ISSN 1407-7493 COMPUTER SCIENCE DATORZINĀTNE 2008-34

APPLIED COMPUTER SYSTEMS
LIETIŠĶĀS DATORSISTĒMAS

ANALYSIS OF MULTIFRACTAL SYSTEM PROPERTIES IN OBJECT-ORIENTED SOFTWARE DEVELOPMENT

MULTIFRAKTĀĻU SISTĒMU ĪPAŠĪBU ANALĪZE OBJEKTORIENTĒTĀ PROGRAMMATŪRAS IZSTRĀDĒ

Erika Asnina, *Riga Technical University,*
Faculty of Computer Science and Information Technology,
Institute of Applied Computer Systems,
Meza str. 1/3, Riga, LV 1048, Latvia, Lecturer, Dr.sc.ing., erika.asnina@cs.rtu.lv

Janis Osis, *Riga Technical University,*
Faculty of Computer Science and Information Technology,
Institute of Applied Computer Systems,
Meza str. 1/3, Riga, LV-1048, Latvia, Professor, Dr.habil.sc.ing., phone: +371 7089523, janis.osis@egle.cs.rtu.lv

Fractal dimension, UML, platform independent model, MDA

1. Introduction

What is a multifractal information system? There many distinct definitions. For example, the following one – "a multifractal 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" [1]. The difference between a fractal and a multifractal is that while the first is characterized by a single component, the second requires a collection of scaling exponents (multiscaling) to be defined completely.

Another definition 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 [2]. According to that the fractal component model heavily uses the *separation of concerns* design principle.

Multifractals are composed by a hierarchy of multiple fractal sets, each one with its own dimension and transforming differently under changes in scale. Therefore, to completely characterize multifractal systems the collection of all dimensions is needed [1].

The main properties of fractal-based systems: self-similarity, self-organization, goal-orientation, dynamics, and vitality [3]. In accordance with [4]:

- 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 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 the dynamically changing environment.

Multifractal 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, multifractal information systems are implemented on different platforms and communication between fractals in such system depends on those platforms. For example, fractals can be used for building dynamically reconfigurable distributed systems that depends on variety of operating systems and middleware. However, this problem can be solved by using OMG Model Driven Architechture (MDA) principles [5].

2. Fractal Properties Reflection by MDA Models

The main purpose of MDA is separation of viewpoints in specifications and strengthening the analysis and design role in the project development. 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 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 similar type;
- 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, does change transferring system functioning from one platform to another.

MDA supports abstraction and refinement in models. Models are obtained by transformations: PIM-to-PIM, PIM-to-PSM, PSM-to-PSM, and PSM-to-PIM. Each transformation must be recorded in a record of transformation [5].

The PIM usually is composed during the analysis and design stage of object-oriented software development. Within MDA, Unified Modeling Language (UML) notation is used for 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 received PIM models onto different execution platforms.

3. Object-Oriented Approach to Platform-Independent Design of Multifractal Systems

In multifractal systems, scale invariants are system's properties that do not change with the change of a scale. The main step in the design of multifractal systems is to define those scale invariants.

The proposed platform-independent object-oriented method consists of the following steps:

- STEP1 – Analyze system behavior
  - Define structure of the system under consideration;
  - Define actors, their goals and common goals among them;
  - Define use cases that are necessary for those goal achievement;
- STEP 2 – Define fractal interfaces;
  - Define dynamic scale invariants;
  - Define structural scale invariants;
  - Define the organizational structure of fractals;
- STEP 3- Define fractal classes that realize fractal interfaces;
- STEP 4- Define classes that inherit fractal classes.

3.1. Step 1 "Analyze system behavior"

In object-oriented analysis and design (OOAD), system analysis starts with identification of system's actors and use cases. G. Schneider and J.P. Winters define use cases as "a behavior of the system that produces a measurable result of value to an actor" [6]. From 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 human roles or computer systems' roles in an organizational unit. Actors activate execution of use cases. For identification of actors and use cases standard advices suggested in [7] can be used.

After identification of *actors*, *functional goals* are identified for each actor. And in accordance with them, *use cases* needed for goal achievement are identified. Use cases are specified by their scenarios that are descriptions of both main flow and alternate flow (in case of errors or exceptions) of each use case.

After identification of use cases, concepts are identified in the system's description. Concepts and their relationships are defined in initial model of concepts.

3.2. Step 2 "Define fractal interfaces"

Dynamic scale invariants

Each use case should be analyzed using UML interaction diagrams – sequence or collaboration diagrams. During analysis of each use case, it is necessary to define messages that are sent and received by objects at different scales during achievement of the same goal.

For each use case, it is necessary to define input and output parameters. 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 but different realization of the inner organization of fractals at the different scales. Therefore, it is possible to define such similarity in behavior by founding activities of that kind at different scales.

Besides that in case of multifractal systems, a fractal at the higher scale needs to activate execution of similar functionality at the lower scales. If fractals at all scales realize the same contract, i.e., the same interface, a fractal at the higher level then need only to apply necessary methods of this interface to corresponding collection of fractals at the lower scale in order to achieve the goal.

Rule 1: The similar activities are a dynamic view of fractals. These activities are to be transformed to methods of the fractal interface.

Fractals can have a number of interfaces, because multifractal systems can consist of fractals of different types.

Structural scale invariants

In multifractal systems, structural data that are related to the *fractal structure* can be candidates to the structural scale invariants. Those of candidates that must be presented at all scales of the multifractal system are separated as attributes of the fractal interface.

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

The interface that reflects responsibilities of fractals must provide attributes that are dedicated to capturing information about the fractal's scale.

Define the organizational structure of fractals

Another property of multifractal systems is possibility to self-organization that help systems in providing dynamics and vitality. In order to implement this property in the multifractal systems, an analyst needs to correspond to the organizational structure of the system in the real world. That how the system is organized determines attributes the fractal interface will capture. The attribute that capture information about fractal's scale can be represented as a collection of fractals of the related scales, for example:

1. If it is a hierarchical structure, then it is necessary to capture information about fractals of the related higher scale and of the related lower scale;
2. If it is a "snowflake" structure, then it is necessary to capture information about the fractal's direct neighborhoods.

By changing this collection during the execution time, it is possible to reorganize the system.

3.3. Step 3 "Define fractal classes that realize fractal interfaces"

The result of two previous steps is obtained fractal interfaces. This means that all particularities of the system that must be implemented in system's fractals are specified.

Since the object-oriented approach supports inheritance, but platform-independent modeling supports implementation of the defined behavior and structure at different platforms, it is useful to define all shared aspects related to fractals as a distinct class – *a fractal class*. Therefore, the next step is to define classes that implement the defined interfaces and that can be inherited by other classes, which define fractals at different scales.

A useful aspect of design of this kind is as follows. If implementation of some method differs at some scale and this situation cannot be avoided using generalization, then implementation of this method can be empty.

3.4. Step 4 "Define classes that inherit fractal classes"

After previous steps, fractal interfaces and fractal classes together with diagrams of method implementations are defined.

STEP 4 is to be used to describe classes, which correspond to the organizational scales and inherit fractal classes. The important point here is to provide that all differences in method implementations would be taken into account.

4. A Case Study

Let us consider a small example that illustrates the head moments of the suggested approach. Let us assume that the system under consideration is a university. University's organizational divisions are faculties, each of which has subdivisions – institutes, in turn each of which has subdivisions – departments (Figure 1).

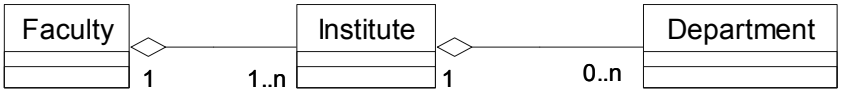

Figure 1. University's organizational divisions

We consider one function of this system, namely, evaluation of the scientific work of each division in the context of a faculty, an institute, and a department. Let us assume that an actor on the level of 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 and STPK secretary.

The goal of the DITF secretary is to evaluate faculty's research, the goal of the LDI secretary and ITI secretary is to evaluate their institute's research and the goal of the LDK secretary and STPK secretary is to evaluate their department's research.

As shown in Figure 2, this goal can be satisfied by the realization of the corresponding use case for each organizational level. However, realization of the use case of the higher level includes realization of the use case of all of the corresponding lower levels. 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 subdivisions of each organizational level in accordance with the actor's level (Figure 3).

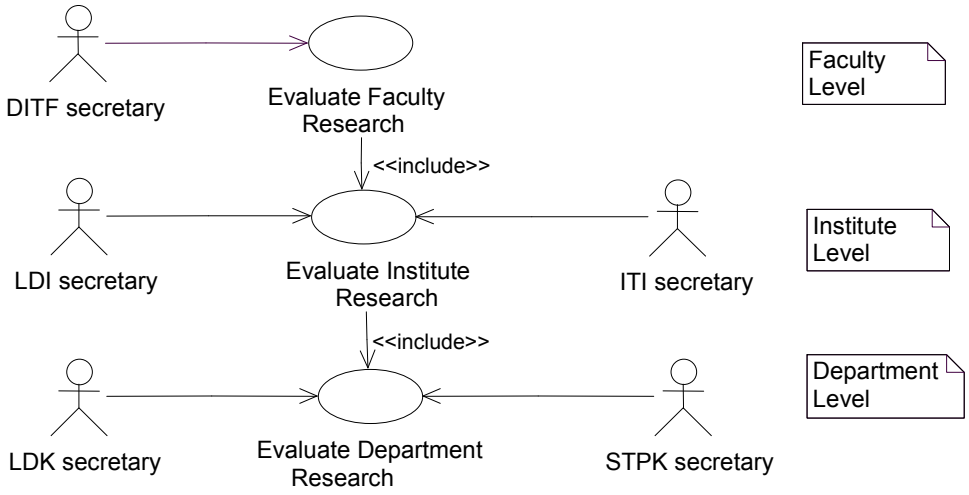

Figure 2. The use case model for university's function of the evaluation of divisions' research

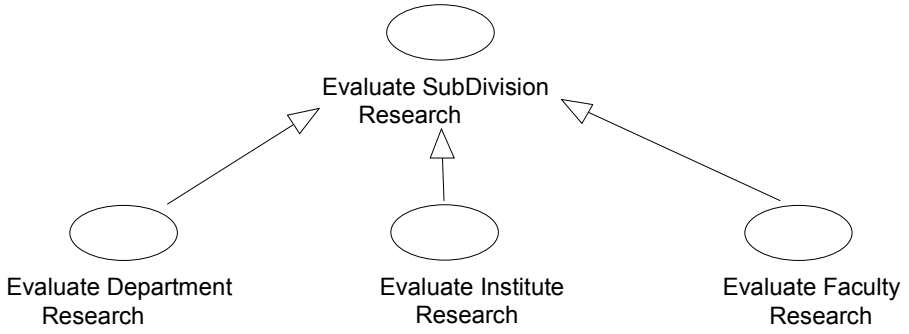

Figure 3. The more general use case of the evaluation of research of a division

But the specification of the more general use case must be defined by analysis of the use cases at all the defined levels of the university. Figure 4 illustrates that evaluation of the research of institute's department consists of handling information about scientific publications and visited conferences of the department's staff.

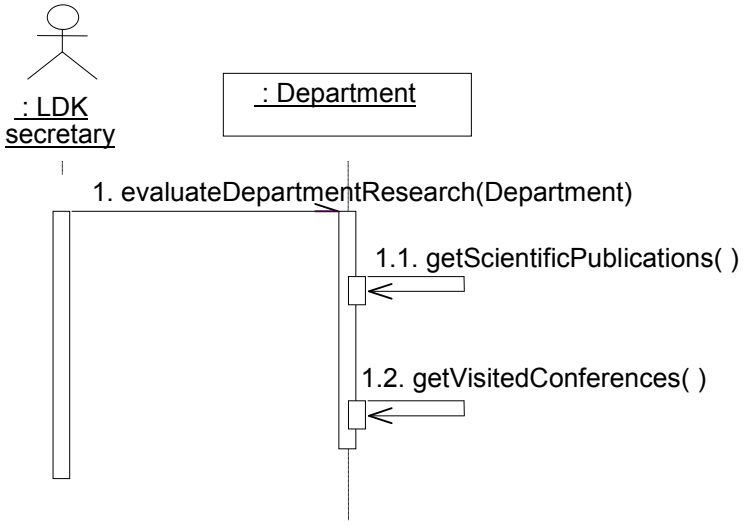

Figure 4. The scenario of the use case "Evaluate Department Research"

In turn, Figure 5 demonstrates the scenario of the evaluation of the research of faculty's institute that contains the evaluation of all the institute's departments and then correction of the received results in accordance with the research specifics. 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 count of the scientific publications on the institute level.

Figure 6 illustrates the same goal satisfaction on the faculty level. This goal is satisfied summing up the information about evaluation of the research of all the faculty's institutes and correcting them in accordance with the faculty's specifics like, for example, in case of the institute's level.

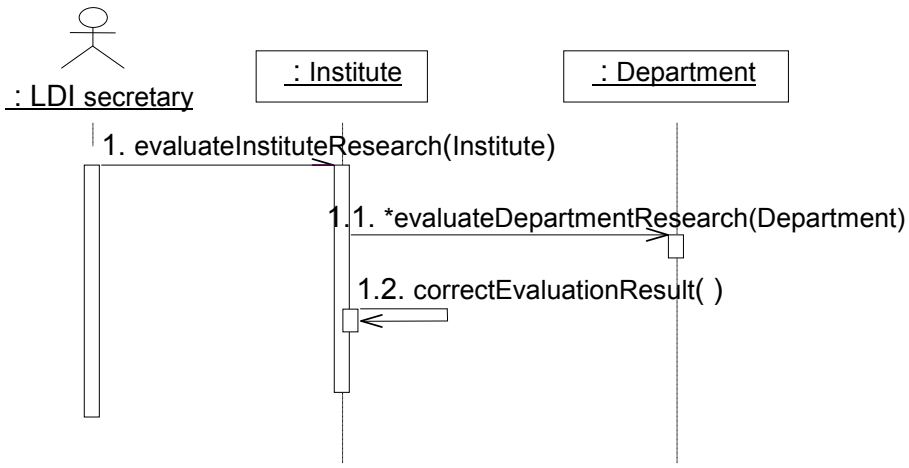

Figure 5. The scenario of the use case "Evaluate Institute Research"

Thus, analyzing scenarios of the use cases for each level, the similarities can be found. These similarities can be separated into the general use case scenario as shown in Figure 7. 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 the special use cases in the more specific way.

As shown in Figure 7, in order to specify execution of such a general scenario of the use cases, the class Division is specified. This is a class that 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 subdivisions and general operation realizations, i.e., it is a *fractal superclass*.

In order to provide that all of the subclasses of this superclass would be able to realize all the necessary operations and attributes, it is useful to separate them into the *fractal interface*. Figure 8 shows the interface class IDivision that is realized by the class Division.

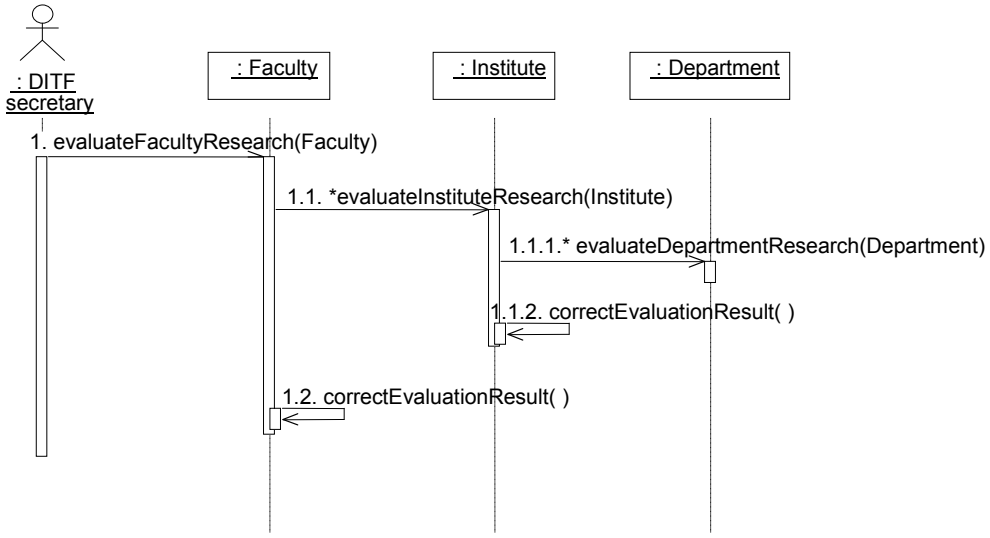

Figure 6. The scenario of the use case "Evaluate Faculty Research"

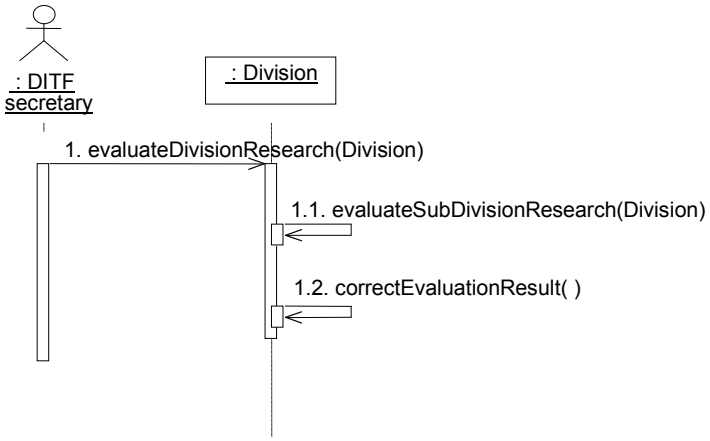

Figure 7. The possible scenario of the general use case "Evaluate SubDivision Research"

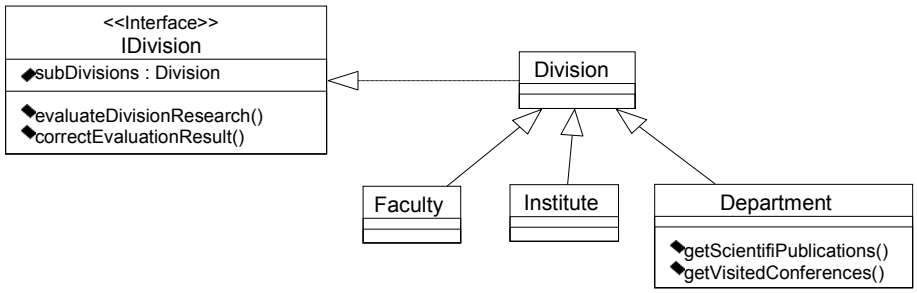

Figure 8. A part of the class diagram with fractal specific design

The interface IDivision specifies that a class that realizes this interface must contain information about its subdivisions and must evaluate its 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.

Another way to design can be only using of fractal interfaces. In this case, classes Faculty, Institute and Department should realize the interface IDivision directly. But this way is less useful.

5. Conclusions

This paper discusses the platform-independent design of fractal-based systems using the object-oriented approach. Related works in this field mostly suppose using components or agents to reflect fractal properties of systems. The suggested approach demonstrates how multifractal systems can be designed wide using such object-oriented principles as polymorphism and inheritance.

The suggested approach helps in discovering dynamic and structural scale invariants of the multifractal systems. All shared scale invariants to be implemented in fractals are separated in distinct fractal interfaces and fractal classes. This means that a system can be easily modified if necessary. Such a design supports all properties of multifractal systems, namely, self-similarity, self-organization, goal-orientation, dynamics, and vitality.

The further research is related to applying the proposed method to design of fractal-based enterprise systems.

References

1. Turiel A., Perez-Vicente C. J. *"Universality class of multifractal systems"* // In: Europhysics Lettters, EDP Science, 2002, 7 pages.
2. E. Bruneton, Coupaye T., Leclercq M., Quéma V., Stefani J.B. *"The FRACTAL component model and its support in Java"* // In: Experiences with Auto-adaptive and Reconfigurable Systems, Volume 36, Issue 11-12 , 2006, p. 1257 – 1284
3. Oh S., Cha Y., and 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, p. 423-429
4. Ryu K., Jung M. *"Chapter XV. Fractal Approach to Managing Intelligent Enterprises" //* Gupta , Sharma (eds.) Creating Knowledge Based Organizations, IGI Publishing, 2004, 373 pages
5. Miller J., Mukerji J. (eds.). *"OMG: MDA Guide Version 1.0.1"*, 2003 // Internet. – http://www.omg.org
6. Schneider G., Winters J.P. *"Applying Use Cases, 2nd ed. A practical Guide" //* The Addison-Wesley, 2001, 245 pages.
7. Larman Cr. *"Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development"*, 3rd ed. // Prentice Hall PTR, 2005, 703 pages

Asņina Ē., Osis J. Multifraktāļu sistēmu īpašību analīze objektorientetā programmatūras izstrādē
Pašlaik biznesam nepārtraukti ir jāadaptējas ārējiem apstākļiem saspringtajās laika robežās. Tas prasa biznesam pāriet no stratēģijas „kā samazināt pārmaiņas" uz stratēģijām „kā pieņemt pārmaiņas un izmainījušos apstākļus". Lai apietu šos ierobežojumus IT sistēmās ir jābūt realizētai atbildei pašam izmaiņu procesam. Tas var būt paveikts ar fraktāļu arhitektūru, kura nodrošina vienu kopēju pilnīgu vidi adaptīvajām sistēmām. Uz fraktāļiem bāzēto sistēmu galvenās īpašības ir pašlīdzība, pašorganizācija, orientācija uz mērķi, dinamiskums un vitalitāte. Šajā rakstā ir apskatīts kā sistēmu fraktālais raksturs var būt analizēts un modelēts objektorientētās paradigmas kontekstā no platformneatkarīga skatupunkta (kuru piedāvā MDA). Šajā rakstā ir apskatīta fraktāļu dimensiju attēlošana vienotas modelēšanas valodas (UML) notācijā. Piedāvātā metode palīdz dināmisku un struktūras invariantu atrašanā uz fraktāļiem pamatotās sistēmās. Visi koplietojamie invarianti, kuriem jābūt realizētiem fraktāļos, ir izdalīti atsevišķos fraktāļu interfeisos un fraktāļu klasēs. Tas nozīmē, ka sistēma var būt vieglāk modificēta, ja nepieciešams.

Asnina E., Osis J. Analysis of multifractal system properties in object-oriented software development
Today's businesses must continuously adapt to external conditions in accelerated time frames. This requires businesses to shift from strategies that eliminate variation to those that embrace variation and changing conditions. For IT to overcome the limitations, the deeper issue of modeling the very process of change itself must be addressed. This can be accomplished with a fractal architecture that provides a single shared complete framework for the adaptive systems. The main properties of fractal-based systems: self-similarity, self-organization, goal-orientation, dynamics, and vitality. This paper discusses how fractal nature of the system can be analyzed and modeled in the context of object-oriented paradigm from the platform-independent viewpoint (proposed by MDA). This paper considers representation of the fractal dimensions by means of the UML (Unified Modeling Language) diagrams. The suggested approach helps in discovering dynamic and structural scale invariants of the fractal-based systems. All shared scale invariants that must be implemented in fractals are separated in distinct fractal interfaces and fractal classes. This means that system can be easily modified if necessary.

Аснина Э., Осис Я. Анализ свойств мультифрактальных систем при объектно-ориетированной разработке программного обеспечения
Современный бизнес должен уметь непрерывно и в сжатые сроки адаптироваться к внешним условиям. Это требует от бизнеса смены стратегии «уменьшающей перемены» на стратегию «принимающую как перемены, так и изменившиеся условия». Для этого при разработке ИТ систем необходимо решить проблему как облегчить в них процесс изменения структуры и функциональности. Решение этой проблемы может быть достигнуто при помощи фрактальной архитектуры, обеспечивающей одну полную, совместно используемую среду для адаптирующихся систем. Основные свойства систем, базирующихся на фракталах, это подобие, самоорганизация, ориентация на цель, динамичность и жизнеспособность. В данной статье рассматривается способ анализа и моделирования фрактального характера системы в контексте объекто-ориентированной парадигмы с независимой от платформы точки зрения (предложенной в MDA). В данной статье рассматривается представление фрактальных дименсий средствами унифицированного языка моделирования (Unified Modeling Language). Предложенный метод помогает обнаружить динамические и структурные инварианты в системах, базирующихся на фракталах. Все совместно используемые инварианты, которые должны быть реализованы во фракталах, выделены в отдельные фрактальные интерфейсы и классы. Таким образом облегчается процесс модифицирование системы.