keeping interoperability of the software and reusability of the designs. The suggested approach helps in discovering behavioural and structural scale invariants of fractal-based systems. All scale invariants that must be implemented in fractals are separated in distinct fractal interfaces and fractal classes.

Keywords: Fractal-based information systems, object-oriented modelling, model driven architecture.

1 Introduction

External environment of today's businesses changes rapidly. The businesses must continuously respond to new challenges in speeded up time frames in order to be competitive. The respond requires business changes as well as organizational changes that must be reflected in used information systems. These cause additional time and costs. Thus, the potentials of changes in software caused by changes in businesses and organizations must be addresses during software modelling.

This means that it is necessary to define those places in the system architecture that ought to be able to be changed as fast as possible if corresponding changes in business occurs. For example, let us consider a case, when owners of one company decided to demerge their business into two *fully* independent businesses. They need not only to divide their resources, but they also need to divide information about their business. Thus, they need to split up their information systems with minimal costs.

This can be accomplished with a fractal-based architecture. Here, the collocation "fractal-based systems" is used as generalization for both fractal and multi-fractal systems. So, what is a fractal and a fractal-based information system (IS)? Mandelbrot proposed a definition of the fractal as follows [Fe88]: "a fractal is a shape made of parts similar to the whole in some way". Pieces of a fractal when enlarged are similar to larger pieces or to that of the whole. The definition given by the authors in [Br06] is that "a fractal is a modular and executable component model that can be used with various programming languages to design, implement, deploy and reconfigure various systems and applications, from operating systems to middleware and to graphical user interfaces". In their turn, authors in [TP02] give the definition of a multi-fractal: "A multi-fractal is a scale-free (scale invariant) system, for which the statistical properties of small regions are the same as for the whole system, they are self-similar".

The difference between a fractal and a multi-fractal is that while the first is characterized by a single component, the second requires a collection of scaling exponents (multi-scaling) to be defined completely. Multi-fractals are composed of a hierarchy of multiple fractal sets, each one with its own dimension and transforming differently under changes in scale. Therefore, to completely characterize multi-fractal systems the collection of all dimensions is needed [TP02]. In other words, a fractal-based system can be an organization composed of self-similar entities that can be considered as fractals or fractal-like structures. The term *"fractal-like structure"* is used to stress that often fractals of the smallest scale are not created by self-similar parts. Within this system, decisions are made through cooperation and negotiation with counterpart fractals that each has an equal responsibility. The role of each fractal changes, depending on changing circumstances in the external environment. The main difference between fractal-based systems and "traditional" ones is in their organization, i.e. not all "traditional" systems can be represented by self-similar parts.

According to [OCJ06], the main properties of fractal-based systems are *self-similarity*, *self-organization*, *goal-orientation*, *dynamics*, and *vitality*. In accordance with [RJ04]:

- Self-similarity means that fractals can make the same outputs with the same inputs regardless of their internal structure;
- Self-organization means that fractals are able to apply suitable methods for controlling the process and workflows, and optimizing the composition of fractals in the system;
- Goal-orientation means that fractals perform a goal-formation process to generate their own goal by coordinating processes with the participating fractals and by modifying goals if necessary;

- Dynamics and vitality means that cooperation and coordination between self-organizing fractals are characterized by dynamics and an ability to adapt to a dynamically changing environment.

A fundamental advantage of a fractal-based system is its scale-invariance, i.e. the ability to scale up while retaining similar properties at every level of granularity.

The multi-fractal nature of the information systems can be found in different domains, e.g., product lines, supply chains, enterprise architecture and so on. In many cases, these information systems are implemented on different platforms, and communication between parts of such systems depends on those platforms. For example, dynamically reconfigurable distributed systems depend on a variety of operating systems and middleware. Fractals can be used for building such systems. However, there is not any approach suitable for design fractal-based systems by means of object-oriented system analysis and design (OOAD).

Since in many cases systems are implemented on different platforms, the possibility of platform changes should also be taken into account as early as possible. This problem can be solved by using OMG Model Driven Architecture (MDA) principles, namely, by using *architectural separation of concerns* [MM03]. MDA suggests distinguishing a platform-independent design from a platform-specific design. Thus, once creating a platform-independent design, it is possible to use this single design to generate code suitable for a necessary platform.

The main principles of MDA are described in Section 2. The related works are discussed in Section 3. The suggested object-oriented approach is considered in Section 4 and illustrated by an example in Section 5. Conclusions state the main results and discuss further research.

2. OMG Model Driven Architecture™ in Brief

The main purpose of MDA is separation of the viewpoints in specifications and strengthening the analysis and design role in the project development. The three primary objectives of MDA are portability, interoperability and reusability through *architectural separation of concerns*. Thus, MDA suggests the framework to make software designs easily portable between different operating platforms.

MDA authors foresee three specification viewpoints and their corresponding models:
- a *Computation Independent Model* (CIM) represents system requirements and the way in which the system works within the environment without any details of the system structure and application implementation;
- a *Platform Independent Model* (PIM) describes a system in such abstraction level that renders this model to be suitable for use with different platforms of a similar type; thus a PIM represents the business model to be implemented by an information system;

- a *Platform Specific Model* (PSM) provides a set of technical concepts, representing different kinds of parts that make up a platform and its services to be used by an application, and, hence, changes transferring system functioning from one platform to another.

MDA supports abstraction and refinement in models. Models are obtained by transformations: from PIM to PIM, from PIM to PSM, from PSM to PSM, and from PSM to PIM [MM03]. Transformations from CIM to PIM are usually considered as intellectual and hard to be automated. Despite this, some research in formalization of this area was done [OAG07]. The motto of MDA can be "model once, generate anywhere" (Figure 1).

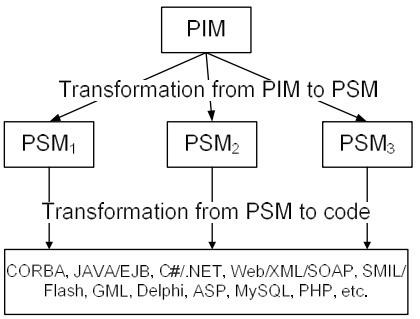

Figure 1: The Principle of Platform Independent Modelling

The PIM usually is composed during the analysis and design stage of the object-oriented software development. A standard related to MDA, Unified Modelling Language (UML), is used for the construction of both platform independent model and platform specific model. The PIM can reflect fractal-based systems independently of the platform-specific information, thus abstracting from the platform-specific details during analysis and general design of the system and making it possible to transform the received PIM models onto different execution platforms. The PIM ignores operating systems, programming languages, hardware, and networking. Therefore, the same PIM can be used even if the platform is changed in the future.

3 Related Work

In order to survive in a competitive and continuously changing environment, organizations must have both flexible structure and processes. To achieve this, architecture of fractals is a powerful means due to fractals' autonomy, flexibility, and self-similarity.

K. Ryu and M. Jung in [RJ04] describe possible fractal-based structures of future organizations including e-biz enterprises, supply chain networks, and manufacturing systems. A component-based architecture for autonomous repair management in distributed systems, called JADE architecture, is realized by means of fractals [Bo05].

The most natural way to design a fractal-based architecture is the use of agents as in the case of networked manufacturing collaboration. Authors described in [Ho05] how fractal theory helped them to transform the complicated manufacturing systems to be transparent and simple. The similarity between the part and the whole illuminated how to construct the proper links between organizing structure and operating style of the multilevel organizations inside an extended enterprise, such as the whole alliance, the individual enterprise, the workshop department. The result is all hierarchical units were able to reconfigure their own architecture by nesting various suballiances and resulted in self-optimization.

R. Murty in [Mu04] illustrated the architecture of JULIET, an application-level load balancing system that guarantees a certain degree of fault tolerance, failovers and replication of data for .NET web services that save state and resume execution using data from saved states. J. Ramanathan in [Ra05] described application of fractals for adaptive complex enterprises by an example of sense-and-respond (S-R) businesses. M. Naaman and A. Zaks in [NZ97] described a blackboard framework that is based on fractals. Besides that they defined kinds of systems that are not suitable for fractal-based architecture, namely, applications that require synchronous work such as complicated 'hand-shaking' with other applications, applications that cannot be decomposed efficiently into small independent module, and small applications, in which parallel work is irrelevant.

There is no approach suitable to *design* of fractal-based systems in OOAD. We think that such an approach should exist, because in that case it will be possible to use opportunities that MDA gives. MDA suggests a new framework that helps not only in designing of fractal-based architectures, but also in reusability of this design and portability of such architecture. Both MDA models and fractal-based models use the *separation of concerns* design principle. Fractal-based design of information systems using the object-oriented paradigm and MDA possibilities is a challenge today.

4 Platform-Independent Object-Oriented Design of Fractal-Based IS

The goal of object-oriented analysis is to determine and specify objects and functionality in the problem domain. In turn, the goal of object-oriented design is to determine and specify software objects and how they communicate in order to satisfy requirements. During object-oriented programming, these objects are implemented using a particular programming language.

For the design of a fractal-based IS the main principles of the object-oriented paradigm are used, namely, encapsulation, inheritance, and polymorphism. In case of fractal-based system design, the most important one is polymorphism. Polymorphism enables systems to be designed so that they can be easily changed, because such a design allows handling objects in equal way.

In essence, the polymorphic structure of the system allows sending the same message to objects of distinct classes and attaining that each object responds accordingly. This makes it possible to realize fractal behaviour (behavioural scale invariants) on different scales.

In fractal-based systems, *scale invariants* are properties of the system that do not change with the change of scale (granularity). Scale invariants can be found in system's behaviour as well as in system's structure, i.e. they represent two fractal's dimensions in the terms of object-oriented paradigm – behaviour and structure. The essential step in the design of such systems is to define those scale invariants. These scale invariants can be specified as contracts by means of interfaces. *Interfaces declare contracts* that can be realized by several classifiers (classes or interfaces). Anything that realizes an interface accepts and agrees to operate according to the contract realized in the interface.

The main idea of the suggested method is to create architecture of the system as polymorphic as possible. The proposed method of construction of platform-independent model for multi-fractal IS is as follows:

- Step 1 – analysis of the system organization and behaviour: 1) define organizational structure of the system; 2) define actors, their goals, use cases that are necessary for those goal achievement, and operation contracts; 3) define a conceptual model;
- Step 2 – definition of fractal scale invariants and organization: 1) define behavioural scale invariants; 2) define structural scale invariants; 3) define fractals' architecture;
- Step 3 – design of fractal classes and interfaces;
- Step 4 – design of fractal classes' behaviour on distinct scales.

4.1 Step 1 – Analysis of the System Organization and Behaviour

The first activity is definition of the organizational structure of the system under consideration. This is necessary to facilitate identification of fractals.

In object-oriented analysis and design, system analysis starts with identification of system's actors and use cases. G. Schneider and J.P. Winters defined use cases as "a behaviour of the system that produces a measurable result of value to an actor" [SW01]. From the user's point of view a use case should represent a complete task that can be performed in a relatively short run. In essence, use cases are goal-oriented. One or several use cases can be used in order to achieve the same goal. Actors are roles of other computer systems in an organization or humans. Actors activate execution of use cases. For identification of actors and use cases standard activities suggested in [La05] can be used, e.g. assisting questions.

The second activity is definition of the behaviour of the system under consideration. After identification of *actors*, their *functional goals* are determined. In accordance with the identified functional goals, *use cases* needed for goal achievement are identified. Use cases are specified by their scenarios that are main flow and alternate flow (in case of errors or exceptions) description of each use case. Here it is important to identify the so called *operation contracts* for each actor request.

Each contract describes one operation with the indicated name, parameter list, *references to other use case that use the same operation, pre and post conditions*. During this activity, it is important to define common actors' goals and operation contracts.

After identification of use cases, the third activity is identification of concepts in the system's description. Concepts are ideas, things, or objects. Concepts and their relationships should be defined in the initial model of concepts.

Completing these iterative activities, the refined use case model and conceptual model as well as inputs and outputs necessary to goal achievement on distinct organizational levels (scales) are specified.

4.2 Step 2 – Definition of Fractal Scale Invariants and Their Architecture

As mentioned before, scale invariants are properties of the system that do not change with the change of scale. The second step is to identify these scale invariants. There are behavioural and structural scale invariants.

To **define behavioural scale invariants**, interactions in each identified use case should be analyzed and may be specified using UML interaction diagrams – sequence or collaboration diagrams. During the analysis of interactions, it is important to define messages that are sent and received by objects on different scales while acting towards achievement of the same goal.

For each use case, it is necessary to define input and output parameters of messages and their types. Two of fractal properties are self-similarity and goal-orientation. As previously mentioned, self-similarity can be reflected as the same inputs and the same outputs and different realizations of the inner organization of fractals on the different scales. Therefore, it is possible to define such similarity in behaviour by detecting activities of that kind on different scales. Goal-orientation of use cases corresponds to goal-orientation of fractals.

Besides, in case of fractal-based systems, a fractal on a higher scale needs to activate execution of similar functionality on lower scales. If fractals on all scales realize self-similar functionalities (activities), a fractal at the higher level then needs only to activate similar functionalities (activities) to corresponding collection of fractals on the lower scale in order to achieve the goal. *Self-similar activities* are those, which are used to achieve the same goal, have the same inputs and outputs, but differ in their operation. Self-similar activities are a behavioural view of fractals.

To **define structural scale invariants**, it is necessary to analyze data about the structure that are independent of scale of consideration. In multi-fractal systems, structural data that are related to the fractal structure can be candidates to structural scale invariants. Those candidates that must be presented on all scales of the multi-fractal system are separated as attributes of the fractal class (classes). For example, it can be assessment of usefulness of each fractal for performing some task. Then this assessment must be presented in each fractal, therefore it is their common structural characteristic.

Another fractal's property is self-organization that helps systems to provide dynamics and vitality. In order to support this property, a designer needs to **define architecture (organization) of fractals** by mapping it to the organizational structure of the system in the real world.

The way how the real-world system is organized will determine communication among fractals in the information system. The arrangement of fractals can be represented as a *collection of references* to fractals of the related scales with which the fractal communicates, for example: 1) If this is a hierarchical structure, then this collection should capture information about fractals of the related lower scale and/or of the related higher scale; or 2) If this is a "snowflake" structure, then this collection should capture information about the fractal's direct neighbourhoods. By changing this collection of references during the execution time it is possible to reorganize the system.

4.3 Step 3 – Design of Fractal Classes and Interfaces

The results of two previous steps are: 1) identified scale invariants, and 2) fractals' architecture. Since the object-oriented approach supports inheritance and platform-independent modelling supports implementation of the defined behaviour and structure at different platforms, it is useful to define all the shared aspects related to the fractals in a distinct class – a fractal class – in accordance with Rule 1 and Rule 2. A *fractal class* is a class that specifies similar structure and functionality of fractals of the same kind. Note that a keyword is *similarity* not equality. This means that all particularities of the system that must be implemented in the system's fractals are specified and can be inherited and overridden if necessary.

Rule 1: Self-similar activities are to be transformed to operations of fractal classes.

Rule 2: Structural data that relates to the fractal structure and needs to be presented in the system independently of the scale are to be transformed to attributes of fractal classes.

A useful aspect of such a design is as follows. If implementation of some method differs on some scale and this situation cannot be avoided using generalization, then implementation of this method can be left empty. In case of multi-fractals, there is more than one fractal class.

Thus, all particularities that shall be implemented in software fractals are defined at this stage. However, thinking about transformation from model to code, it is important to note that not all programming languages support multiple inheritances. Therefore, the next activity is to specify *fractal interfaces* that declare fractal classes' contracts.

A *fractal interface* is an interface that should be implemented by a class that specifies a fractal. The interface specifies those scale invariants that are mandatory for specific kinds of fractals. If fractal classes on all scales implement the same interface, then a fractal class on the larger scale only needs to apply the interface method to all class objects in the fractal collection in order to achieve a goal.

Rule 3: All behavioural and structural invariants may be specified in fractal interfaces that must be realized by fractal classes.

4.4 Step 4 – Design of Fractal Classes' Behaviour on Distinct Scales

As a result of previous steps, fractal interfaces and fractal classes together with specifications of operations are defined. The specifications of operations are defined in platform-independent model, for instance, using UML activity diagrams or using OCL (Object Constraint Language) expressions.

During Step 4 classes (and their behaviour), which correspond to the organizational scales and inherit fractal classes, must be defined. The important point here is to make sure that all differences in operation implementations would be taken into account.

According to MDA, this platform-independent design should be transformed to the platform-specific one. There are developed several tools for transformations from platform-independent models to platform specific ones, where platforms are such as J2EE and .NET. Therefore, the way of implementation of polymorphism should be chosen taking into account which platforms will be used.

Polymorphism can be implemented using interfaces or inheritance mechanism. For instance, polymorphism in Java and C# is implemented using interfaces and single inheritance, while C++ supports multiple inheritances. Thus, operation overloading or overriding can be used to realize the polymorphism.

Overriding occurs when a subclass method has the same name, same return type, and same argument list as the super-class method. Within the proposed method the overriding is mainly used, because self-similarity of fractals expresses in the same inputs and outputs for all fractals. Besides that in case of multi-fractals, we recommend to use interface mechanisms, since it can be transformed, for example, to Java and C++, without any changes in the *platform-independent* model.

5 An Application Example

Let us consider a small example that illustrates the key moments of the suggested approach. The system under consideration is a faculty titled DITF. Faculty's organizational units are two institutes (titled LDI and ITI), each of which has sub-units – departments (Figure 2). LDI departments are Department LDK and Department STPK; one of ITI's departments is MIK. We consider one goal that this system needs to achieve, namely, evaluation of the research of each structural unit in the context of a faculty, an institute, and a department. In this section, the words "structural unit" and "division" are applied interchangeably.

Let us assume that an actor on the level of the faculty is DITF secretary, actors on the level of this faculty's institutes are LDI secretary and ITI secretary, and actors on the level of departments are LDK secretary, STPK secretary, and MIK secretary.

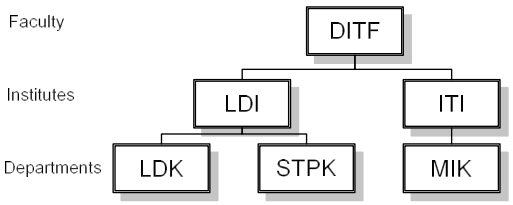

Figure 2: DITF faculty organizational units

The goal of the DITF secretary is to create a report of evaluation of faculty's research, the goal of the LDI secretary and ITI secretary is to create a report of evaluation of their institute's research and the goal of the LDK secretary, STPK secretary and MIK secretary is to create a report of evaluation of their department's research.

As shown in Figure 3, this goal can be satisfied by the realization of the corresponding use case for each organizational level. However, realization of the use case of a higher level includes realization of use cases of all related lower levels.

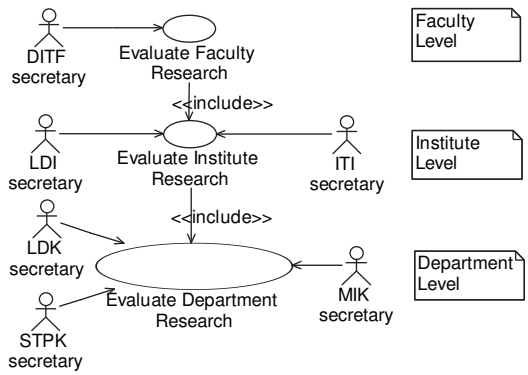

Figure 3: The use case model for faculty's function of the evaluation of research in structural units

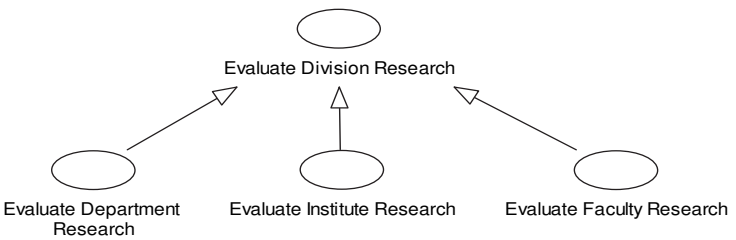

Figure 4: The more general use case of the evaluation of research of a structural unit

This means that each of these use cases is a specialization of a more general case, i.e., a specialization of the use case that specifies evaluation of research of sub-units of each organizational level in accordance with the actor's level (Figure 4). A part of the initial conceptual model for our example is illustrated in Figure 5.

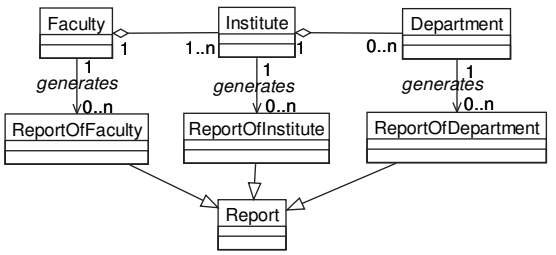

Figure 5: A part of the initial conceptual model

Specification of the more general use case must be defined by analysis of the use cases at all the defined levels of the faculty. Figure 6 illustrates that evaluation of research (operation 1- *evaluateDepartmentResearch(dep:Department):ReportOfDepartment*) of institute's departments consists of handling information about scientific publications (operation 1.1– *getScientificPublication():Boolean*) and visited conferences of the department's staff (operation 1.2– *getVisitedConferences():Boolean*); the return value is an object of class *ReportOfDepartment*.

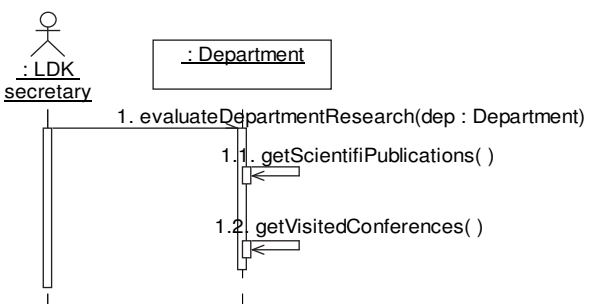

Figure 6: The scenario of the use case "Evaluate Department Research"

In turn, Figure 7 demonstrates interactions to the evaluation of research of faculty's institutes (operation 1– *evaluateInstituteResearch(inst:Institute):ReportOfInstitute*), which includes the evaluation of all the institute's departments (operation 1.1 – *evaluateDepartmentResearch*) and then correction of the received results in accordance with the work specifics (operation 1.2– *correctEvaluationResult*). For example, the staff of different departments can write scientific publications as co-authors. This means that the sum of the count of the scientific publications on the department level will differ from the real number of the scientific publications on the institute level.

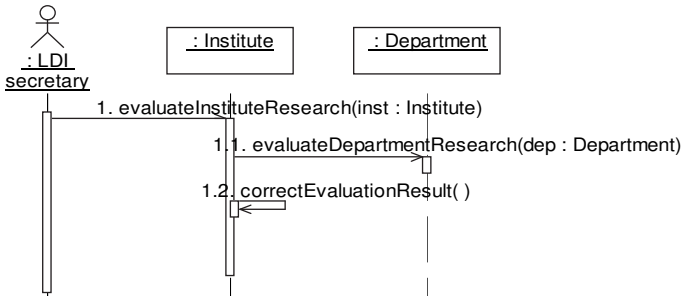

Figure 7: The scenario of the use case "Evaluate Institute Research"

Figure 8 illustrates the same goal satisfaction on the faculty level. This goal is satisfied by summing up the information about evaluation of the research of all the faculty's institutes (operation 1– *evaluateFacultyResearch(faculty:Faculty):ReportOfFaculty*) and by correcting the acquired information (operation 1.2– *correctEvaluationResult*) in accordance with the faculty's specifics, for example, as in the case of the institute's level.

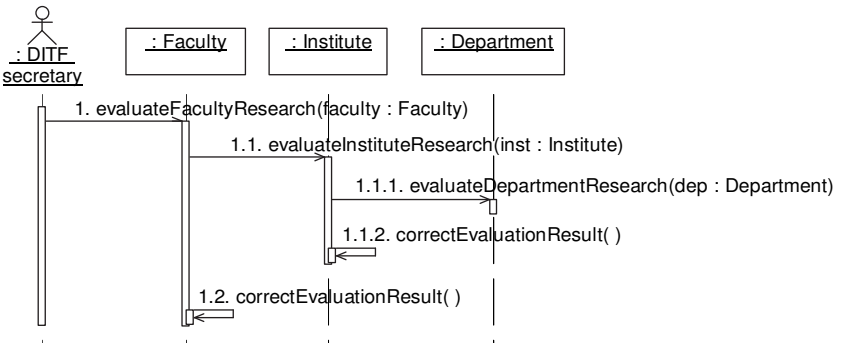

Figure 8: The scenario of the use case "Evaluate Faculty Research"

Thus, analysing scenarios of the use cases on each scale, similarities can be found. As figures 6, 7, and 8 illustrate the input to the evaluation of research is a unit (a faculty, an institute, or a department) and the output is this unit's report, although achievement of the goal differs. These similarities can be separated into the general use case scenario as shown in Figure 9. As previously mentioned, one of the main principles of the object-oriented paradigm is polymorphism. This means that the scenario specified by the general use case is realized in special use cases in a more specific way. As shown in Figure 9, in order to specify execution of a general scenario of the use cases, the class *Division* is specified. This is a super-class for classes *Faculty*, *Institute*, and *Department*. It should contain only fractal-specific details, which shall be specified in classes realizing the scenarios mentioned.

This means that the class *Division* should contain the information about its sub-units (subdivisions) and general operation realizations, i.e., it is a *fractal super-class*.

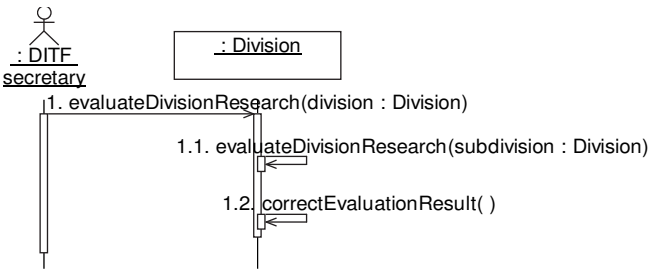

Figure 9: The possible scenario of the general use case "Evaluate Division Research"

Hence, the class Division contains the operation *evaluateDivisionResearch(division:Division):Report*. In our example, this operation is called recursively. This means that if it is activated by the DITF secretary, then this operation is executed recursively indirectly starting from the object of the class *Faculty* to the objects of the class *Institute* to the objects of the class *Department*. The return object is an object of the general class *Report*; hence objects of its child classes can be referenced.

In order to provide that all child classes of the fractal class would be able to realize all the necessary operations and attributes, it is useful to separate them into a *fractal interface*. Figure 10 shows the fractal interface class *IDivision* that is realized by the fractal class *Division* (shadowed elements in the diagram).

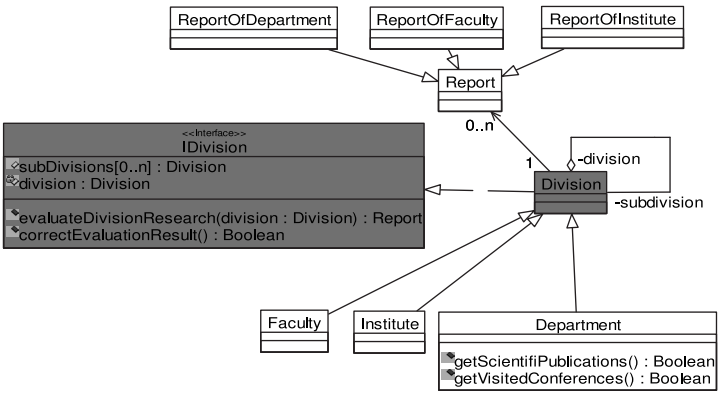

Figure 10: A part of the class diagram with fractal specific design

The interface *IDivision* specifies that a class that realizes this interface must contain information about its sub-units and must evaluate unit's research and correct the received result if necessary. Hence, classes *Faculty*, *Institute*, and *Department* inherit this responsibility and may realize this responsibility in their specific way. One more way how to design can be by using only fractal interfaces. In this case, classes *Faculty*, *Institute* and *Department* should realize the interface *IDivision* directly. But this way is less useful. All other goals must be analyzed and modelled in the same way.

6 Conclusions

This paper describes how fractals and fractal-like structures can be designed by means of object-oriented paradigm. However, rapid changes in the environment can require not only fast changes of business logic, but also fast changes of a platform. Thus, the problem of design reuse also has been taken into account. This problem is suggested to be solved within MDA that is widely illustrated in scientific researches and industrial reports.

The suggested approach helps in discovering scale invariants of fractal-based systems in two dimensions, namely, in both behaviour and structure. All scale invariants that must be implemented as fractal properties are considered as fractal responsibilities. In order to avoid multiple inheritances, these responsibilities are declared using fractal interfaces. The shared behaviour and structure are specified by using fractal classes. Using this approach the modification of systems design is less complicated and time-consuming than in conventional approaches, because changes made in the particular place, namely, in a fractal interface or a fractal class, will impact all related fractals. If fractal responsibilities were not separated, then each change in shared behaviour or structure of the system would require modification in many places throughout the design.

Platform-independence of this design enables its transformation to the platform-specific design – each fractal can be implemented on a distinct platform using automated MDA transformations. Thus, the reuse of the design can be supported by that methodology.

Further research is related to applying the proposed method to integration of enterprise applications.

Bibliography

[Bo05] Bouchenak, S. et. al.: Architecture-Based Autonomous Repair Management: An Application to J2EE Clusters. In: Proceedings of the 2005 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05), IEEE, 2005.

[Br06] Bruneton, E. et. al.: The FRACTAL Component Model and Its Support in Java. In: Experiences with Auto-adaptive and Reconfigurable Systems, Volume 36, Issue 11-12, 2006; pp. 1257 – 1284.

[Fe88] Feder, J.: Fractals, Plenum, New York, 1988.

[Ho05] Hongzhao, D. et. al.: A Novel Approach of Networked Manufacturing Collaboration: Fractal Web-Based Extended Enterprise. In: International Journal of Advanced Manufacturing Technology, Volume 26(11-12), Springer, 2005; pp. 1436-1442.

[La05] Larman, Cr.: Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design and Iterative Development, Prentice Hall PTR, 3rd edition, 2005.

[MM03] Miller, J., Mukerji, J. (eds.): MDA Guide Version 1.0.1, OMG, 2003. http://www.omg.org

[Mu04] Murty, R.: JULIET: A Distributed Fault Tolerant Load Balancer for .NET Web Services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04), IEEE Computer Society, Washington, DC, 2004: pp. 778.

[NZ97] Naaman, M., Zaks, A.: Fractal Blackboard Framework. In: Proceedings of the 8th Israeli Conference on Computer-Based Systems and Software Engineering (ICCSSE'97), 1997; p. 23.

[OCJ06] Oh, S., Cha, Y., Jung, M.: Fractal Goal Model for the Fractal-Based SCM. In: Proceedings of the 7th Asia Pacific Industrial Engineering and Management Systems Conference 2006, Thailand, 2006; pp. 423-429.

[OAG07] Osis, J., Asnina, E., Grave, A.: MDA Oriented Computation Independent Modelling of the Problem Domain. In: Proceedings of the Second International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007), Barcelona, Spain. INSTICC, 2007; pp. 66-71.

[Ra05] Ramanathan, J.: Fractal Architecture for the Adaptive Complex Enterprise. In: Communications of the ACM, Vol. 48, No. 5, May 2005; pp. 51-57.

[RJ04] Ryu, K., Jung, M. Chapter XV: Fractal Approach to Managing Intelligent Enterprises. In: Creating Knowledge Based Organizations, IGI Publishing, 2004.

[SW01] Schneider, G., Winters, J.P.: Applying Use Cases: A Practical Guide, The Addison-Wesley, 2nd edition, 2001.

[TP02] Turiel, A., Perez-Vicente, C. J.: Universality Class of Multi-fractal Systems. In: Europhysics Letters, EDP Science, 2002.