МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 01.00.00.000 ПЗ

Група ІПм-22-1

Бабій Давид

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Бабій Давид Віталійович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та методології комплексної інфраструктури побудови інформаційних систем

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Бабій Д.В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Юрчишин Володимир Миколайович, к.т.н., професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

В.о. завідувача кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

доц.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2023 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Бабію Давиду Віталійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі, методи та методології комплексної інфраструктури побудови інформаційних систем ”

керівник проекту (роботи) Юрчишин Володимир Миколайович, д.т.н., професор

затверджені наказом закладу вищої освіти від “ 18 ” грудня 2023 р. № 738/7

2. Строк подання студентом проекту (роботи) 15 січня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови програмного забезпечення інформаційних систем

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз структури та концепцій системних фреймворків різного рівня
2. Структуризація доменів теоретичної та практичної активностей
3. Імплементація системних моделей рівня комплексних фреймворків
4. Застосування методів та моделей для комплексних фреймворків інформаційних систем

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Концептуальна модель, використана в аналізі
2. Структура телекомунікаційної мережі
3. Вимоги безпеки для вузлів RBS
4. Діаграма потоку інформації: конструкція багатофункціонального модуля
5. Початкова версія концептуальної моделі

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Нормоконтроль	доц., к.т.н. Вовк Р.Б.	
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі дослідження	01.10.2023	виконано
2	Аналіз структури та концепцій системних фреймворків різного рівня	25.10.2023	виконано
3	Структуризація доменів теоретичної та практичної активностей	10.11.2023	виконано
4	Імплементація системних моделей рівня комплексних фреймворків	22.11.2023	виконано
5	Застосування методів та моделей для комплексних фреймворків інформаційних систем	01.12.2023	виконано
6	Реалізація функціональності запропонованої інформаційної технології	15.12.2023	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.01.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 84 с., 27 рис., 1 табл., 48 джерел.

Тема: Моделі, методи та методології комплексної інфраструктури побудови інформаційних систем

Об'єкт дослідження: структура та концепції системних фреймворків різного рівня, з виділеними рівнями імплементації.

Мета роботи: опис та дослідження моделі та структури системних фреймворків на макро та мікрорівнях програмної імплементації проекту.

Предмет дослідження: процес імплементації системних моделей рівня комплексних фреймворків, для побудови доменів координації концептуальної моделі процесів і переходів розробки рішення.

Результати дослідження:

В роботі проведена розробка шаблону процесу завдання, що детально описує кроки для виконання конкретного завдання процесу та модель етапу процесу, що відображає кроки, які необхідно виконати для імплементації стадії процесу.

Висновок

Досліджено методології розробки програмного забезпечення, як способу взаємодії активних суб'єктів (так званих ролей в процесі), спрямованих на виконання певних операцій, які називаються видами діяльності, на наборі програмних артефактів, які називаються продуктами роботи.

КОМПЛЕКСНИЙ ФРЕЙМВОРК, КОНЦЕПТУАЛЬНА МОДЕЛЬ, ТЕЛЕКОМУНІКАЦІЙНА МЕРЕЖА, ДОМЕН КООРДИНАЦІЇ, СТРАТЕГІЯ, ІНФОРМАЦІЙНА СИСТЕМА

ABSTRACT

Master Thesis: 84 pp., 27 fig., 1 tab., 48 sources.

Thesis Subject: Models, methods and methodologies of the complex infrastructure of building information systems

Object of research: the structure and concepts of system frameworks of different levels, with selected levels of implementation..

Research goal: description and research of the model and structure of system frameworks at the macro and micro levels of the software implementation of the project.

Subject of research: the process of implementing system models at the level of complex frameworks, for the construction of domains of coordination of the conceptual model of processes and transitions of solution development.

The results:

The paper developed a task process template that describes in detail the steps to perform a specific process task and a process stage model that reflects the steps that must be performed to implement a process stage.

Conclusion

The methodology of software development was studied as a way of interaction of active subjects (so-called roles in the process) aimed at performing certain operations, called types of activities, on a set of software artefacts, called work products.

COMPLEX FRAMEWORK, CONCEPTUAL MODEL, TELECOMMUNICATION NETWORK, COORDINATION DOMAIN, STRATEGY, INFORMATION SYSTEM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SMS - Small Message Service

GT - Grounded Theory

RBS - Radio Base Stations

RNC - Radio Network Controllers

MGW - Media Gateway

RTES - Real-Time Embedded Systems

CRUD - Create, Read, Update, Delete

IFD - Information Flow Diagrams

SBDM - Specification Based Data Model

PDM - Product Data Management

ISA - Information System Architectures

HDE - Hardware Design Environment

DIM - Design Information Interchange Model

SDM - Software Development Methodolog

SA- Structured Analysis

OMG - Object Management Group

OOA - Object-Oriented Analysis

ADFD - Action Data Flow Diagrams

OODLE - Object Oriented Design LanguagE

PDC - Problem Domain Component

DMC - Data Management Component

RDD - Responsibility-Driven Design

OMT - Object Modeling Technique

DM - Dynamic Model

ВСТУП

Актуальність дослідження.

Незважаючи на те, що методології розробки програмного забезпечення та відповідні фреймворки страждають від різних проблеми, вони постійно розвиваються, а дослідження спрямовані на їх удосконалення є постійним еволюційним процесом. Стан проблеми чітко демонструє потенціал для покращення шляхом вирішення цих питань. Існує мотивація для розробки методологій використовуючи висновки, отримані з UML і довгої історії об'єктно-орієнтованих методологій у створенні основи для розробки програмного забезпечення. Як загальну характеристику таких методологій можна виділити наступне:

1. Компактність: ядро, що розширюється, є кращим, чим сутність, що налаштовується, або загальна структура зі складними параметрами.

2. Розширюваність: з чітко визначеними механізмами розширення та вказівками.

3. Простежуваність до вимог: усі артефакти мають бути простежуваними до вимог.

4. Послідовність: вироблені артефакти не повинні суперечити один одному; як альтернатива, повинні бути механізми для виявлення невідповідностей.

5. Можливість тестування артефактів із самого початку: це дозволить розробити інструменти для перевірки артефактів.

6. Відчутність артефактів: створення виконуваного матеріалу якомога раніше в життєвому циклі.

7. Видима раціональність: за кожним завданням має бути очевидна раціональність, порядок, у якому виконуються завдання, і незаперечне використання кожного створеного артефакту; розробники повинні мати можливість бачити цю логіку, справді відчуваючи, що будь-яке відхилення поставить під загрозу цілі проекту.

Усвідомлюючи необхідність і потенціал для подальшого вдосконалення в цій галузі, важливо зазначити, що відносно довга історія розвитку методології є багатим джерелом досвіду, який потрібно вивчити. У кожній методології є особливості, які слід використовувати і пастки, яких слід уникати, багато з яких є прямими чи непрямими наслідками методу застосування, що використовувалися при розробці методології або деталі, пов’язані з розробкою.

Отже, вибір правильного методу для розробки цільової методології є задачею надзвичайної важливості. Об'єктно-орієнтовані методології можна класифікувати відповідно до обставин, що призвели до їх розвитку, включаючи підхід і методи застосування.

Велика кількість методологій була розроблена досвідченими практиками або науковцями, які пробували нові ідеї та підходи у своїй повсякденній інженерній практиці, що зрештою призвело до створення методології, що пропонує абсолютно новий підхід, і позначає переломний крок в історії розробки програмного забезпечення. Такі методології діють, починаючи власні нитки еволюції. Методології, що належать до першого покоління, є революційними по своїй суті.

Мета і задачі дослідження полягають в застосуванні методів та моделей комплексних фреймворків інформаційних систем в рамках традиційної та альтернативної архітектури з метою формування рівня корпоративних підключеннь.

Метою магістерської роботи є опис моделі та структури системних фреймворків на макро та мікрорівнях програмної імплементації проекту.

Об’єктом дослідження є структура та концепції системних фреймворків різного рівня, з виділеними рівнями імплементації.

Предметом дослідження є процес імплементації системних моделей рівня комплексних фреймворків, для побудови доменів координації концептуальної моделі процесів і переходів розробки рішення.

Методи дослідження. В магістерській роботі використані методи та моделі побудови процесів переходу, в рамках процесу розробки програмного забезпечення у відповідному комплексному фреймворку розробки; методи та моделі синтаксичного та семантичного аналізу програмного коду в процесі його компіляції.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- Виконати аналіз структури та концепцій системних фреймворків різного рівня.
- Структуризувати домени теоретичної та практичної активностей.
- Виконати імплементацію системних моделей рівня комплексних фреймворків

Наукова новизна роботи полягає в розробці типів шаблонів процесів: шаблону процесу завдання, що детально описує кроки для виконання конкретного завдання процесу; модель етапу процесу, що відображає кроки, які необхідно виконати для імплементації стадії процесу; шаблон етапу процесу – зазвичай складається з кількох завдань шаблонів процесу; шаблон фазового процесу.

Практичне значення одержаних результатів полягає в розробці методології розробки програмного забезпечення, як способу взаємодії активних суб'єктів (так званих ролей в процесі), спрямованих на виконання певних операцій, які називаються видами діяльності, на наборі програмних артефактів, які називаються продуктами роботи, доки такі програмні артефакти не будуть доведені до чітко визначеного стану та не оголошені завершеними.

Структура та обсяг роботи. Магістерська робота складається з вступу, трьох розділів, висновків та списку використаних джерел (48 найменувань). Загальний обсяг магістерської роботи становить 84 сторінки, що містить 27 рисунків і 1 таблицю.

РОЗДІЛ 1. АНАЛІЗ СТРУКТУРИ ТА КОНЦЕПЦІЙ СИСТЕМНИХ ФРЕЙМВОРКІВ РІЗНОГО РІВНЯ

1.1. Аналіз предметної області дослідження

Сукупність телекомунікаційних систем назвали найбільшою машиною світу. Він складається з мережі взаємодіючих вузлів, кожен з яких виконує певну функцію в загальній мережі, як-от відстеження положення мобільного телефону, забезпечення функцій заряджання, надання даних про власника мобільного тощо (рис. 1.1).

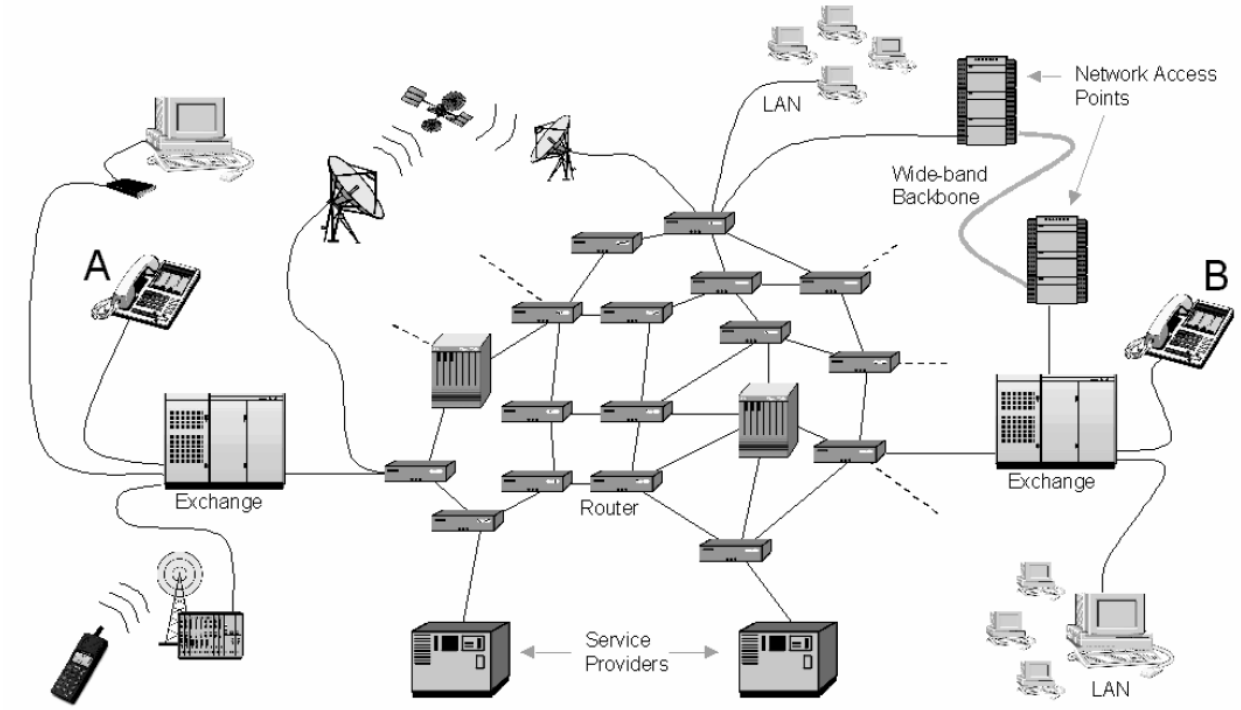

Рис. 1.1. Приклад телекомунікаційної мережі

Взаємодії, які відбуваються під час телефонної розмови, здійснюються відповідно до складних протоколів із високими вимогами до продуктивності, якості мовлення тощо. Інші взаємодії відбуваються всередині вузла, наприклад частина програмного забезпечення взаємодіє з частиною обладнання, наприклад процесор. Крім того, існує взаємодія з різними

учасниками, такими як ті, хто виконує завдання з обслуговування та спостереження.

Багато різних технологій використовуються в телекомунікаційній системі. Для підключення мобільного телефону до базових станцій використовується радіотехнологія. Базові станції – це радіоантени, видимі вздовж доріг, на дахах будинків тощо. Програмна технологія використовується для різних цілей керування та для надання послуг, як от SMS (Small Message Service). Апаратна технологія використовується в інтегральних схемах мобільного зв'язку, процесорах і комутаційному обладнанні тощо. Оптична технологія використовується в кабелях, що з'єднують вузли, і в лазерах, що випромінюють світлові імпульси, що передають інформацію. Механічна технологія використовується в шафах і магазинах, де розміщено обладнання. Хімічна технологія використовується в конструкціях мобільних корпусів, в друкованих платах, в проводах і т.д.

Мережа знаходиться в стані постійної еволюції. Нові вузли додаються, а застарілі видаляються. Впроваджуються нові види послуг і збільшується ємність мережі. Наприклад, впровадження 5-го покоління мобільних систем, яке зараз триває, дозволить передавати відео на мобільний і з мобільного надзвичайно швидко. Весь час потрібно контролювати мережу. Наприклад, якщо вузол перестав працювати або десь обірвався кабель, потрібно виконати переналаштування. На додаток до цього існує спадок існуючого обладнання та мереж, який необхідно враховувати під час внесення змін у мережу. Кілька великих клієнтів, головним чином телекомунікаційні департаменти, які раніше контролювалися урядом, інвестували значні суми у свої мережі протягом тривалого часу. Їхнє обладнання, термін служби якого становить кілька десятиліть, має жити пліч-о-пліч з новими технологіями та новими мережами, якими володіють інші компанії.

Характер телекомунікаційних систем передбачає, що розробка цих систем також є складним завданням. Сам розмір завдання розробки є величезним з точки зору підпродуктів, розробників, зв'язків,

розповсюдження тощо. Проект із розробки 5-го покоління мобільних систем може тривати більше року на виконання та залучати кілька тисяч учасників на різних ролях по всьому світу. Ця робота виконується на сайтах розробки, які мають певну автономію, щоб структурувати власний спосіб роботи. Під час розробки необхідно узгодити кілька сотень етапів розробки, які можна вважати міні-проектами.

Щоб проілюструвати цю складність, на рисунку 1.2 показано залежності в одному вузлі мобільної мережі (вузол Mobile Switching Center).

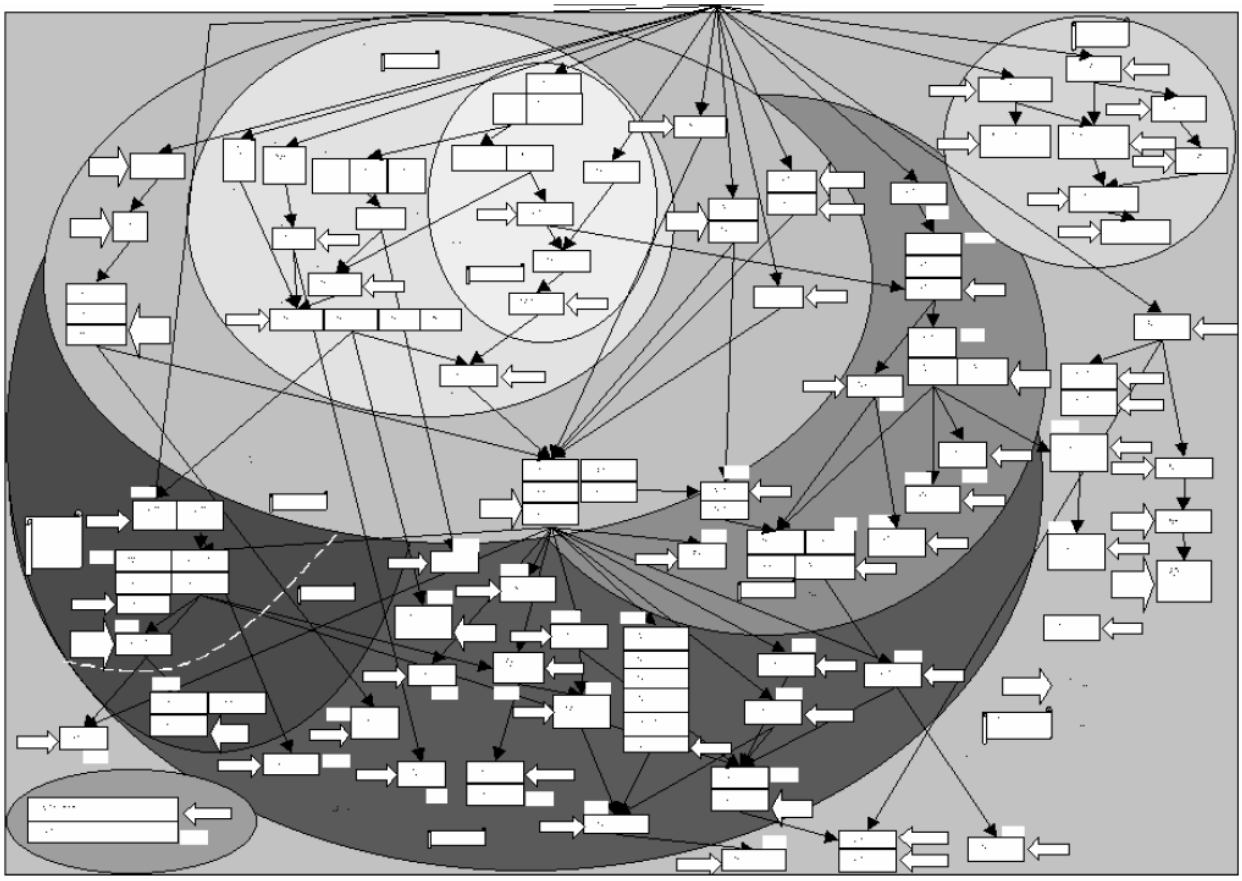

Рис. 1.2. Залежності між завданнями розробки у вузлі мережі

Кожне біле поле означає внесок у загальну функціональність. Стрілки між прямокутниками вказують на залежності між завданнями розробки, починаючи з найпростішого завдання у верхній частині малюнка. Білі стрілки позначають зміст і дату конкретної інтеграції та перевірки ряду завдань розробки. «Бульбашки» представляють основні послуги у вузлі, як-от

реєстрація місцезнаходження мобільного телефону, дзвінок на мобільний телефон, відповідь на мобільний дзвінок тощо. У більшості випадків ці функції надаються програмним забезпеченням. Фігура, яка називається «анатомією», в принципі ілюструє інтеграцію великого проекту розробки програмного забезпечення, в якому кількість рядків коду може бути порядку мільйонів.

Крім того, оскільки існує низка різних технологій, необхідні дуже різнорідні навички. Наприклад, завдання проектування антени мобільного телефону дуже відрізняється від написання програмного забезпечення для керування комутатором. Тим не менш, ці та багато інших навичок потрібні для впровадження всієї системи.

Оскільки проекти розподіляються між розробницькими організаціями по всьому світу, спосіб, у який здійснюється розробка, відрізняється від сайту до сайту. Місцеві традиції та культура еволюціонували з часом. Це стосується процесів, інструментів, різних способів мислення, умов місцевих ринків тощо. Тим не менш, у проекті має бути певна спільність, щоб зібрати всі частини разом.

На телекомунікаційному ринку існує жорстка конкуренція серед постачальників систем. Крім того, цей ринок швидко змінюється, в основному завдяки двом факторам: дерегуляції з появою багатьох нових операторів, що призводить до посилення конкуренції, і поширенню нових технологій, таких як мобільний зв’язок, інтелектуальні мережі, Інтернет тощо. як наслідок, постачальники повинні бути більш реактивними та гнучкими до потреб ринку, що означає коротші терміни виконання та гнучкий процес управління вимогами.

Крім того, наразі оператори вклали значні кошти в ліцензії та мережі для мобільних систем нового покоління. Це мало величезний вплив на плани розвитку, організаційну структуру та працівників організації.

Для того, щоб відповідати змінам на ринках і технологіях, проводяться часті реорганізації, включаючи аутсорсинг компанії та створення нових

партнерств. Наприклад, більшість попереднього внутрішнього виробництва апаратного обладнання, такого як друковані плати, за кілька років було передано зовнішнім компаніям, що спеціалізуються на цьому типі бізнесу.

Усі обставини, розглянуті вище, створюють високу напругу на можливості акторів, які працюють у цій сфері. Зокрема, це стосується однієї можливості: координації завдання розробки. Узгоджена дія передбачає, що актори сприймають «координацію» подібним чином.

Однак обставини, описані вище, надзвичайно ускладнюють розвиток і збереження спільного значення координації. Це ще складніше, оскільки організації, що займаються розробкою продуктів, звикли зосереджуватися на технічних артефактах і, як правило, ігнорують соціальні аспекти, такі як те, як досягти спільного значення.

Через турбулентність і складні умови на ринку необхідно проводити більш-менш постійне перепланування, що також впливає на координацію. Якщо координація з якихось причин не вдається, це загрожує всьому проекту та може бути втрачено бізнес-можливості. Продукт може бути затриманий або взагалі не доставлений клієнту, доставка може містити помилкові або відсутні деталі тощо.

1.2. Дослідницькі процеси та стратегії

Роль теорій і стратегій для отримання очікуваного внеску знань у цьому дослідженні можна описати наступним чином.

Перше дослідницьке запитання RQ1 «Як змінився Framework у практиці розробки телекомунікаційних систем ?» забезпечує реконструктивні / історичні знання. Жодної конкретної ролі теорії у зв’язку з цим дослідницьким питанням не можна знайти. Стратегія надання знань є реконструктивною.

Друге дослідницьке запитання RQ2 «Які загальні наслідки втручання застосування Frameworky в практику розробки телекомунікаційних систем ?» надає пояснювальні знання щодо наслідків використання фреймворків.

У цьому сенсі його можна розглядати як підхід до створення теорії, хоча й не спрямований на створення повноцінної теорії. Точка зору теорії домену діяльності використовується лише як сенсибілізуючий засіб при аналізі даних. Процес дослідження цього дослідницького питання можна охарактеризувати як переважно індуктивний.

Третє дослідницьке питання RQ3 «Які елементи в практиці розробки телекомунікаційних систем сприяють успішним результатам проектів розробки ? » надає пояснювальні знання. Тут початкове проникнення теорії є низьким і його найкраще можна охарактеризувати як обрамлення дослідницького інтересу шляхом визначення одиниці аналізу як «проекту».

Результатом RQ3 стане глибше проникнення теорії, що дасть уявлення про елементи, що сприяють успішним результатам. Створення теорії відбувається за допомогою кількох поєднань між теорією та збором даних. Таким чином, це можна охарактеризувати як процес викрадення.

Що стосується четвертого дослідницького питання RQ4 «Який вплив на координацію втручання Frameworky у практику розробки телекомунікаційних систем ? », то на початку існує високе проникнення теорії, що складається з теорії сфери діяльності, яка, у свою чергу, виведена з точки зору практики.

Ця перспектива сприяла як розробці структури, так і збору та аналізу даних. Таким чином, це питання дослідження можна розглядати як підтвердження конструктивної сили цієї теоретичної перспективи щодо координації розвитку складних систем. Це означає, що дослідницький процес, пов'язаний з цим дослідницьким питанням, можна охарактеризувати як процес перевірки теорії, тобто дедуктивний.

На рисунку 1.3 проілюстровано різні дослідницькі процеси, які беруть участь у дослідженні.

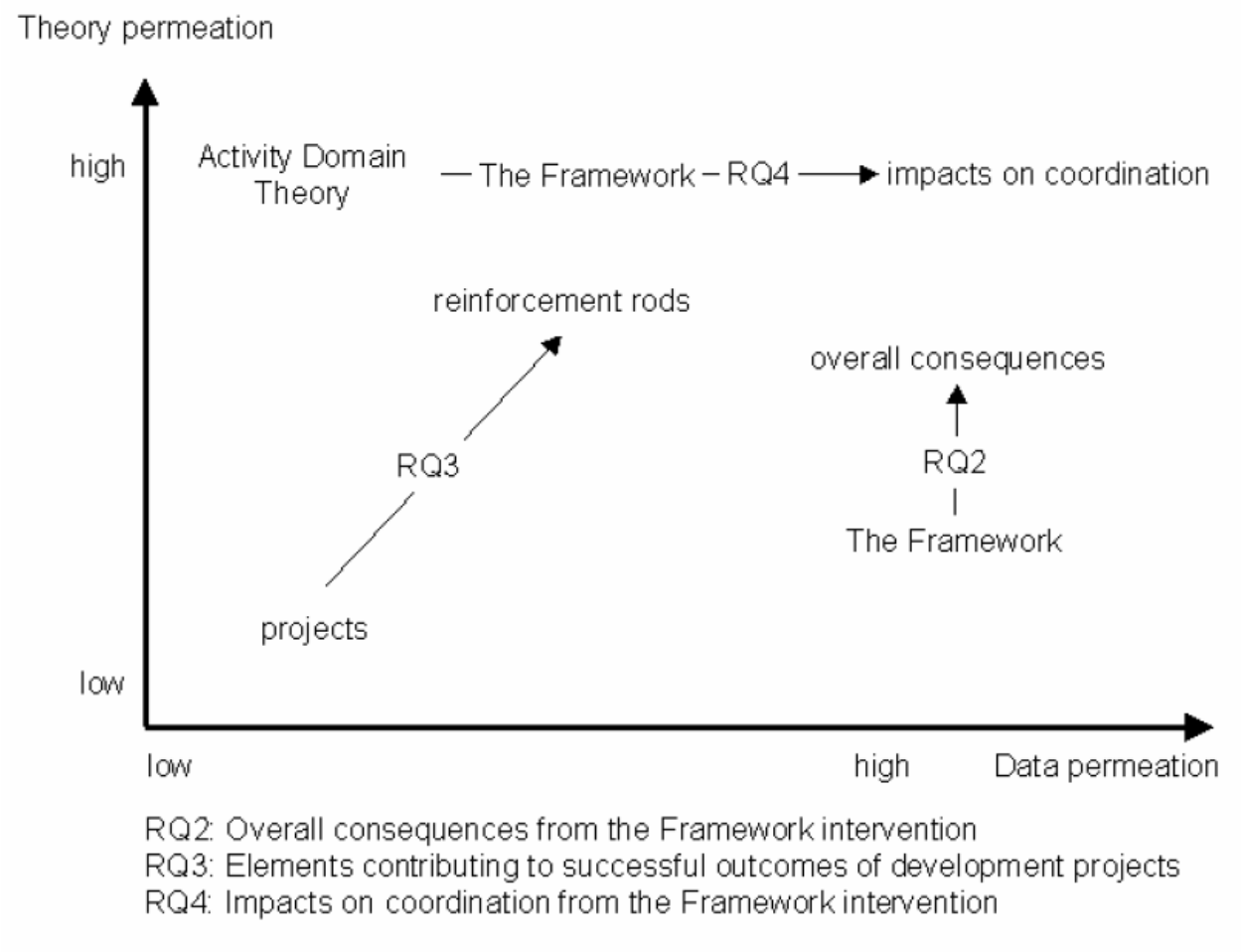

Рис. 1.3. Дослідницькі процеси в роботі

Метод, який використовується для дослідницьких питань RQ2, RQ3 і RQ4, в основному є модифікованим підходом Grounded Theory (GT). Причина, чому не використано GT як є, полягає в тому, що я вважаю його нечітким щодо використання теорії для збору емпіричних даних. Здається, ідеальним є те, що дослідник вступає в дослідницьку сферу без будь-яких упереджень щодо неї, намагаючись уникнути упередженості. « ...фактичні властивості випливають із даних... ».

Щодо дослідницького питання RQ3, був проведений попередній аналіз у якому було визначено п’ять стрижнів: участь, фокус, орієнтація, федерація та ядро. Тут більш суворо дотримувалися методу GT. Однак розробка властивостей і розмірів кожної категорії була спрощена. Це означає, що теоретичного насичення кожної категорії не було досягнуто, що, у свою чергу, означає, що більш ретельний аналіз може виявити інші категорії або

краще артикуляцію існуючих. Однак, оскільки RQ3 не є головною темою дослідження, я стверджую, що таке спрощення підходу GT є виправданим. Результат цього аналізу був використаний для формулювання визначення координації. Крім того, знайдені категорії були додатково обґрунтовані даними проектів, у яких використовувався Framework. Таким чином, я стверджую, що знайдені категорії є добре обґрунтованими, хоча й не вичерпними.

Для дослідницьких запитань RQ2 і RQ4 в якості теоретичного керівництва використовувалася теорія сфери діяльності. Крім того, сама структура була використана в аналітичній роботі, що означає, що я визначив область діяльності для аналізу. Використана концептуальна модель показана на рисунку 1.4.

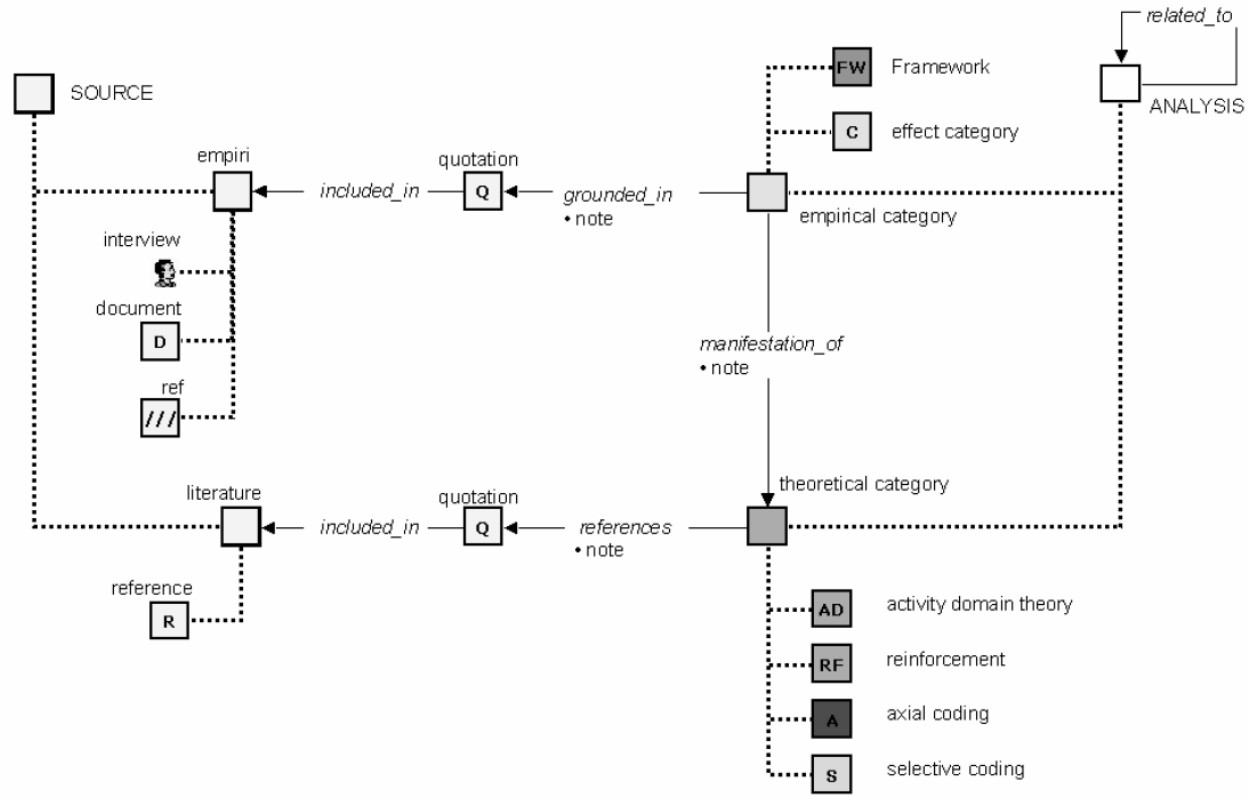

Рис. 1.4. Концептуальна модель, використана в аналізі

Аналіз слідує за діалектичним процесом виробництва знань. У цьому процесі аналіз переходить від конкретного й локалізованого до абстрактного

й загального, а потім повертається до конкретного й ситуаційного. Це означає, що існує два типи категорій аналізу: емпіричний і теоретичний. Емпіричні категорії намагаються охопити багату та локальну сукупність, до якої я також включаю елементи Framework для RQ2 та RQ4. Емпірична категорія, наприклад «продукт», сприймається як контекст, а не ізольована категорія. Щоб якомога повніше охарактеризувати цей контекст, розглядаються характерні дії, що відносяться до цієї категорії. Ці дії описані у формі GT: передумови, дії та наслідки. Комбінована статична та динамічна характеристика відповідає процесу кодування GT: «Кодування процесу відбувається одночасно з кодуванням властивостей, розмірів і зв'язків між поняттями».

1.3. Опис моделі та структури системних фреймворків телекомунікаційних систем

Як тип вбудованих систем реального часу телекомунікаційні системи мають специфічні характеристики, які висувають певні вимоги та пріоритетність одних вимог над іншими. Ці системи мають бути захищеними, добре розподіленими, мати динамічний характер, вимагати величезної потужності обробки та високої доступності (доступність 99,999%, яку іноді називають п'ятьма дев'ятками) і мають бути масштабованими. Розподіл у цих системах можна розглядати з двох точок зору:

- розподіл всередині одного вузла (наприклад, використання багатоядерних рішень і розподіл програмних функцій між різними процесорами),

географічний розподіл вузлів у різних регіонах і зв'язок між ними.

Загальна структура вузлів телекомунікаційної мережі показана на рисунку 1.5. Як показано на цьому рисунку, мережа складається з багатьох різних типів вузлів, таких як базові радіостанції (RBS - Radio Base Stations), контролери радіомереж (RNC - Radio Network Controllers), медіашлюзи

(MGW - Media Gateway) та інші, які охоплюють велику географічну територію та пов’язані різними видами ліній.

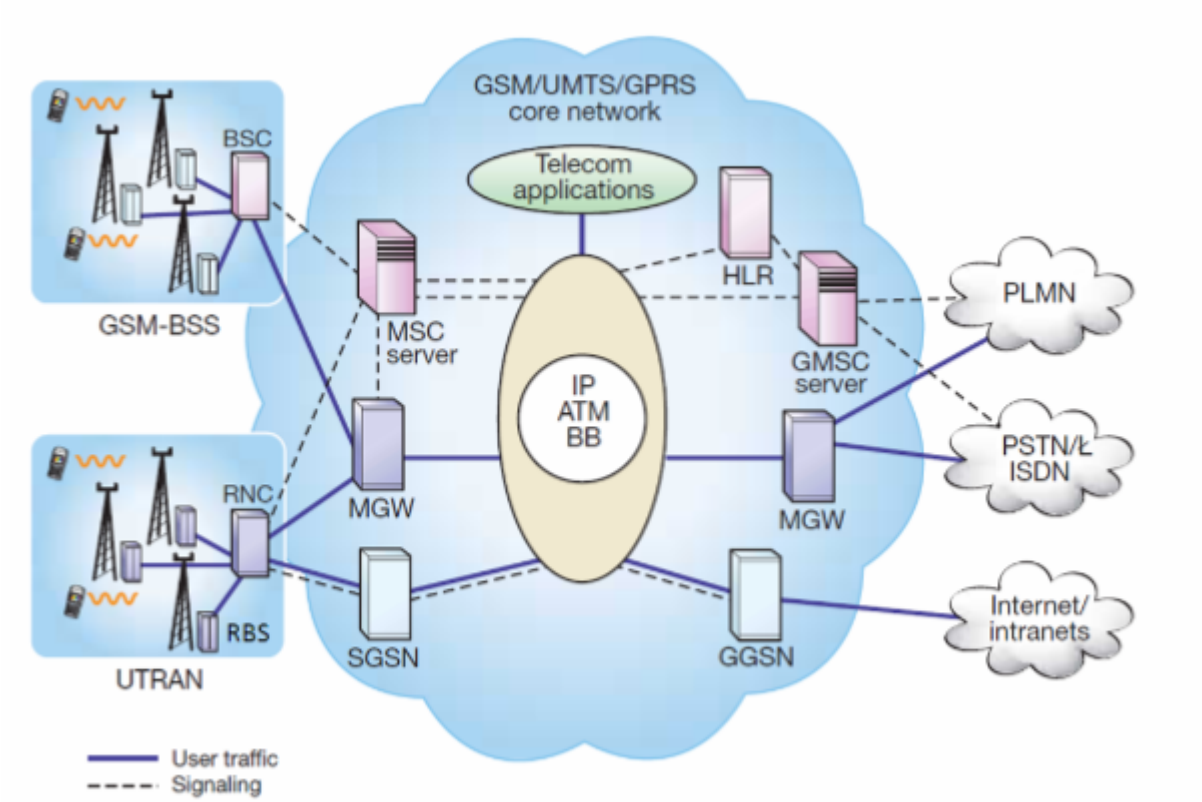

Рис. 1.5. Структура телекомунікаційної мережі

Незалежно від інтеграції та взаємозв’язку різних вузлів у мережі, дизайн кожного вузла сам по собі є великою складною проблемою. Наприклад, RNC може легко містити від 500 до 700 ЦП, а функції програмного забезпечення охоплюють декілька ЦП. Однак ця кількість зменшується, оскільки випускаються нові процесори з більшою потужністю. Це зниження є важливим для загальної вартості, споживання електроенергії та виробництва тепла системами.

Що стосується функціональності та послуг, у типовій телекомунікаційній системі за секунду має бути встановлено, маршрутизовано та керовано великою кількістю з’єднань. Крім того, на них також повинен бути зроблений розрахунок вартості. Крім того, типова телекомунікаційна система може мати термін служби приблизно 20-30 років.

Таким чином, можливість оновлення та обслуговування таких систем також має велике значення. Оновлення програмного забезпечення має здійснюватися таким чином, щоб найменше вплинуло на доступність системи. Ось чому такі функції, як гаряча заміна та підключення, а також можливість виконувати перезапуски на різних рівнях деталізації (одна плата, колекція плат або повний вузол) є дуже бажаними та затребуваними в цьому домені. Щоб охопити більше аспектів щодо моделювання та представлення нефункціональних вимог у проектуванні таких складних систем, ми пропонуємо рішення профілювання UML, що складається з концепцій із SysML [1] для відстеження та MARTE [2] для моделювання загальних нефункціональних властивостей та їх аналіз. Для вимог безпеки, які властиві телекомунікаційному домену, але не охоплюються MARTE ми використовуємо доступні профілі UML для безпеки (а саме UMLsec [3]). Крім того, оскільки MARTE, SysML і UMLsec є профілями UML, вони швидше для розробників, які використовують UML, і вони також служать можливим об’єднуючим фактором між відділами розробки.

Як показано на рисунку 1.6, вимоги та зв’язки між ними та артефактами дизайну моделюються за допомогою концепцій SysML.

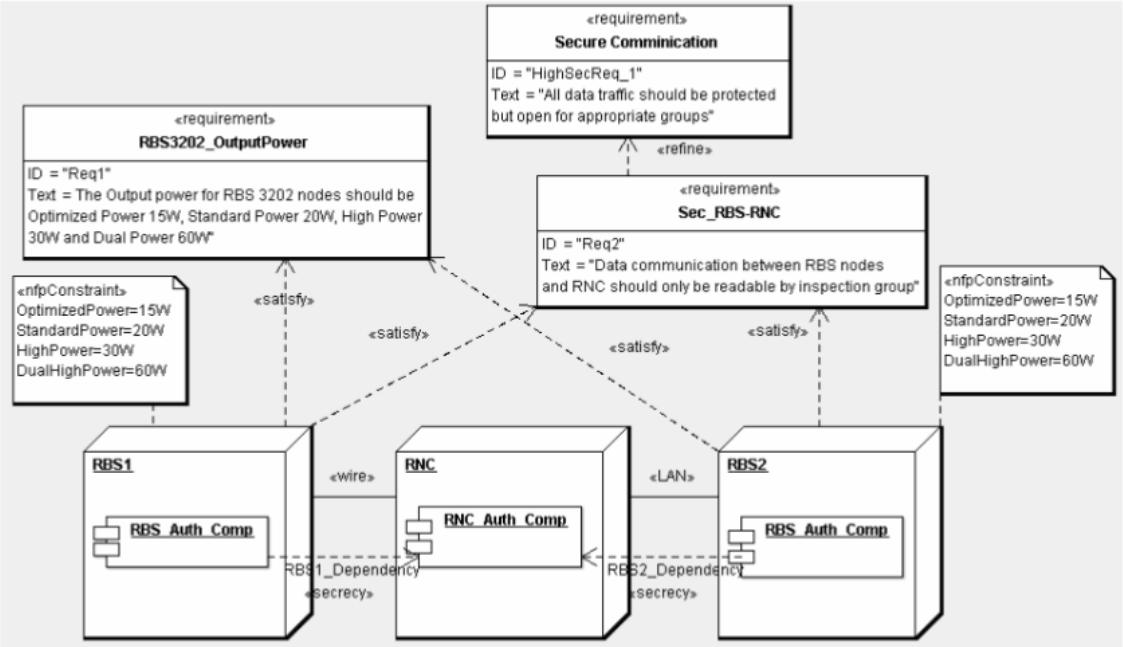

Рис. 1.6. Вимоги безпеки для вузлів RBS

Нефункціональні концепції MARTE (тобто nfp, nfpconstraint, PowerUnitKind і NFP Power) використовуються для моделювання вимог до вихідної потужності вузлів RBS. Концепції безпеки в даній моделі представлені за допомогою стереотипів UMLsec. Зв'язок між RBS1 і RNC позначено стереотипом дроту, а зв'язок між RBS2 і RNC позначено протоколом LAN.

В даному підрозділі подано підхід який полягає у розгляді телекомунікаційних систем як субдомену RTES (Real-Time Embedded Systems) а, отже, прийнятний з доступних рішень моделювання для нефункціональних вимог та їх аналізу, які вже існують у домені RTES.

Висновки до розділу

В даному розділі проведено аналіз дослідницьких позицій щодо деяких питань, що стосуються дизайну роботи. Виконано опис використовуваних стратегій та методів дослідження. Здійснено опис та дослідження моделі та структури системних фреймворків телекомунікаційних систем.

РОЗДІЛ 2. СТРУКТУРИЗАЦІЯ ДОМЕНІВ ТЕОРЕТИЧНОЇ ТА ПРАКТИЧНОЇ АКТИВНОСТЕЙ

2.1. Структура концепції об’єднання доменів при розробці комплексного фреймворку

Багато ідей для Framework виникли в результаті роботи над розробкою середовища проектування телекомунікаційного обладнання (HDE – hardware design environment). Керівний принцип був виражений таким чином: «HDE має поєднувати довговічну архітектуру з гнучкою функціональністю на основі покупки».

У цьому документі описується модульна архітектура процесу, в якій основними елементами є ядро процесу, компоненти процесу та прикладні процеси. Ідея полягала в тому, щоб розглядати процес як будь-який інший продукт і налаштовувати індивідуальні процеси з компонентів процесу. У цій архітектурі ядро процесу було раннім прикладом стабілізуючого ядра.

Компонент процесу — це пакет описів процесів, перевірок статусу, правил, інструментів, шаблонів тощо з метою надання дизайнеру повного набору утиліт для конкретного завдання проектування. Процес розробки апаратного забезпечення був більш неоднорідним, ніж процес розробки програмного забезпечення, оскільки розроблялися різні типи апаратного забезпечення, такі як друковані плати, багатокристальні модулі тощо. Таким чином, контекстний аспект був більш помітним у проектуванні обладнання, ніж у проектуванні програмного забезпечення (контекстуальність).

Раніше було визнано, що дані, які використовуються в компонентах процесу, повинні бути перекладені між внутрішнім і зовнішнім компонентом. Однією з цілей було архівувати дані у форматі, який не залежить від конкретних інструментів, які можуть використовуватися в компоненті процесу. Для цього було розроблено механізм трансляції даних між компонентами процесу. Цей механізм, який називається Модель обміну

інформацією про дизайн (DIM - Design Information Interchange Model), був визначений таким чином: «DIM — це своєрідна структура для обробки інтерфейсів між даними. DIM встановлює правила, за якими формати можуть використовуватися для різних переглядів інформації про апаратне забезпечення, наприклад, описи поведінки, описи логічних схем, списки мереж, тестові дані, макет. Крім того, DIM надає інструменти для спрощення перекладу між форматами” [29]

Іншим джерелом моделі переходу була модель даних на основі специфікацій (SBDM). Модель SBDM може використовуватися як потенційна модель для обробки межі між різними контекстами: «Модель даних на основі специфікацій (SBDM) — це об’єднуюча структура для моделювання конфігурацій систем, які містять компоненти з різних інженерних дисциплін» [31].

Основна конструкція SBDM — це рекурсивна структура специфікацій, які реалізуються реалізаціями та, у свою чергу, потребують інших специфікацій. Це було використано для спроби поєднати модулі процесу з моделлю SBDM системи, яка буде розроблена. На рисунку 2.1 подана специфікація може бути реалізована в програмному або апаратному забезпеченні з використанням різних компонентів процесу для розробки реалізацій (перехід домену).

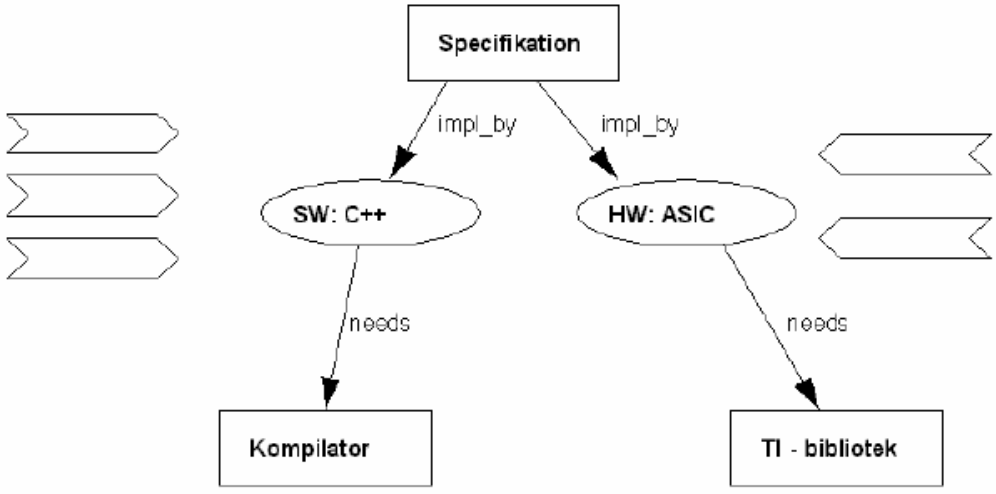

Рис. 2.1. Компоненти процесу та SBDM

Контекстний погляд на архітектуру це модель процесу яка залежить від ролі акторів, які використовують процес (контекстуальність). Спочатку можна використовувати діаграми інформаційних потоків (IFD) як комплексний спосіб представлення процесів, що подано на рисунку 2.2. В цьому випадку IFD мають ряд переваг, наприклад, вони мають основну структуру, яку можна застосувати до будь-якого модуля процесу та будь-якого рівня в ієрархії процесів

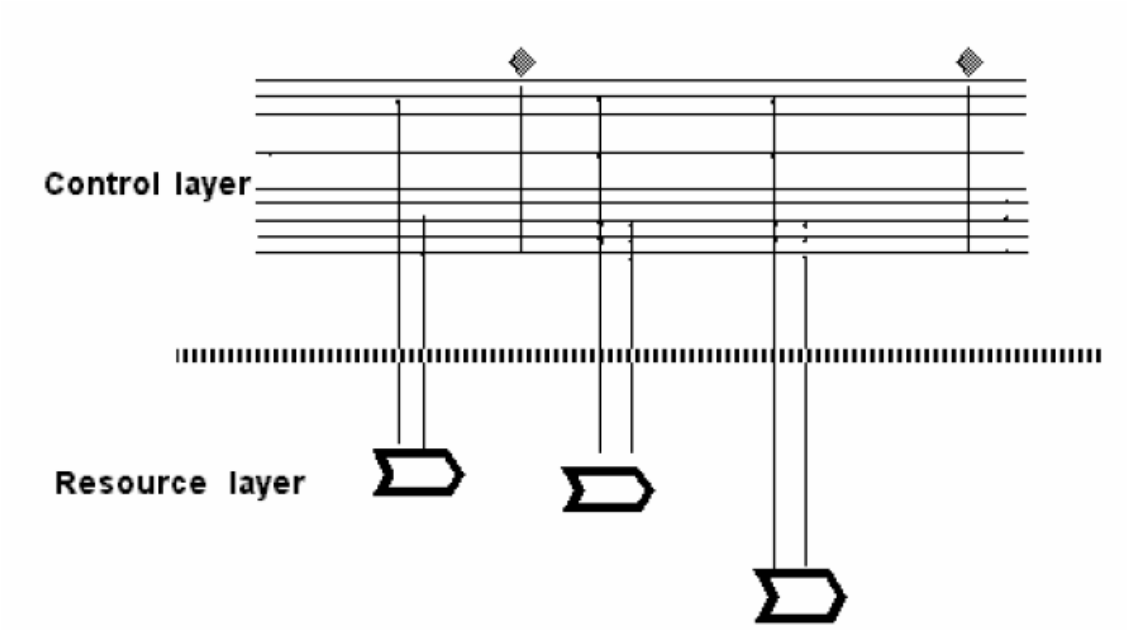

Рис.2.2. Діаграми потоку інформації

бачимо, що розділення моделі процесу на рівні «контролю» та рівні «ресурсів» сумісно з визначенням координації — це управління залежностями між видами діяльності».

Ще одне зауваження полягає в тому, що необхідно розділити ті аспекти процесу, які є спільними для групи користувачів, і те, що є унікальним для конкретної групи. Ці аспекти називаються стороною процесу «адміністратор» і «підприємець» відповідно. Методологія та системи підтримки вважаються товарами і як такі належать стороні підприємця, тоді як правила ідентифікації, системи якості тощо є більш довговічними та

належать стороні адміністратора. Тут ми можемо помітити аспект балансу, який пізніше став проявлятися як стабілізуюче ядро в структурі.

Пізніше діаграми інформаційних потоків можна використовувати як стандартний спосіб представлення процесів для розробки обладнання в проекті.

Одним з яскравих прикладів стала розробка модуля, який є складною інтегральною схемою. Діаграми інформаційних потоків для цього компонента процесу показано на рисунку 2.3.

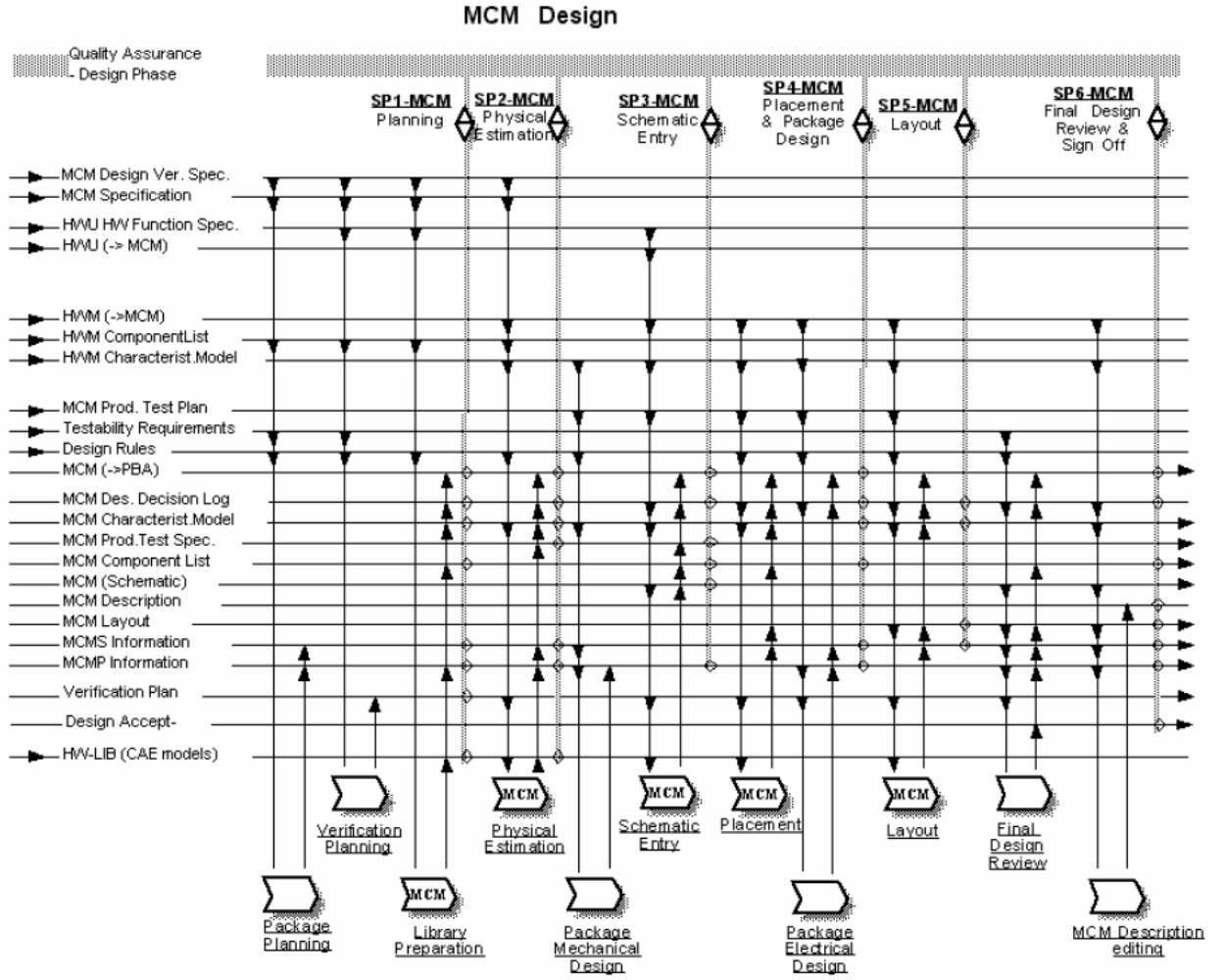

Рис. 2.3. Діаграма потоку інформації: конструкція багатофункціонального модуля

Щоб досягти більш рівномірного балансу, стратегія була визначена таким чином: «Починаючи із загальної архітектури процесу, ми спочатку зосереджуємося на кожній області процесу окремо, а потім ми їх інтегруємо».

За допомогою концептуальної моделі (рис. 2.4) можна визначити структуру всієї системи. Деякі частини моделі вже існували в традиційному способі роботи (модель, орієнтована на водоспад), але деякі частини були новими. Оскільки пункти координації на той час були переважно документами, більшість ящиків представляють різні типи документів. Це також може бути IDMP. що було вдосконалено до продукту, що містить інструкції, описи методів, шаблонів тощо.

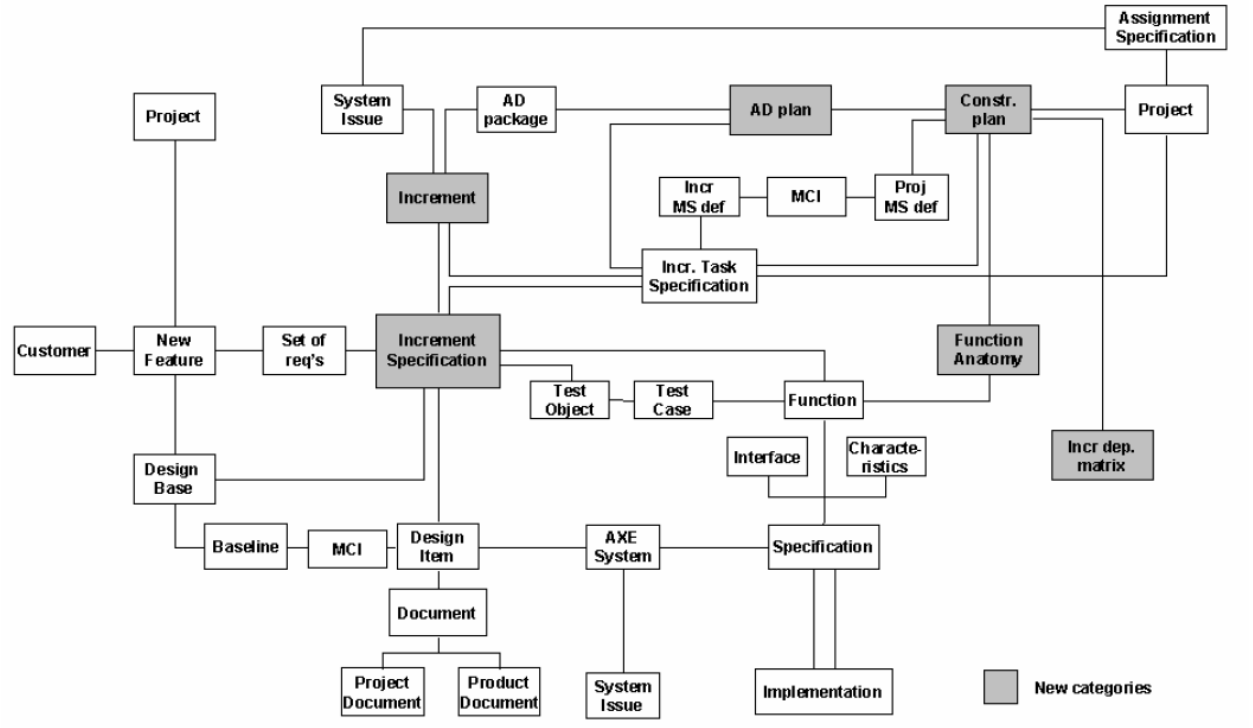

Рис. 2.4. Початкова версія концептуальної моделі

Висновки з оцінки пакету методів є наступними:

• Інкрементний дизайн є можливістю, але потребує деяких покращень.

• Проміжні етапи слід планувати крок за кроком (потрібен хороший інструмент).

• Слід погодитися, що для одного проекту потрібно кілька точок прийняття рішень.

2.2. Методологічні аспекти стратифікації процесу імплементації фреймворку

Вивчення програмної інженерії розвивалося протягом останніх тридцяти років, але воно ніколи повністю не вирішувало кризу програмного забезпечення. Як складова частина інженерії програмного забезпечення, методології розробки програмного забезпечення також еволюціонували, починаючи з поверхневих і неформальних внутрішніх методологій до об'єктно-орієнтованого методології нового тисячоліття. Зміни парадигми були тривалими й болісними, але об’єктно-орієнтовані методології зуміли вижити і розвиватися. Методології розробки об’єктно-орієнтованого програмного забезпечення існують близько двох десятиліть, але багато проблем, пов'язаних з цими методологіями на початку досі залишаються невирішеними.

Ставлення до методологій як до товару призводять до зайвого конструктивного безладу, привабливих, але тьмяних обгорток і неінформативних, іноді навіть рекламних описів, які корисні, коли намагаються продати методологію, але відволікають від пояснення та розуміння її основного процесу.

Крім того, методології за своєю суттю є складними; навіть методисти, які намагаються бути науковими та професійними у своєму підході до визначення суті, їхні процеси занадто часто закінчуються тим, що дають занадто мало або занадто багато деталей на неправильному рівні. І часто методисти не є об’єктивними та неупередженими щодо власних творінь (і цього не слід очікувати); особливості, на яких наголошують методисти, часто не є такими суттєвими для розв’язання проблем предметної області методології, але є ті, що методист вважає важливим або унікальним. Усе це

ще більше додає плутанини розробнику, який планує використовувати методологію.

Необхідний комплексний огляд методологій розробки об'єктно-орієнтованого програмного забезпечення, але з точки зору процесу. Існують відгуки про методології з точки зору мов моделювання, доступних у літературі, але є мало доступного сучасного матеріалу синтезу, який фокусується на процесі.

Такий огляд необхідний виходячи з таких мотивів:

— щоб допомогти користувачам методологій — наприклад, керівникам проектів, командам розробників;

— допомогти розуміти процеси, чергування фаз та очікувані взаємодії, які неявно або явно вимагає методологія;

— надавати інформацію, яка може бути корисною для користувачів методик, які виконують реальну імплементацію;

— адаптувати або вибрати процеси для свого конкретного проекту чи контексту;

— допомогти користувачам оцінити процеси з огляду на їхні проекти та контексти.

В такому огляді неминуче будуть питання, пов'язані з мовами моделювання де необхідно абстрагуватися від цих деталей, щоб зосередитися на процесі.

Наприклад, зусилля зі стандартизації для UML, звичайно, є важливими та корисними. Але ми узагальнюємо та переглядаємо методології за допомогою шаблону, орієнтованого на процес, виділення заходів, передбачених кожною методологією, зберігаючи опис і обговорення використовуваних мов (переважно діаграм і таблиць) як вторинних видів діяльності.

Опис, створений за допомогою цього шаблону, містить невелику критику щодо методологій — справді, це не є нашою метою — проте абстрагує та структурує їх у а спосіб, який дає змогу детально аналізувати

окремі методології, а також порівняти аналіз колекцій . Опис методології на основі цього шаблону складається з наступних частин: опис історії методології та відмінні риси, а також абстрактний огляд процесу розробки передбачені методикою; ряд підрозділів, по одному для кожного підпроцесу високого рівня в методології процесу розробки, кожен з яких складається з:

— детальних дій, що виконуються в підпроцесі, і порядок їх виконання зберігається;

— стислий опис мов моделювання, які використовуються в підпроцесі в рамках своєї відповідної діяльності.

Мови моделювання, хоча і необхідні для повного розуміння механізмів, що використовуються в процесі методології, як правило об’єктно-орієнтованого програмного забезпечення, є орієнтованими на процес, тому немає потреби захаращувати методологію та процес.

Слід почати з основних визначень і короткої історії методологій розробки об'єктно-орієнтованого програмного забезпечення, побудови схем появи гібридних методологій, перехід до основоположних методологій та розробки інтегрованих методологій та їх гнучких відповідників. Потім можна надати комплексний огляд процесів вибору основоположних, інтегрованих і гнучких методологій, а також шаблонів процесів і метамоделей процесів, перш ніж завершити обговоренням розробки процесів методологій рівня фреймворку власне. Методології, шаблони процесів і метамоделі процесів, розглянуті в цьому дослідженні орієнтовані виключно на їх імплементацію саме на рівні фреймворку IDE рівня.

Методологія розробки програмного забезпечення (SDM) — це основа для застосування програмного забезпечення, інженерні практики з конкретною метою забезпечення необхідних засобів для розробки програмно-інтенсивних систем.

Отже, методології розробки програмного забезпечення вважаються невід'ємною частиною програмної інженерії, оскільки методології надають

засоби для своєчасного та впорядкованого виконання різноманітних більш детальних технік і методів розробки програмного забезпечення.

Хоча методологію розробки програмного забезпечення можна умовно визначити як «рекомендовану колекцію етапів, процедур, правил, методів, інструментів, документації, управління та навчання, що використовуються для розробки системи», її легше зрозуміти, коли вона описується як така, що складається з двох основних частин:

1) набір конвенцій моделювання, що включає мову моделювання (синтаксис і семантика);

2) процес, який дає вказівки щодо порядку проведення заходів; визначає, які артефакти, які слід розробляти за допомогою мови моделювання; керує виконанням завдань окремих розробників і колективу в цілому; пропонує критерії для моніторингу та вимірювання продуктів і заходів проекту.

У той час, як мова моделювання надає розробникам засоби для моделювання різних аспектів системи, процес визначає, які дії слід виконувати щоб розробити систему, в якому порядку та як. У своїй найбільш абстрактній формі процес це послідовність кроків (іноді називають рецептом), яка спрямована на імплементацію у застосування мови моделювання для виконання ряду завдань розробки програмного забезпечення. Таким чином, процес діє як динамічний, поведінковий компонент методології, яка керує підпроцесами технічного розвитку та управління, а також, охоплюючи фази, процедури, правила, методи та інструменти, що призначаються за методологією, а також питання, що стосуються документування та управління проектом. Також було припущено, що кожна методологія має основу в формі сукупності припущень і гіпотез методології, що відображені в методології, і які підтримують і раціоналізують використання методології в її передбаченому контексті. Тут слід зазначити, що існує серйозний аргумент проти використання цього терміну методології в цьому контексті, прихильники якої вказують на те, що

методологія є вивченням методів, а не тип методу, який тут використовується, і тому пропонуємо використання більш загального терміна.

Хоча цей пункт лексично правильний, ми вирішили використовувати термін методологія через його широке впровадження в спільноті розробників програмного забезпечення, а також через узагальненість терміну метод; тоді як метод використовується в інженерії програмного забезпечення посилаючись на різні концепції та конструкції на різних рівнях абстракції , та різноманітні ступені деталізації, методологія однозначно використовується у зазначеному тут значенні.

2.3. Сутність об'єктно-орієнтованої методології

Методології розробки об'єктно-орієнтованого програмного забезпечення (OOSDM) спеціально спрямовані на перегляд, моделювання та реалізацію системи як набору взаємодіючих об'єктів, що використовують спеціалізовані мови моделювання, необхідні для вирішення конкретних питань об'єктно-орієнтованої парадигми.

Спочатку парадигма базувалася на поняттях, введених у системне моделювання, операційні системи, дані абстракції та штучного інтелекту. Об'єктно-орієнтована парадигма набула широкого поширення та популярності в минулому столітті, саме через об'єктно-орієнтовані мови програмування. Була визнана застосовність об’єктно-орієнтованого підходу до системного аналізу та проектування, і наступний ентузіазм був таким, що ряд запроваджень методології розробки об’єктно-орієнтованого програмного забезпечення мали місце. Короткий опис категорій OOSDM і тенденцій їх розвитку допоможе в подальшому уточненні доменів побудови фреймворку.

Перші методології розробки програмного забезпечення, названі об'єктно-орієнтованими, були насправді гібридними: частково структурними і частково об'єктно-орієнтованими. Фаза аналізу була типово виконаною з

використанням методів структурованого аналізу (SA), створюючи діаграми потоків даних, діаграми зв’язків сутностей і діаграми переходів станів, тоді як фаза проектування була в основному відображенням результатів аналізу в об’єктно-орієнтований проект програмного забезпечення. Тому ці методології були класифіковані як трансформаційні.

Перші суто об'єктно-орієнтовані методології з'явилися під впливом структурованих та/або орієнтованих підходів. Друге покоління об'єктно-орієнтованих методологій еволюціонували з першого покоління. Так звана «методологічна війна», під час якої десятки методологій конкурували за частку в індустрії розробки програмного забезпечення, відбулася в цей період. Величезна кількість запроваджених методологій стала настільки непомірною, що вибір правильної методології для проекту програмного забезпечення була серйозною спробою. Розчарування в співтоваристві розробників програмного забезпечення незабаром призвело до зусиль, спрямованих на інтеграцію та уніфікацію.

Першим суттєвим результатом стала Уніфікована мова моделювання (UML), прийнята групою управління об'єктами (OMG) як стандартна мова об'єктно-орієнтованого моделювання. Поки розроблявся UML, також робилися широкі спроби інтегрувати основоположні методології, що означало кінець ери другого покоління.

Методології першого та другого покоління спільно називаються основоположними методологіями, оскільки вони започаткували недосліджену область чистого об’єктно-орієнтованого аналізу та дизайну, і тим самим заклали основу для подальшої еволюції. Відповідно, ідеї, викладені цими методологіями, глибоко вплинули на галузь об'єктно-орієнтованої інженерії програмного забезпечення, що швидко розвивається.

Багато з концепцій, умов моделювання та методів, запроваджених цими методологіями, досі широко використовуються, і деякі з цих методологій все ще мають ряд відданих послідовників, що підтверджує що основоположні методології аж ніяк не застаріли. На малюнку показано методики, розроблені

в період максимально насиченої еволюції. Він також показує хронологію еволюції та генеалогічні зв’язки серед методологій, підкреслюючи впливи та внески окремих підходів.

UML є результатом спроб уніфікувати мови візуального моделювання, які використовуються в об’єктно-орієнтованих методологіях, після усвідомлення того, що, хоча вони здебільшого відрізняються за моделлю процесу та життєвого циклу, багато об’єктно-орієнтованих методологій, використовують діаграми, які по суті були ідентичними. Тому починається тенденція інтеграції з уніфікацією мов моделювання. UML тому вважається важливою віхою, що знаменує кінець основоположних методологій і початок інтеграційної ейфорії. Це наголошувалося з самого початку багатьма методистами, які займаються оцінюванням внесок у UML, що він повинен бути незалежним від процесу, щоб методології могли використовувати його без необхідності відповідати певному процесу.

Це дійсно було встановлено, як основна мета розробки UML і явно згадується в офіційних специфікаціях. Проте протилежне не вірно: процеси дійсно стають залежними від мови моделювання, яку вони приймають. Це призвело до ускладнень в деяких методологіях, нав'язування UML як стандарту, або наполягання на власному ексклюзивному моделюванні мови, або використання UML разом із конструкціями моделювання, що не підтримується. Таким чином, на UML найбільше впливають мови моделювання, які використовуються в методології OMT, OOSE та Booch. Після прийняття на об’єкт управління Group (OMG) UML зараз вважається стандартом де-факто для об’єктно-орієнтованого моделювання та постійно переглядається та переглядається під наглядом OMG.

Методології цієї категорії є результатом інтеграції початкових методологій і характеризуються своїм орієнтованим на процес ставленням до розробки програмного забезпечення, як правило, націленим на широкий спектр програм.

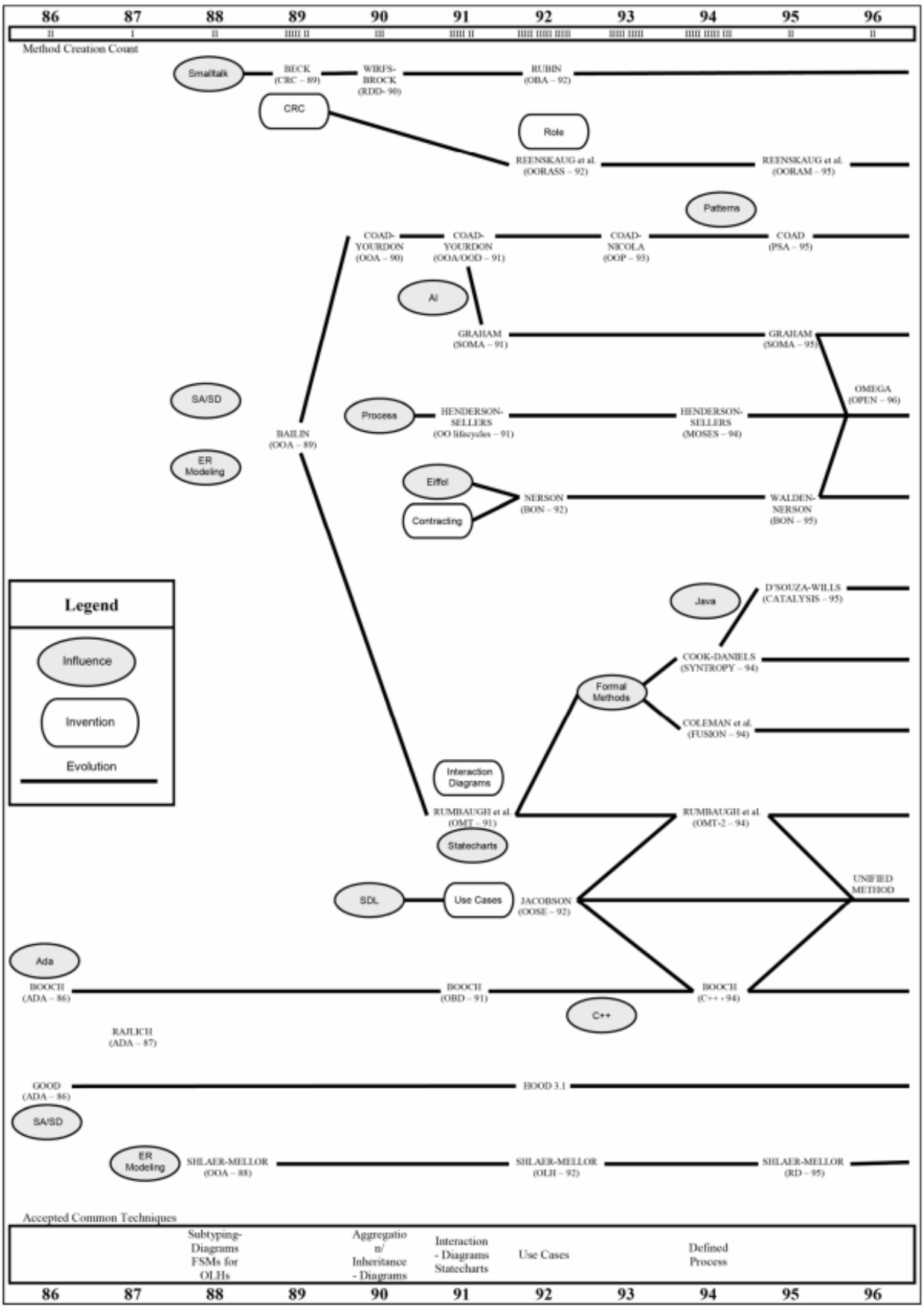

Рис. 2.5. Еволюція об’єктно-орієнтованих методологій

Зусилля інтеграції призвели до величезних змін методології, якими важко керувати та впроваджувати. У спробі щоб досягти керованості, деякі з них пішли на крайні заходи для забезпечення керованості.

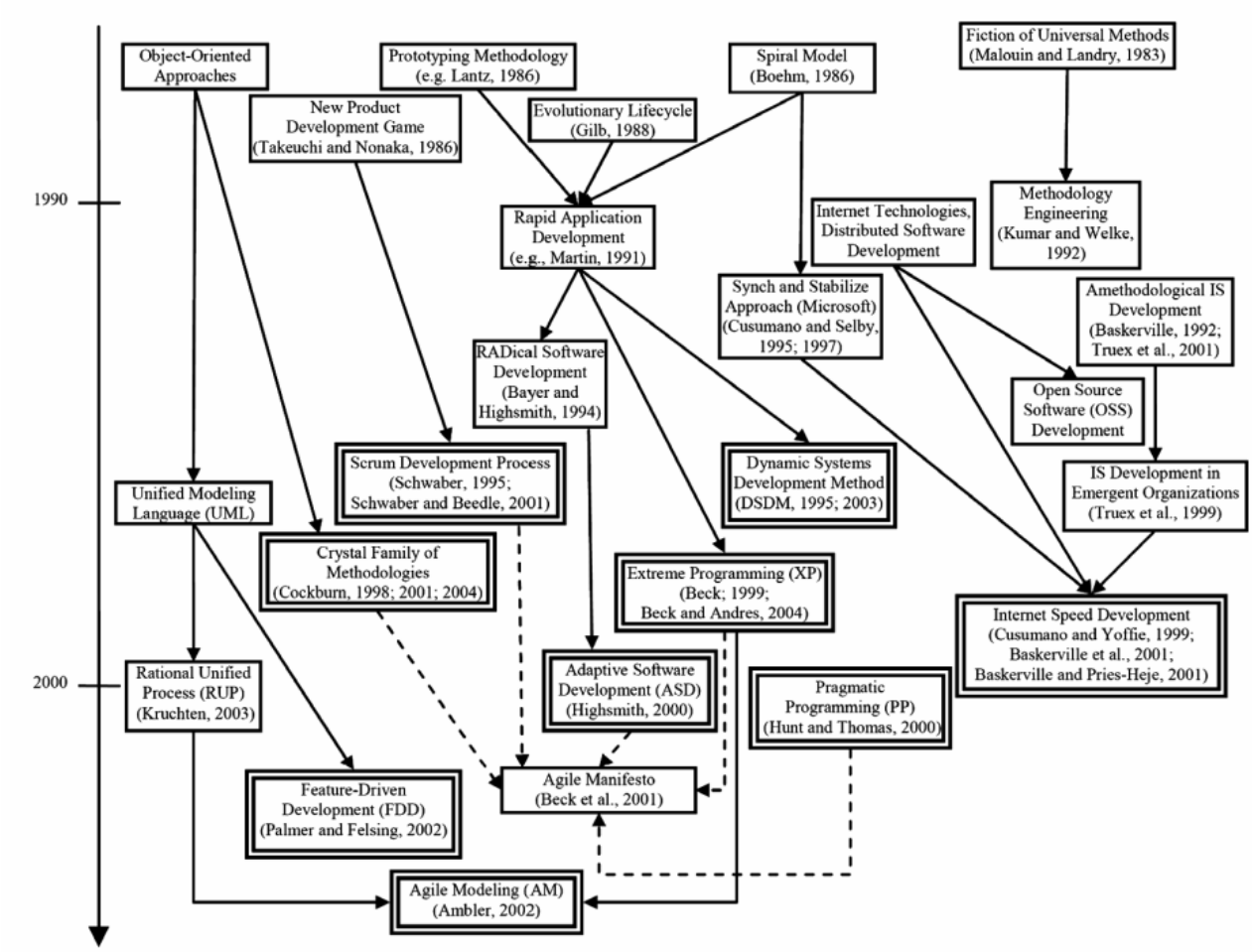

Рис. 2.6. Еволюція AGILE- методологій

Можливості щодо налаштування (RUP), перетворилися на загальні структури процесів, які повинні бути створеними, щоб отримати процес, а інші вдаються до шаблонів процесу для його налаштування. Громіздкі і складні інтегровані методології можуть багато чого запропонувати з точки зору компонентів процесу, моделі, а також питання управління та вимірювання. Крім того, деякі з них пропонують корисні ідеї щодо безперебійної розробки, управління складністю та моделювання підходів керування.

Колись було поширене уявлення про те, що гнучкі методології — це не що інше лише контрольовані підходи кодування та виправлення з невеликими ознаками чіткого процесу або без нього. Це вірно для невеликої — хоча й впливової — меншості цих методологій, які, по суті, базуються на практиці розробки програм, кодування та тестування, які, як вважають, забезпечать підвищення гнучкості та продуктивності розробки програмного забезпечення. Найгнучкіші методології повинні включати явні процеси, але прагнучи зберегти їх якомога меншими. На рисунку показано еволюційну карту для ряду гнучких методологій, підкреслюючи те, як на них вплинули попередні методології та практики. Гнучка розробка програмного забезпечення також зазнала впливу теорії шаблонів. Загальне ставлення гнучких методологій до програмного забезпечення та розробки в цілому було підсумовано в «Manifesto Agile», узгодженому як головний документ в agile методології.

Поява UML дозволила методологам зосередитися на процесах замість того, щоб займатися розробкою нових мов моделювання, а досвід, отриманий порівняно складно допоміг методистам визначити закономірності та загальні риси процесів. Шаблони процесів – це результати застосування абстракції до компонентів процесу, тим самим представляє шляхи розвитку методологій через композицію відповідних екземплярів шаблону. Метамоделі процесу, з іншого боку, є результатами застосування абстракції на загальний процес, що забезпечує узагальнення процесу або метамоделі; процеси можуть тоді бути побудованими через інстанціювання цих метамоделей.

Через велику кількість цих методологій ми розглядаємо лише відомі та значимі методології, які, відповідно до еволюційної шкали на малюнку є відповідними представниками окремих гілок. Кожна методологія використовує власну мову моделювання, яку також слід охопити в описі методології, яка буде корисною; однак умови позначення виходять за межі дослідження.

Було представлено методологію для об’єктно-орієнтованого аналізу та проектування. Їх аналіз при створенні методу розглядав об’єкти як сутності даних, а не інкапсуляцію даних і поведінки, таким чином нехтуючи об'єктними методами. Тому його в основному розглядали як метод інформаційного моделювання, а не як повноцінну об’єктно-орієнтовану методологію. Впровадження методу проектування та подальші вдосконалення перетворили цей спочатку неадекватний метод на конкурентну методику проектування. Остаточна версія процесу охоплює фази аналізу, проектування та реалізації життєвого циклу розробки програмного забезпечення. Його можна розбити на ряд кроків (зазвичай виконуються послідовно), розділити систему на домени відповідно до визначених типів доменів в методиці; введені структури практично розділяють функціональність методології .

Огляд об'єктно-орієнтованої розробки програмного забезпечення, орієнтований на процес і поведінку програмної системи включає наступні рівні: проблемна область, незалежні від додатків служби, фізична архітектура та фізична реалізація; аналіз прикладної (проблемної) області.

Необхідно виконати наступні дії : підтвердити аналіз за допомогою статичної та динамічної перевірки; виділити вимоги до незалежних від додатків доменів послуг, які підтримують домен додатків; проаналізувати домени обслуговування; вказати компоненти архітектурної області (фізична конфігурація програмне забезпечення); побудувати архітектурні компоненти; транслювати (впроваджувати) моделі аналізу відповідних доменів в архітектурні компоненти.

Система спочатку розділяється на кілька доменів. Існує чотири типи доменів, один або кілька з яких (відповідно до наступного списку) визначені в кожній системі:

— домен програми: домен, що стосується конкретно кінцевого користувача (проблема домен);

— кілька доменів послуг: відносно загальні, незалежні від програми домени, безпосередня підтримка домену програми; приклади включають інтерфейс користувача та домен датчиків і виконавчих механізмів у системах реального часу;

— архітектурна область: зображення фізичної конфігурації програмного забезпечення системи і стосується організації даних, управління та алгоритму в системі в цілому;

— кілька доменів реалізації: охоплюють легкодоступні, компоненти рівня реалізації, що підтримують програмну систему під час виконання; наприклад, операційна система та конструкція мови програмування і компоненти.

Домени організовані у відносинах клієнт-сервер із клієнтськими доменами залежно від доменів серверів, щоб надавати їм необхідні послуги. Результати цього кроку моделюються в діаграмі доменів, що зображує домени та їхні відносини клієнт-сервер (називаються мостами).

Наступний крок передбачає застосування об'єктно-орієнтованого аналізу (ООА) до прикладної області. Моделі Shlaer-Mellor ООА складаються з трьох окремих частин, побудованих у такому порядку:

1) Будується інформаційна модель об’єкта, яка визначає об’єкти домену, і відносини між ними.

2) Будуються моделі стану, які показують життєвий цикл (поведінку) кожного об'єкта.

3) Створюються моделі специфікації дій, які зображують обробку, що відбувається в стандартній моделі. Зазвичай існує одна специфікація дії для кожного стану в кожному об’єкті життєвого циклу. Специфікації дій зазвичай виконуються на діаграмах потоку даних дій (ADFD).

Якщо домен завеликий для аналізу як єдиного цілого, можливо, необхідно розділити його на частини підсистеми. Створено три моделі, щоб показати зв’язки між підсистемами в межах домену. Ці моделі:

1) Модель зв’язку підсистеми: показує структурні зв’язки між об’єктами в різних підсистемах;

2) Модель зв’язку підсистеми: показ зв’язку подій між об’єктами в різних підсистемах;

3) Модель доступу до підсистеми: показує доступ до даних між об’єктами в різних підсистемах.

Методологія також передбачає виготовлення ряду похідних моделей для кожної з перелічених тут підсистем:

1) Модель зв’язку об’єкта: показ зв’язку подій між об’єктами.

2) Список подій: показ подій, що надсилаються в межах або між моделями стану.

3) Модель доступу до об’єктів: показ доступу до даних між об’єктами.

4) Таблиця процесів стану: показує процеси в усіх ADFD.

5) Діаграма потоків керування: показує послідовності дій, які виконуються у відповідь на кожну зовнішню подію.

Набір правил, формує основу для статичної перевірки набору моделей ООА. Крім того, є процес призначений для динамічної перевірки моделі-набору шляхом імітації виконання моделі. Моделювання бажаної поведінки виконується в чотири етапи:

1) Встановлення бажаного початкового стану системи в значеннях даних в інформаційній моделі об’єкта.

2) Ініціація бажаної поведінки за допомогою події, надісланої в модель стану.

3) Виконання обробки, як визначено моделями специфікації дій та відповідно до послідовності моделей стану.

4) Оцінка результату порівняно з очікуваними результатами відповідно до бажаної поведінки.

В домені на діаграмі кожен міст між доменами відображає те, що вимагає домен клієнта та домен сервера. Ці вимоги призначаються доменам і формі служби і є основою для аналізу решти доменів у системі.

Після уточнення вимог до усіх клієнтських доменів, Shlaer-Mellor OOA застосовується до кожної служби, що залишилася в домені. Після аналізу кожного домену його поведінка динамічно перевіряється шляхом виконання його моделі OOA. Цей процес продовжується вниз уздовж мостів до досяжних доменів, що або належать до загальносистемної архітектури (тобто архітектурного домену) або вже існує (домени реалізації, такі як операційна система, мова програмування або комунікаційна мережа).

Огляд об'єктно-орієнтованої розробки програмного забезпечення, орієнтований на процеси в кінцевій системі для реалізації елементів моделей (кінцевих автоматів, черги прийому подій тощо). Структури представляють сценарій для перекладу моделей OOA клієнтських доменів. Вони реалізовані як шаблони фрагментів коду, які заповнюються в (заповнений) на основі елементів моделі OOA (наприклад, архетипи для класів C++ заповнені з об’єктів моделі OOA).

Архітектурну область можна спроектувати з використанням нотацій Shlaer-Mellor OOA, але також можуть бути розробки за допомогою інших методів. Якщо потрібен об’єктно-орієнтований дизайн, то рекомендується мова об’єктно-орієнтованого проектування Shlaer/Mellor (OODLE). OODLE використовує чотири типи діаграм, упорядкованих у багатошарову структуру для моделювання дизайну об'єктно-орієнтованої програми, бібліотеки або середовища:

- діаграми успадкування, які показують відносини успадкування між класами;

- діаграми залежностей, що показують використання (клієнт/сервер) і реляції між ними;

- діаграми класів, які показують зовнішній вигляд кожного класу;

- діаграми структури класів, що показують структуру методів і потік даних і контроль в межах класу.

Механізми та конструкції із зазначенням фізичного дизайну системи деталізуються і налаштовуються в такій постановці задачі. Будуються

архітектурні механізми, що реалізують загальносистемне управління даними, функціональність і контроль, а також архітектурні структури достатньо детально, щоб визначити однозначні шаблони для додавання функціональності клієнтським доменам програмних механізмів. Таким чином, створено основу для впровадження компонентів, що відносяться до клієнтських доменів; вони будуть побудовані та вбудовані в архітектуру на наступному етапі. Як це завдання, так і наступне займаються реалізацією загальної проблеми, широкого використання компонент та конструкцій, що надаються доменом реалізації.

Моделі, що реалізовано в домені, які є прямими або непрямими клієнтами архітектурного домену в архітектурній конфігурації за допомогою виділених конструкцій, деталізовані у постановці завдання. Деталі останнього кроку багато в чому залежать від конструкції, обраної для системи та створених архітектурних компонент. Наприклад, розглядаючи загальний випадок багатозадачних і багатопроцесорних систем основними видами діяльності будуть:

1) призначення екземплярів об’єктів завданням, а завдання — процесорам;

2) створення завдання на основі моделей ООА.

Як і багато інших ранніх об'єктно-орієнтованих методологій, методологія Coad-Yourdon має двофазне впровадження. Вони представили свій метод об’єктно-орієнтованого аналізу (ООА), що є порівняно спрощеною у своєму підході, але вона служила своїй меті як вступна об’єктно-орієнтована методологія в той час, коли інерція у прийнятті об’єктно-орієнтованої методології техніки здавалися надто великою, щоб їх подолати. Хоча дана методологія як правило, вважається такою, що охоплює лише загальний аналіз і фази проектування, вона тим не менше містить керівні принципи для реалізації, пропонуючи методи для переведення дизайну моделі в код. Загальний процес застосування методів аналізу та проектування показано на малюнку.

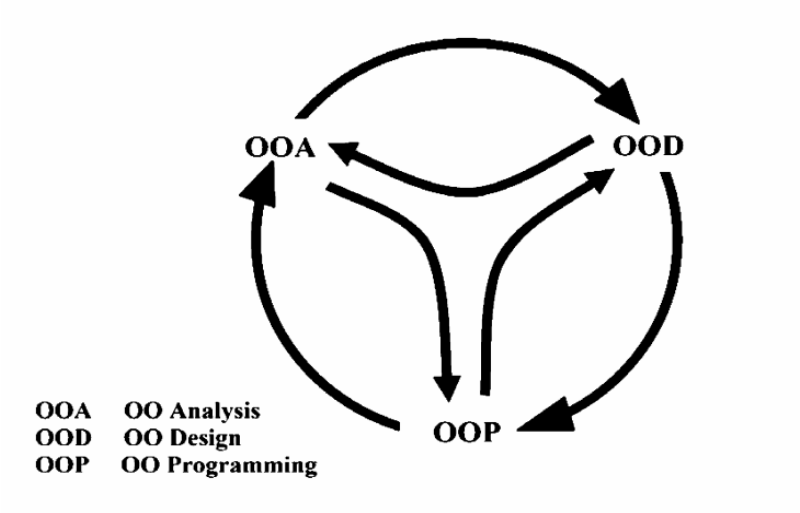

Рис.2.7. Модель розробки програмного забезпечення

Діяльність і результати OOA та OOD згідно з методологією Coad-Yourdon розглядаються в контексті основної задачі. Аналіз (OOA) частини методології складається з п'яти основних видів діяльності:

1) пошук класів (абстрактних класів) і класів і об’єктів (конкретних класів);

2) ідентифікаційні структури (узагальнення-спеціалізація та відношення «ціле-частина» між класами);

3) ідентифікація суб’єктів (розділів/підсистем);

4) визначення атрибутів і з’єднань екземплярів (зв’язки асоціації між сутностями);

5) визначення служб (операцій класу) і зв’язків із повідомленнями (викликів операцій).

Дослідники наголошують, що хоча ця діяльність ініціювалася послідовно, дії користувача не є послідовними кроками, оскільки перехід від однієї діяльності до іншої, особливо до раніше ініційованих кроків, є неминучим.

Результати цієї діяльності відображаються в діаграмі класів і об'єктів, яка є основною моделлю системи. Відповідно, до цих основних дій, результуюча діаграма класів і об’єктів складається з п’яти рівнів, кожна поверх попередньої, таким чином контрольовано додаючи деталі. Ці шари:

1) предметний рівень: який показує загальні розділи системи;

2) рівень класів і об’єктів: показ абстрактних і конкретних класів системи;

3) структурний рівень: який показує зв’язки «узагальнення-специфікації» та «ціле-частина» між класами;

4) рівень атрибутів: показ атрибутів класів і асоціаційних відносин між класами;

5) сервісний рівень: який показує роботу класів і потенціал передачі повідомлень між об’єктами (послідовність повідомлень, що може бути змодельована).

Діаграма класів і об’єктів доповнюється різними поведінковими діаграмами, створеними під час ідентифікації операцій і з’єднань повідомлень. Як правило, динамічна поведінка кожного класу є частиною об'єктно-орієнтованої розробки програмного забезпечення, сценарію орієнтованого на процеси, що фіксується на діаграмі стану об’єкта, простій формі діаграми переходів станів, і алгоритму, який слід застосовувати для кожного зі значущих сервісів, що описується простою блок-схемою, яка називається сервісною діаграмою.

На етапі проектування методології (OOD) система складається з чотирьох компонентів, кожен з яких забезпечує певну функціональність необхідну для реалізації вимог і впровадження системи. Компоненти перечислені нижче:

1) Компонент проблемної області (PDC): спочатку містить результати фази аналізу. Під час OOD його вдосконалюють і збагачують деталями реалізації, що все ще представляє частину дизайну, що містить функції, пов’язані з доменом користувача; тобто вимоги.

2) Компонент людської взаємодії (HIC): обробляє надсилання та отримання повідомлень до і від користувача. Класи в компоненті людської взаємодії мають імена взяті з мови інтерфейсу користувача, наприклад, вікна та меню.

3) Компонент керування завданнями (TMC): для систем, які потребують реалізації кількох потоків керування, розробник повинен створити компонент керування завданнями, організовувати обробку, координувати завдання (процеси) і надавати ресурси для міжзадачної комунікації. Цей компонент містить класи, які і забезпечують відповідні послуги.

4) Компонент керування даними (DMC): забезпечує інфраструктуру для зберігання та отримання об'єктів. Це може бути проста файлова система, система керування реляційною базою даних або навіть об’єктно-орієнтована система керування базою даних. Внесення в цей домен зазвичай представляє реляційні таблиці та/або більш складні дані/об’єкти серверів.

Основною діаграмою в кожному компоненті є діаграма класів і об’єктів (п'ятишарова архітектура). Динамічні діаграми (діаграми стану, об'єктів і сервісні діаграми) використовуються для розширення та доповнення інформації, яку вони передають за допомогою класу та об’єкту діаграми.

Представлений дизайн, орієнтований на відповідальність (RDD) починається з детальних вимог специфікації системи, що вже надана. Це означає, що певні вимоги до типових дій аналізу, включаючи виявлення вимог, були пропущені в методології, залишаючи інженеру вирішувати, який метод використовувати для виробництва та специфікації вимог.

Незважаючи на це, RDD справила великий вплив на сучасний стан об'єктно-орієнтованої інженерії програмного забезпечення, оскільки були дуже корисними введені поняття, та вперше продемонстровано та використано їх в цій методології. Нова версія RDD, що використовує ідеї з UML і практики, орієнтовані на випадки використання, також має було доцільною.

RDD моделює програму як набір об’єктів, які співпрацюють для виконання своїх завдань. Обов'язки сутностей включають два ключових пункти:

1) знання, які зберігає об’єкт;

2) дії, які може виконувати об’єкт.

Процес поділяється на дві фази: фазу дослідження та фазу аналізу.

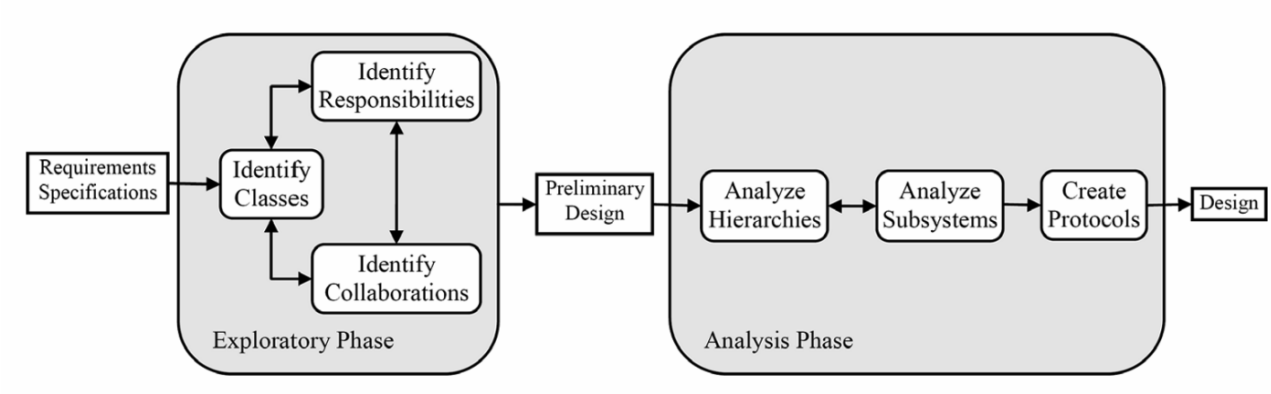

Рис. 2.8. Структура RDD процесу

Дослідницька фаза (RDD). Основними завданнями на цьому етапі є:

1) виявити класи, необхідні для моделювання програми;

2) визначити, за яку поведінку відповідає система, і призначити ці обов'язки конкретним класам;

3) визначити, яка співпраця має відбуватися між класами об’єктів для виконання накладених обов'язкових методів.

Як видно на діаграмі, що зображує процес RDD, виділяється три види діяльності: ідентифікація класів, визначення обов’язків та ідентифікація співпраці, повинні виконуватися ітеративно, щоб бути ефективними, оскільки результати кожної діяльності будуть вплинути на результати інших. Метод розробки, орієнтований на відповідальність, використовує картки Class Responsibility-Collaborator (CRC) щоб охопити окупації, обов’язки та співпрацю. Ці картки також записують відносини підклас-суперклас і спільні обов'язки, визначені суперкласами.

Фаза аналізу (RDD).

Під час другої фази RDD наступні заходи в першу чергу виконуються:

1) факторизація обов’язків в ієрархіях успадкування, для отримання максимально можливого повторного використання проектів класів; ієрархії успадкування моделюються в графах успадкування; обов'язки кожного класу

згруповані в контракти, список запитів, які клієнт може зробити з класу; клас може підтримувати численні контракти;

2) визначення можливих підсистем об’єктів і моделювання взаємодії між ними на рівні об'єктів більш детально; ця діяльність передбачає моделювання структури підсистем та їх вмісту, а також клієнт-сервер зв'язки між ними в графах співпраці; ці діаграми також показують контракти кожного сервера і відносини клієнт-сервер, що явно показують, від якого контракту сервера залежить клієнт;

3) визначення протоколів класів і завершення специфікації класів, підсистем класів і контрактів клієнт-сервер; протоколи визначаються для кожного класу шляхом уточнення обов'язків в наборах сигнатур методу; є докладні текстові характеристики написані для кожної підсистеми, кожного класу та кожного контракту.

2.3. Макро та мікрорівні моделювання в фреймворку

Перші дві дії виконуються ітеративно, оскільки рішення щодо меж підсистеми можуть вплинути на розподіл відповідальності, і навпаки.

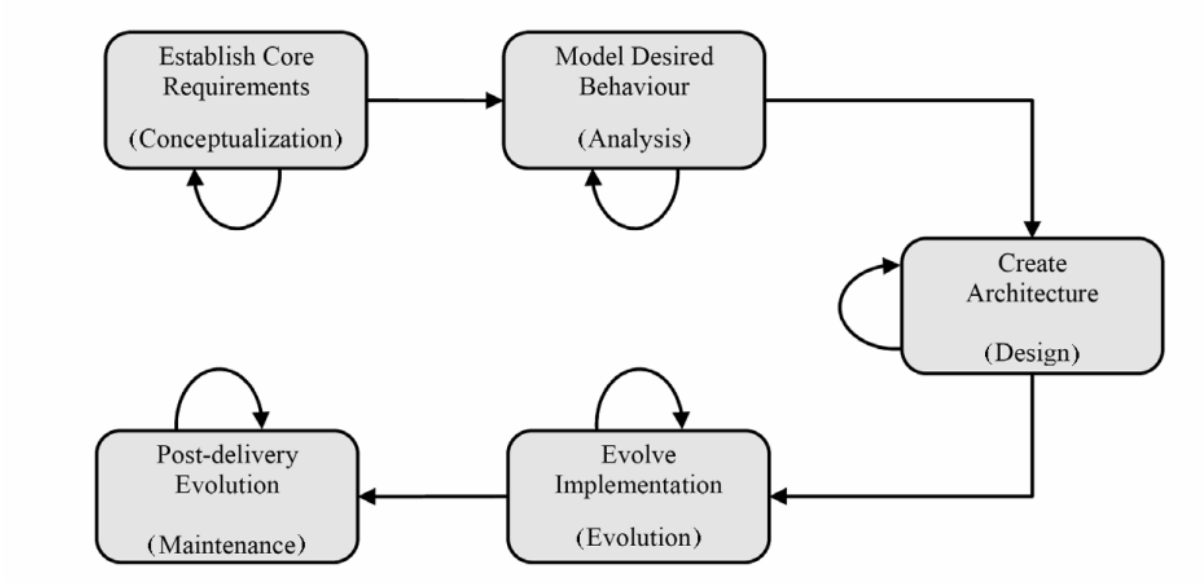

Рис. 2.9. Макрорівень процесу моделювання

Представлена об'єктно-орієнтована методологія, розглядається виключно як метод проектування. Макропроцес служить керуючою основою для мікропроцесу. Він представляє діяльність групи розробників у масштабі від тижнів до місяців. Багато частин цього процесу складають базові практики управління програмним забезпеченням, такі як забезпечення якості, покрокове керівництво кодом і документація. На цьому рівні більше уваги приділяється клієнтам та їх бажанням щодо таких речей, як якість, повнота та планування. На рисунку 2.9 показаний макропроцес, як це було запропоновано (самоітерації представляють мікропроцес). Мікропроцес керується сценаріями та архітектурними специфікаціями, які проходять в рамках макропроцесу. Він представляє щоденну діяльність індивіда або невеликої групи розробників.

Макропроцес має тенденцію виконувати такі кроки:

1) встановити основні вимоги до програмного забезпечення (концептуалізація);

2) розробити модель бажаної поведінки системи (аналіз);

3) створити архітектуру для реалізації (дизайн);

4) розвивати реалізацію шляхом послідовних вдосконалень (еволюція);

5) керувати еволюцією після доставки клієнту (технічне обслуговування).

Мікропроцес має тенденцію до наступного циклу діяльності :

1) Ідентифікація класів та об’єктів на заданому рівні абстракції шляхом встановлення меж проблеми, пошуку абстракцій у проблемній області, обмеження проблеми через визначення того, що цікавить, а що ні, і генерування словника даних, який визначає всі класи та об’єкти в розробці. Завдяки ітераційній природі мікропроцесу словник даних може змінюватися в процесі розробки. Рекомендується використання карт CRC протягом усього процесу. Класи, визначені на ранніх етапах макропроцесів в основному належать до проблемної області, а ті, що додаються під час дизайну зазвичай відносяться до реалізації.

2) Ідентифікація семантики класів і об'єктів, для встановлення значень класів та об’єктів, визначених на попередньому кроці, з головним наголосом на поведінку системи та її складових, а не структури. Це робиться шляхом встановлення поведінки та атрибутів кожної абстракції, визначеної в попередньому етапі, а також шляхом уточнення абстракцій.

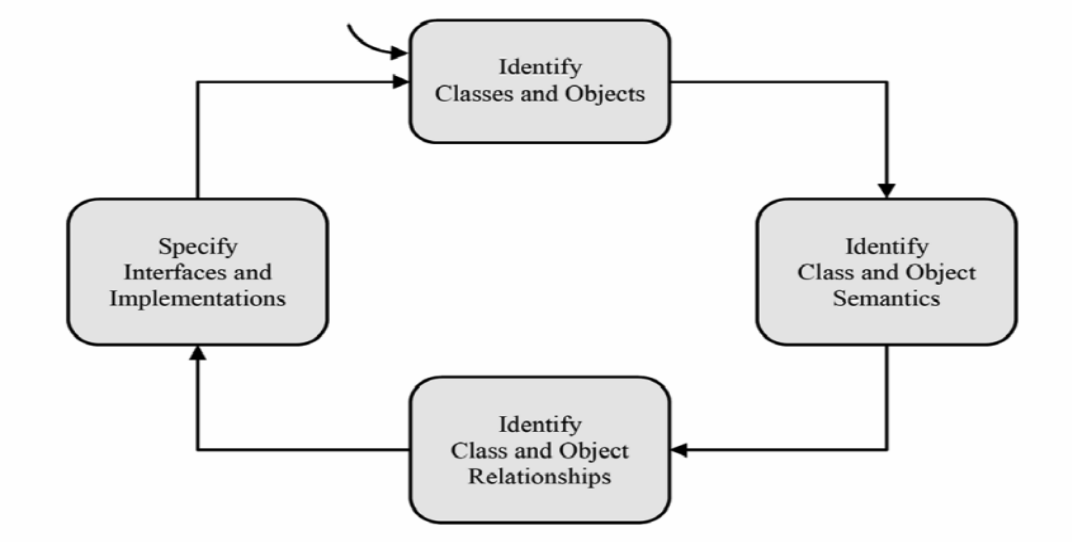

Рис. 2.10. Мікро рівень процесу моделювання

Обов'язки додаються абстракції та іменованим операціям розробленим для кожного класу. Стандартні діаграми виробляються для класів із значною динамічною поведінкою. Також створюються діаграми об’єктів і діаграми взаємодії, що зображують моделі взаємодії між ними на рівні об'єктів; ці діаграми фактично ізоморфні, причому перша підкреслює статику зв'язків між об'єктами, а остання підкреслює послідовність взаємодії між об'єктами. Діаграми об’єктів також дуже корисні на ранніх етапах макропроцесу для відображення структурних зв’язків між об’єктами. В у цьому контексті відображаються лише зв’язки між об’єктами; а стрілки та послідовності значень, які визначають поведінкові аспекти (передача повідомлень), додаються на наступних етапах. Ці прості об'єкти-діаграми будуються під час третього кроку мікропроцесу (виявлення зв'язків); іншими словами, прості об'єкти діаграми, побудовані під час третьої активності мікропроцесу на

попередніх стадіях проекту (перші дві фази макропроцесу), декоровані поведінковими деталями під час другої дії в наступних ітераціях.

3) Визначення зв’язків між класами та об’єктами: як тільки поведінка визначена, наступним кроком є визначення зв'язків між класами та об'єктами. Це здійснюється шляхом встановлення того, як саме взаємодіють речі в системі.

Шаблони класів, які дозволяють реорганізацію та спрощення структури класу ідентифікуються. У цей час приймаються рішення щодо видимості. Кінцевий результат цього кроку це виготовлення класових, об'єктних і модульних діаграм. Діаграми класів показують класи та їхні відносини (асоціація, успадкування та агрегація). Огляд об'єктно-орієнтованої розробки програмного забезпечення, орієнтований на діаграми процесів, як правило, будуються на етапах макропроцесу і використовуються для отримання фізичних модулів та взаємозалежностей між ними, зображуючи таким чином фізичну архітектуру системи.

4) Вказання інтерфейсу і реалізацію класів і об'єктів: проектні рішення приймаються щодо представлення вже визначених класів і об’єктів. Класи та об’єкти розподіляються за модулями та процесами, які їх реалізують, модулі розподілені по процесам. Декоровані схеми модулів виготовляють деталі та технологічні схеми. Схема процесу показує апаратне забезпечення архітектури платформи системи шляхом зображення процесорів і пристроїв та їх взаємозв'язків. Вона також показує, які процеси призначені для кожного процесора.

Як правило, акцент поступово зміщується з попередніх дій мікропроцесу на останні, коли проект просувається через макропроцес, від концептуалізації до аналізу, а потім до дизайну. Однак через ітеративний характер загального процесу, ймовірно, що попередні дії мікропроцесу будуть переглянуті по всьому дизайну.

Три моделі, за допомогою яких OMT графічно визначає систему:

1) Об’єктна модель (OM) є основною моделлю. Вона зображує класи об'єктів у системі і їхні зв’язки, а також їхні атрибути та операції, і таким чином представляє статичну структуру системи. Модель об'єкта представлена графічно
за діаграмою класів.

2) Динамічна модель (DM) вказує на динаміку об’єктів та їх зміни станів. Вона фіксує суттєву поведінку системи, досліджуючи поведінку об’єктів в часі та потік керування для подій між об’єктами. Сценарії потоку подій відображаються на діаграмах трасування подій. Ці діаграми разом з діаграмами переходів станів (діаграми станів), складають динамічну модель OMT.

3) *Функціональна модель* (FM) – це ієрархічний набір діаграм потоків даних (DFD) системи, що описує її внутрішні процеси, не звертаючи уваги на те, як вони фактично виконуються.

Кожна модель описує один аспект системи, але містить посилання на інші моделі: об'єктна модель описує структуру даних, яка є динамічною та функціональною. Моделі працюють на операції в об’єктній моделі і відповідають подіям у динамічній моделі та функціям у функціональній моделі; динамічна модель описує структура управління об'єктами, що показує рішення, які залежать від значень об'єктів і дії, що викликаються, що змінюють значення об’єктів і викликають функції; функціональна модель описує функції, викликані операціями в об’єктній моделі та діями в динамічній моделі; функції оперують значеннями даних, заданими об'єктною моделлю; функціональна модель також показує обмеження на значення об’єктів.

Процес OMT складається з п’яти етапів, як показано на рисунку. Перші три етапи (аналіз, проектування системи та проектування об’єктів) вважаються основними функціями процесу OMT.

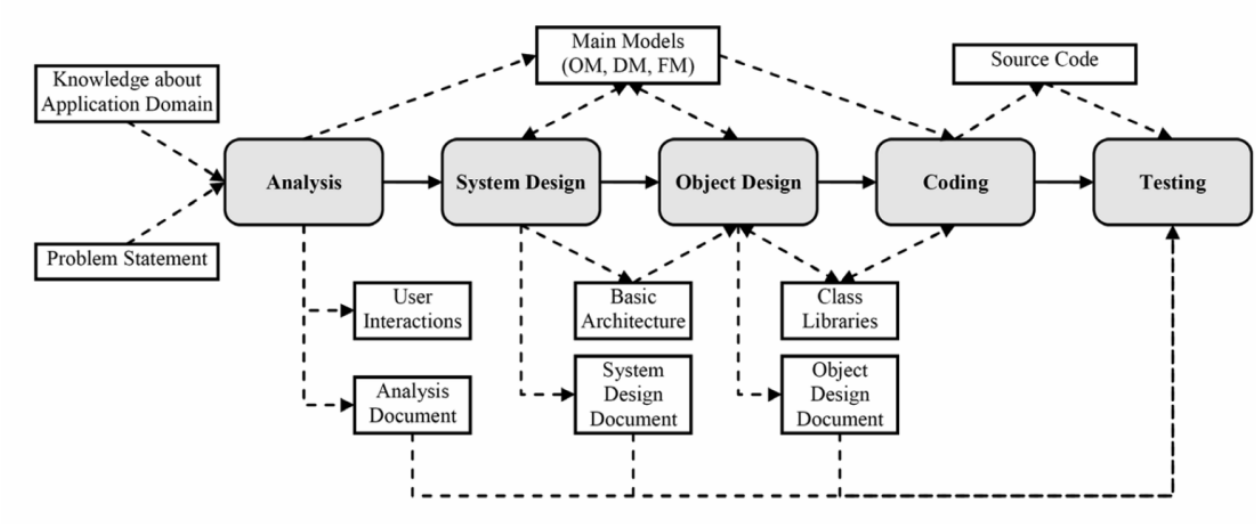

Рис. 2.11. Аналіз структури процесу моделювання

Мета аналізу – побудувати правильну і зрозумілу модель реального світу. Вимоги користувачів, розробників і менеджерів надають інформацію, необхідну для розробки початкової постановки проблеми. Як тільки початкова проблема визначена, виконуються наступні завдання:

1) побудувати об’єктну модель, включаючи діаграму класів, що зображує класи системи і їхні зв'язки, а також словник даних;

2) розробити динамічну модель, включаючи діаграми переходів станів і глобальні діаграми трасування подій; OMT визначає наступні етапи побудови динаміки моделі;

3) визначити шаблони використання системи та підготувати сценарії типової послідовності взаємодії;

4) ідентифікувати події серед об’єктів і підготувати діаграму трасування подій для кожного сценарію;

5) підготувати діаграму трасування подій для системи, що показує події, що протікають у межах системи;

6) розробити діаграми переходів станів для класів з понеційною динамікою поведінки;

7) перевірка узгодженості та повноти подій, які розподіляються між діаграмами переходів станів;

8) побудувати функціональну модель, включаючи діаграми потоку даних і обмеження;

9) перевірити, переглянути та вдосконалити отримані моделі.

Під час проектування системи із структурами високого рівня вибирається конфігурація системи. Рішення, які будуть прийняті під час проектування системи:

1) організація системи в підсистеми;

2) визначення паралельності;

3) розподіл підсистем між процесорами та завданнями;

4) вибір стратегії реалізації сховищ даних з точки зору структур даних, файлів та бази даних;

5) визначення глобальних ресурсів і визначення механізмів контролю доступу до них;

6) вибір підходу до реалізації програмного контролю;

7) врахування граничних умов;

8) встановлення компромісних пріоритетів.

Висновки до розділу

Отже, в цьому розділі представлено структуру концепції об’єднання доменів при розробці комплексного фреймворку Також в розділі визначено, що об'єктний дизайн стосується повної специфікації існуючих і інших класів, асоціацій, атрибутів і операцій, необхідних для впровадження системи. Операції та структури даних повністю визначені разом з будь-якими внутрішніми об'єктами, що необхідні для реалізації. По суті, всі деталі повністю визначені до того, як буде реалізована система і уточнюються під час проектування об'єкта.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ СИСТЕМНИХ МОДЕЛЕЙ РІВНЯ КОМПЛЕКСНИХ ФРЕЙМВОРКІВ

3.1. Дослідження елементів фрейморку для побудови доменів координації

Фреймворк складається з трьох моделей, інформаційної системи (ІС), стабілізуючого ядра та стратегії побудови домену. Кожен елемент є похідним від однієї або кількох координаційних складових. Фреймворк розроблено канонічним способом і це означає, що елементів у структурі має бути якомога менше, але все ж охоплювати всі складові координації. Це пояснює, чому у Framework є три моделі, а не, наприклад, дві або чотири. В принципі, можна уявити інші структури застосування фрейморку.

Фреймворк слід розглядати як узгоджене ціле в тому сенсі, що всі елементи є важливі для побудови координаційних областей. Термін «фреймворк» стосується того факту, що елементи фреймворку та їх взаємодія визначені. Проте конкретна реалізація цих елементів може бути замінена на інші реалізації, що відповідають тим самим специфікаціям. Наприклад, інформаційна система eMatrix, яка використовується в фрейморку, може бути замінена іншою інформаційною системою за умови збереження еволюційного способу роботи, властивого фреймворку.

Термін «конструкція» має два значення. По-перше, Framework призначений для роботи в організаціях, що розробляють систему. Це означає, що елементи мають бути створені в певному середовищі учасниками сфери координації. По-друге, «конструювання» відноситься до соціального аспекту координації. Те, що вважається реальністю щодо координації, виникає в результаті соціальної взаємодії між акторами в області координації. Це не означає, що координація може мати будь-яку форму. Навпаки, значення координації обмежується конкретними культурними, історичними, організаційними та технологічними обставинами. Результатом застосування

Framework буде домен координації, який включає спільне значення між учасниками, екземпляри моделей та інстанційовану підтримку IS.

Існує дві сфери діяльності, залучені до побудови координації. Одним з них є область використання, яка створює, наприклад, телекомунікаційні системи шляхом координації низки трансформаційних дій. Інший домен є доменом координації, який створює «координацію», яка використовується в домені використання. Це означає, що моделі та інформаційна система з'являються як у сфері використання, так і в області координації, але в різних ролях. У сфері координації вони є робочим об'єктом і результатом діяльності. У сфері використання вони є засобами для координації діяльності для отримання іншого результату, такого як телекомунікаційна система.

Елементи Framework визначаються таким чином:

• Концептуальна модель означає структуру координаційної області з точки зору елементів координації та їхніх статичних зв'язків. Одним із прикладів цього є те, що «вимоги розподілені_до елементів дизайну». Концептуальна модель є похідною від контекстуальності та складових орієнтації в теорії сфери діяльності.

• Модель процесу вказує на залежності між видами діяльності, що впливають на елементи координації, наприклад, що «пріоритезація вимог» стоїть перед «розподілом вимог». Ця модель відповідає визначенню координації, даному в [5]. Модель процесу є похідною від тимчасової складової в теорії сфери діяльності.

• Модель переходу вказує на те, як область координаційної діяльності взаємодіє з іншими областями діяльності, тобто як елементи координації інтерпретуються та перекладаються, коли вони з'являються в інших контекстах. Наприклад, з усієї інформації, створеної в контексті розробки програмного забезпечення, лише частина є важливою в домен координації, наприклад, коли файл вихідного коду змінив свій статус або перегляд. Модель переходу є похідною від компонента переходу домену в теорії домену діяльності.

• Моделі реалізовані в інформаційній системі (ІС) і випробувані в області використання. Інформаційна система є похідною від компонента використання інструментів у теорії сфери діяльності.

• Стабілізуюче ядро складається з правил, норм, стандартів тощо, які забезпечують необхідну стабільність у домені. Стабілізуюче ядро походить від стабілізуючого основного компонента в теорії області діяльності. Одним із прикладів є «Корпоративні базові стандарти», які є набором правил ідентифікації та контролю версій продуктів і документів.

• Для того, щоб досягти спільного значення щодо домену координації, до фреймворку включено стратегію побудови домену. Ця стратегія складається з безперервної ітерації між фазами роздумів і діями, під час яких моделі та IS альтернативно відображаються та випробовуються на практиці. У цьому процесі IS постійно змінюється. Таким чином, використовується еволюційний підхід до розробки інформаційної системи. Стратегія побудови домену є похідною від складової експериментального навчання в теорії домену діяльності. На рисунку 3.1 проілюстровано елементи фреймворку.

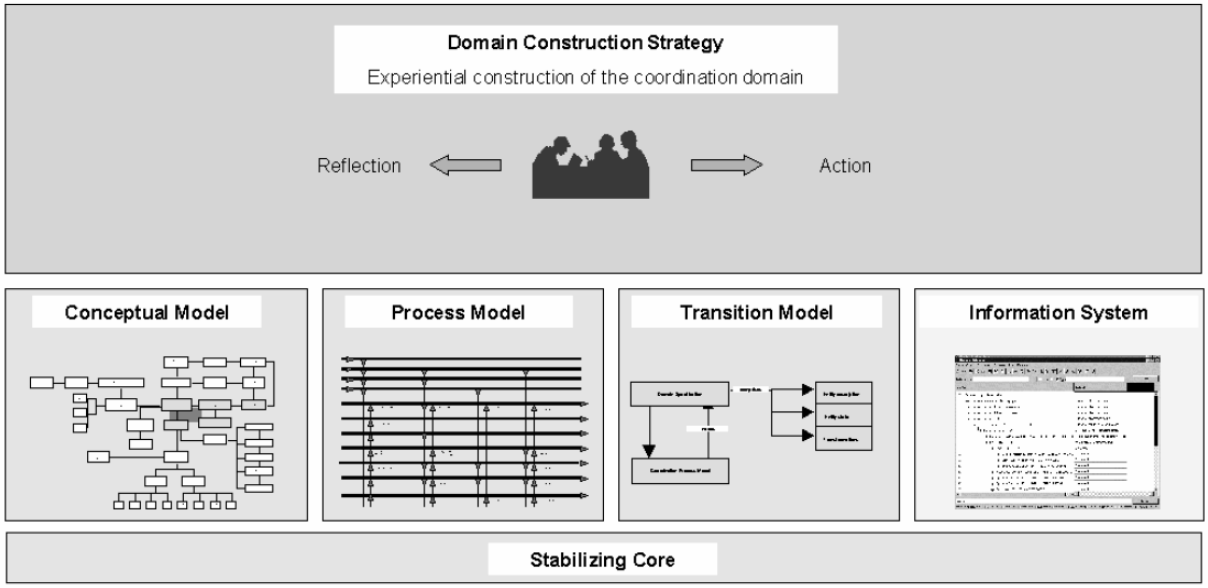

Рис. 3.1. Елементи фреймворку

Разом ці елементи пов’язані, як показано на концептуальній карті (рис. 3.2)

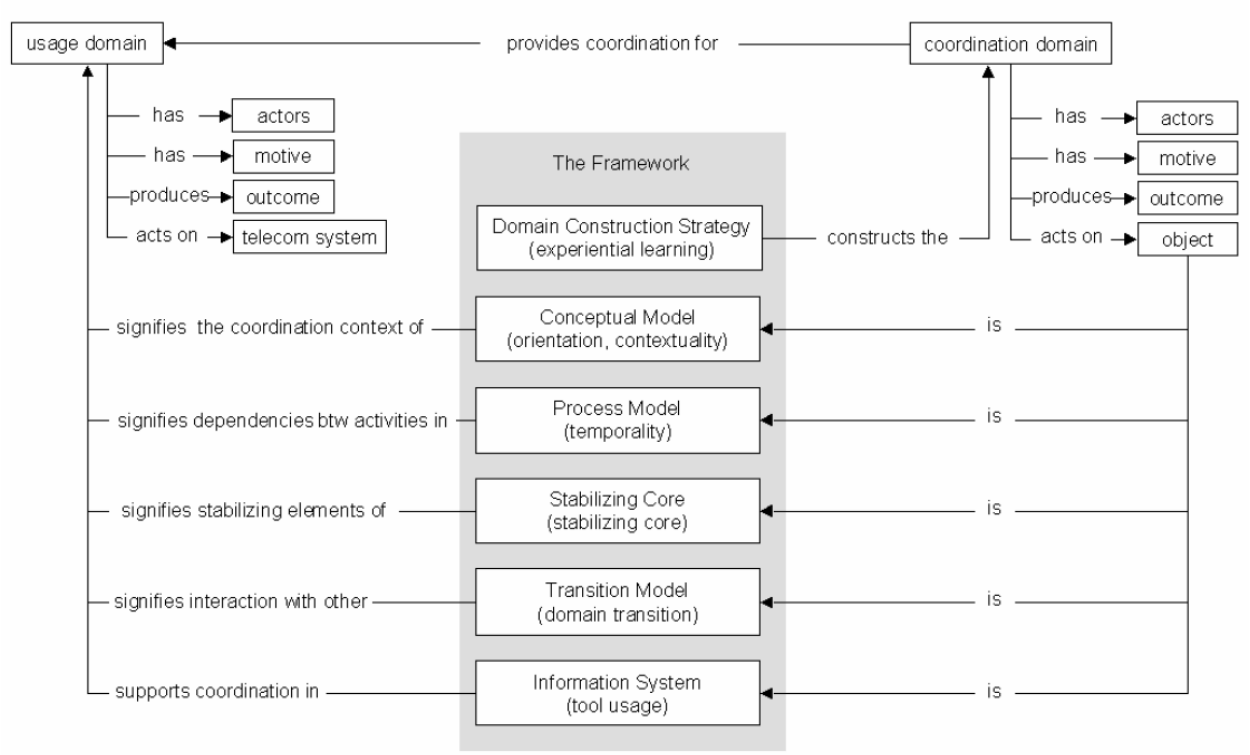

Рис. 3.2. Концептуальна карта фреймворку та її елементів

У наступних розділах ці елементи детально розглядаються.

3.2. Концептуальна модель та моделі процесу і переходу

3.2.1. Опис концептуальної моделі

Окрім позначення орієнтаційної структури координаційної області, концептуальна модель також розмежовує контекст координаційної області. Пункти, які не входять до концептуальної моделі, вважаються незначними учасниками. Знову ж таки, важливо пам’ятати, що структура та обсяг сфери координації є предметом згоди між учасниками.

Концептуальна модель служить двом цілям. По-перше, це спільний знак для учасників у сфері координації, а по-друге, він визначає, що має бути реалізовано в інформаційній системі. Зазвичай найважче досягти першого аспекту. Номенклатура моделі, яка використовується в концептуальній моделі, може відрізнятися. Однак найважливішим питанням у визначенні номенклатури є розуміння. Актори повинні якомога легше інтерпретувати

модель. З цією метою можна використовувати методику моделювання об’єктів, наприклад, техніку моделювання об’єктів (Rumbaugh et al., 1991). Це також має ту перевагу, що модель легко реалізується, якщо інформаційні системи є об’єктно-орієнтованою інформаційною системою, як у випадку з eMatrix.

3.2.2 Модель процесу

Існує кілька можливих класів моделей, які можна використовувати як модель процесу. Найважливішим критерієм для їх оцінки для використання в рамках є їхня точність у позначенні залежностей між видами діяльності. Відомо, що діяльність взаємопов’язана з елементами координації. Тому важливо, щоб природа цих відносин була зрозуміла акторам. На рисунку 3.3 показано деякі моделі процесу.

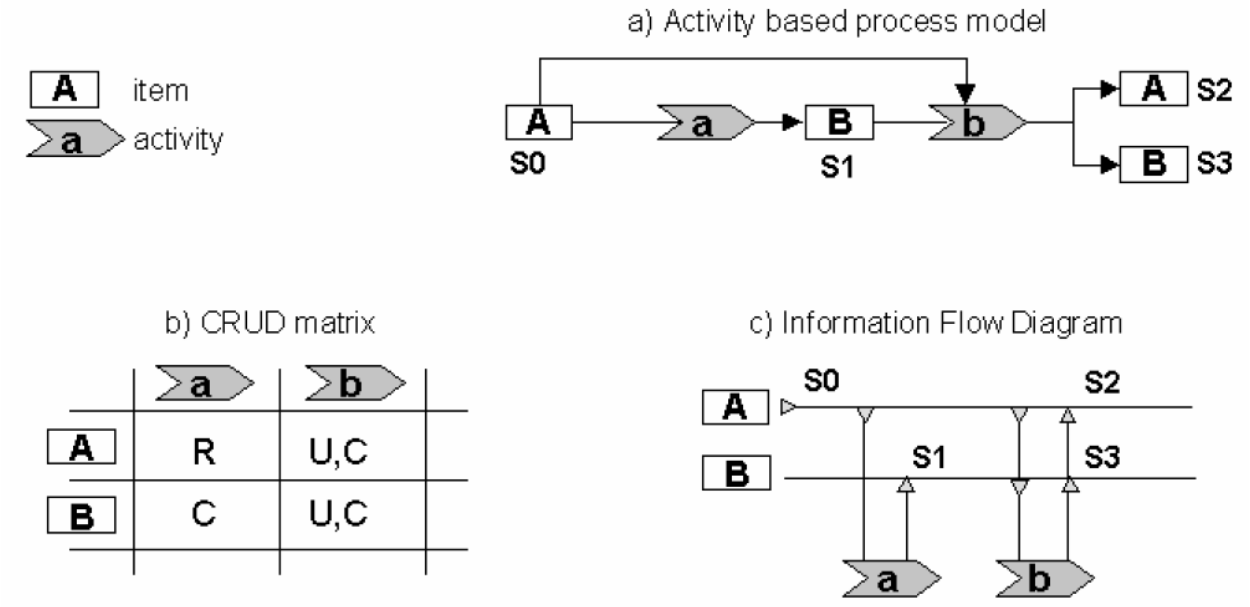

Рис. 3.3. Альтернативні моделі процесу

Найбільш часто використовуваним класом моделі є а), у якому дії знаходяться у фокусі, а елементи розміщені на задньому плані. Цей клас моделей процесів добре підходить для моделювання ситуацій, коли залежності між видами діяльності в основному послідовні. Він менш

придатний для моделювання паралельних видів діяльності. Елементи розкидані в моделі і прогрес стану предмета важко відстежити. Наприклад, елемент А на рис. 3.3 а) з’являється в двох непересічних місцях моделі. Крім того, контроль прогресу в цьому класі моделей зазвичай здійснюється шляхом зміни стану дій, наприклад «очікування введення», «виконання», «готовий» тощо, а не за станом елемента (S0, S1 , S2 і S3 на рисунку). Останній недолік цієї моделі полягає в тому, що важко побачити схему взаємодії між предметами та видами діяльності.

Шаблон взаємодії знаходиться в центрі уваги для іншого класу моделей процесів: матриці CRUD (Create, Read, Update, Delete), показаної на рисунку 3.3 b). Матриця CRUD фіксує, які дії впливають на які елементи та характер цього впливу (створення, читання, оновлення, видалення). Залежності між елементами та діяльністю більш очевидні в матриці CRUD, ніж у моделях на основі діяльності. Однак немає вказівок на стани елементів, що означає, що шаблону взаємодії в матриці CRUD бракує важливого аспекту. Крім того, форма матриці ускладнює читання матриць CRUD, тобто точність позначення означника (матриця CRUD) є низькою для акторів.

Останнім класом розглянутих моделей процесу є діаграми інформаційних потоків (IFD - Information Flow Diagrams). Ця модель складається з трьох елементів: набору елементів, набору дій і діаграми, яка показує, як вони взаємодіють. Елементи представлені горизонтальними лініями, які можна сприймати як проекцію відповідної концептуальної моделі, де фактичні зв’язки між елементами пригнічуються. Діяльність показана внизу. Вертикальні стрілки на діаграмі, спрямовані вниз, вказують на вхідні дані для діяльності, тоді як вертикальна стрілка, спрямована вгору, вказує на результати діяльності. Там, де спрямована вгору стрілка стикається з горизонтальною лінією, відповідний елемент змінює свій стан. Це означає, що діаграма в моделі IFD показує залежності між видами діяльності, що прямо відповідає визначенню координації.

Діаграми інформаційних потоків належать до певного класу моделей процесів, які називають моделями на основі об’єктів. Форма, яка використовується в IFD, схожа за своєю природою на моделі, що використовуються в диференціальному аналізі з використанням аналогових комп’ютерів. Тут змінні були намальовані у вигляді ліній з інтеграторами, що відповідають видам діяльності [15]. Моделі, засновані на об’єктах, придатні в ситуаціях динамічного розвитку, коли прогрес стану координаційних елементів є досить стабільним, а дії та інструменти використовуються досить нерегулярно для отримання результату.

У фреймворку використовується клас моделей IFD. Знову ж таки, основною причиною такого вибору є прозорість моделі щодо спільного розуміння акторів. Точність позначення в моделях цього типу вища, ніж в інших класах моделей. На рисунку 3.4 показано приклад IFD для процесу запиту на зміну.

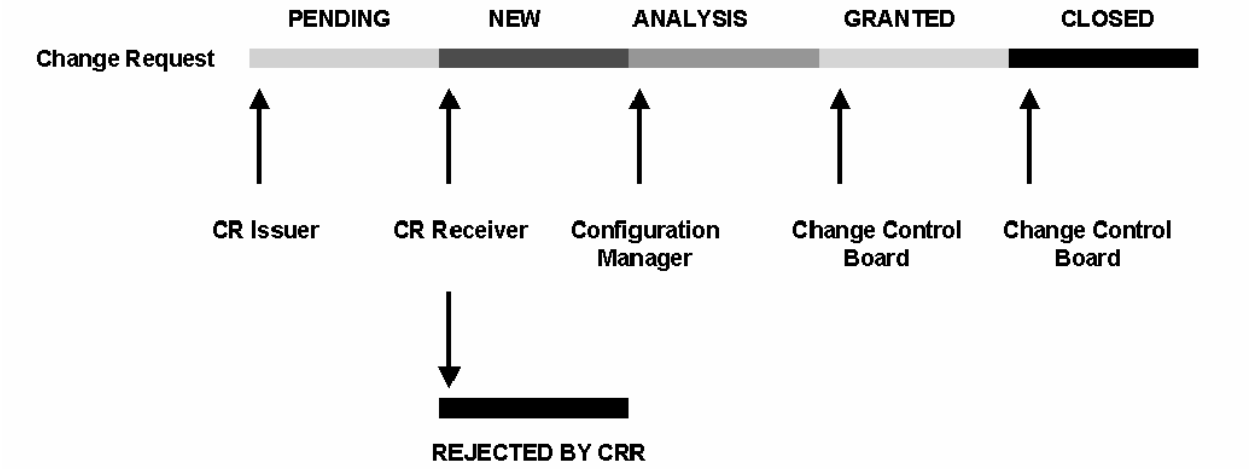

Рис. 3.4. Модель процесу запиту на зміну в нотації IFD

3.2.3. Модель переходу

Щоб прояснити значення моделі переходу, припустімо, що ми маємо одну область діяльності C, яка забезпечує деякий результат O з деякої початкової бази I. Крім того, це робиться у співпраці з двома іншими

областями, A і B (рис. 3.5). Навколишнє середовище C може розглядатися як сфера діяльності, структура якої загалом відрізняється від C. Це означає, що результат O розглядається по-різному залежно від того, в якому домені він з'являється, що, у свою чергу, означає, що необхідно виконати перетворення між відповідними елементами всередині та поза C.

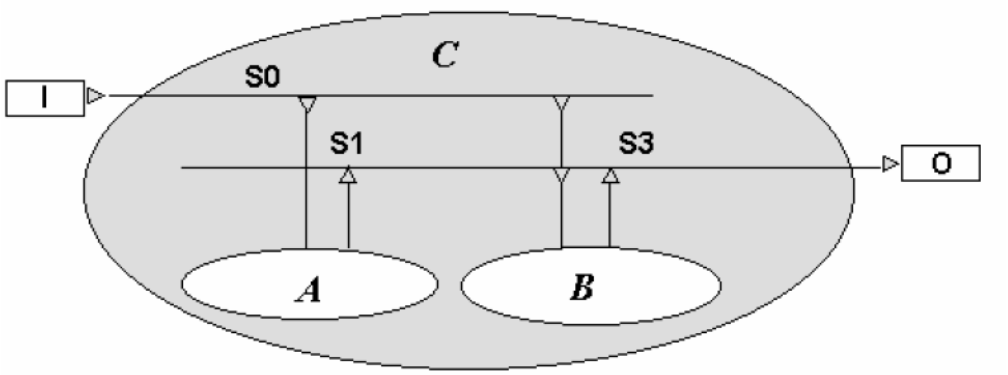

Рис. 3.5. Взаємодіючі домени діяльності

Крім того, як показано на рисунку 3.5, результати двох взаємодіючих доменів A і B повинні бути скоординовані, щоб отримати результат O. Для цих доменів C представляє середовище, і подібний перехід повинен бути здійснений між елементами в C і елементи інтер'єру в A і B. Таким чином, ми бачимо, що той самий шаблон повторюється незалежно від того, які домени взаємодіють.

Загалом, перехід між двома доменами може впливати на будь-який тип елементів, таких як знаки, мова, норми тощо. З метою координації ми зосередимося на управлінні залежностями між видами діяльності. Необхідно розглянути два аспекти: як трансформуються вхідні дані та вихідні дані діяльності та як зафіксувати повторюваний шаблон, який обговорювався вище.

З цією метою ми будемо використовувати специфікаційну модель даних (SBDM – Specification Based Data Model) [16]. Ця модель в основному має дві конструкції: «специфікацію» та «реалізацію». Вони пов'язані двома відносинами: «специфікація» реалізується за допомогою «реалізації», а

«реалізація» потребує «специфікації» (рис. 3.6). Таким чином ми бачимо, що повторення є невід'ємною функцією.

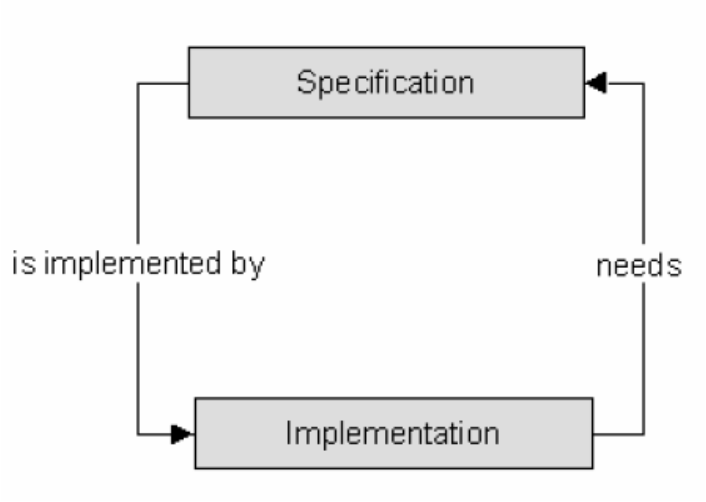

Рис. 3.6. Модель даних на основі специфікації в SBDM

Для доменів діяльності SBDM можна інтерпретувати таким чином: «специфікація» фіксує «вторгнення» домену діяльності в інший домен, наприклад, як С виглядає в його середовищі. Це еквівалентно тому, що С задуманий як «чорний ящик» у середовищі, де характеристики «коробки» С фіксуються сутністю «специфікації» в SBDM.

Якщо ми розглядаємо конкретну сутність у вхідних або вихідних даних для діяльності, необхідно трансформувати два елементи: опис сутності та стан сутності. У SBDM цей елемент пов’язаний із конструкцією «специфікація». Прикладом перетворення опису є те, що продукти ідентифікуються за назвою та версією. Під час керування розробкою програмного коду, що належить до певного продукту, це робиться в іншій області діяльності, де використовується інструмент конфігурації програмного забезпечення. Цей інструмент використовує «гілки» та «вузли» для визначення конкретної версії цього коду. У цьому випадку необхідно виконати перетворення між (назва, версія) і (гілка, вузол). Подібним чином спосіб вираження станів у інструменті конфігурації програмного забезпечення має бути перетворений у стани продукту.

Інший аспект переходу домену стосується повторення. У SBDM це досягається за допомогою конструкції «реалізація», яка пов’язує специфікації через два відносини «реалізовано» та «потреби». Для цілей координації ми можемо інтерпретувати «впровадження» як координацію внесків сфер спільної діяльності в результат. У Framework координація моделюється моделлю процесу. Таким чином, сутність «впровадження» в SBDM є саме цією моделлю процесу.

Це означає, що модель переходу можна проілюструвати, як показано на рисунку 3.7. Специфікація домену містить описи вхідних і вихідних об’єктів, їхні стани та те, як вони перетворюються між внутрішнім і зовнішнім доменом. Сама трансформація може бути виражена різними способами, наприклад у вигляді правил або функцій. Специфікація домену реалізується взаємодіючими доменами процесу.

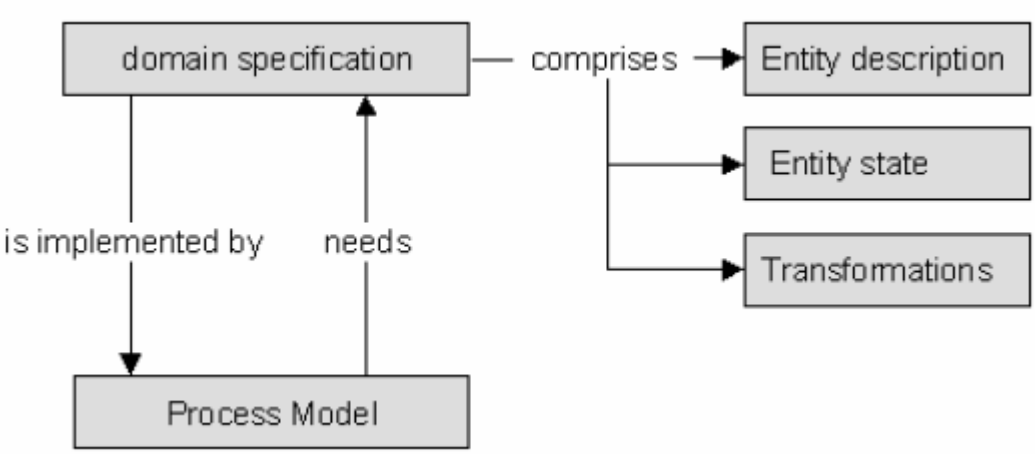

Рис. 3.7. Модель переходу, яка координується відповідно до моделі

Необхідні домени для співпраці в свою чергу визначаються специфікаціями домену. Цей спосіб моделювання співпраці між областями діяльності має кілька приємних особливостей. Він забезпечує відстеження в будь-якому шаблоні доменів, що співпрацюють, незалежно від розміру та рівня доменів. Крім того, це забезпечує свободу для різних сфер діяльності

розвивати власну структуру з точки зору мови, норм, стандартів, традицій тощо, не повертаючись до ізольованих острівців. Нарешті, це відповідає загальновідомому принципу поділу завдань, щоб справлятися зі складними системами.

3.3. Представлення інформаційної системи структуризації доменів

Метою інформаційної системи (ІС) є підтримка завдання координації в предметній області. Це означає, що концептуальна модель, модель процесу, модель переходу та стабілізаційне ядро мають бути можливими для реалізації в інформаційній системі. Наслідком такої позиції є те, що існує лише одна інформаційна система, яка підтримує завдання координації в фрейворку. Крім того, ІС повинна взаємодіяти з іншими ІС у взаємодіючих областях діяльності. Це означає, що трансформації, змодельовані в моделі переходу, також повинні бути реалізовані в інформаційній системі, а також у інтерфейсних ІС.

Крім того, можна буде використовувати інформаційну систему як інструмент у Стратегії побудови домену, що означає, що модифікація реалізації моделей має бути дуже легкою та простою. На додаток до цих вимог, інші більш традиційні вимоги до інформаційної системи, такі як продуктивність, глобальне розповсюдження, безпека даних, веб-доступ тощо, повинні бути розглянуті. Загалом це висуває дуже суворі вимоги до властивостей інформаційної системи.

Оскільки інформаційна система є похідною від компонента використання інструменту в області діяльності, саме структура та еволюція області використання є рушійною силою еволюції інформаційної системи. У цій еволюції необхідно враховувати всі елементи фреймворку, а також їх взаємодію.

Інформаційна система означає важливі для акторів явища в області використання. Отже, важливі властивості інформаційної системи. Одним із

прикладів є те, що означальники, такі як піктограми, повинні бути однаковими в моделях та в інформаційній системі, щоб уникнути тягаря для акторів вивчати кілька означників, що позначають ті самі явища.

Так само, як можна використовувати різні номенклатури для моделей Framework, можна використовувати будь-яку ІС, якщо вона відповідає вимогам, зазначеним вище. У практиці ми використовуємо eMatrix від Matrix-One Inc., яка є об'єктно-орієнтованою системою керування даними про продукт (PDM - Product Data Management). Ця ІС має всі необхідні властивості, перш за все, простоту зміни реалізації. На рисунку малюнку 3.8 схематично показана процедура впровадження.

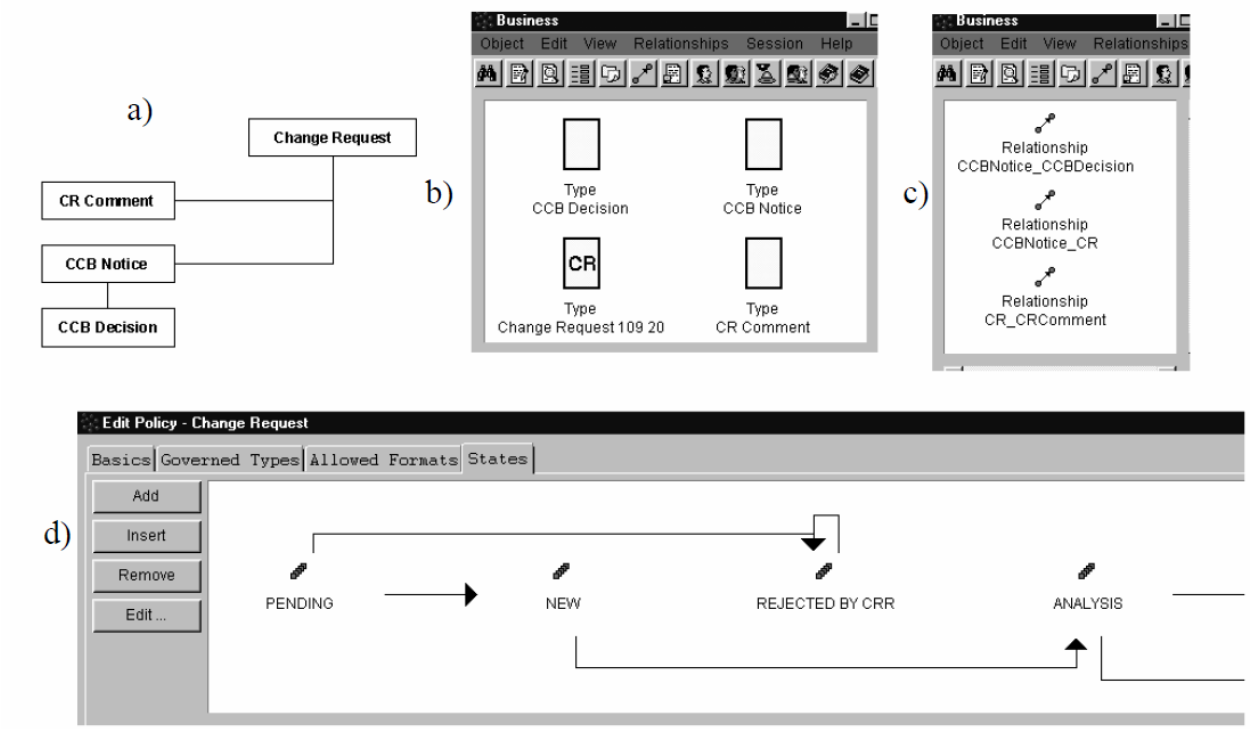

Рис. 3.8. Реалізація моделей в інформаційній системі

Кожен позначений елемент у концептуальній моделі a) реалізовано як тип в eMatrix b). Це включає атрибути, імена, операції тощо, пов'язані з елементом. Залежності між означеним елементом у концептуальній моделі реалізовано як зв'язки в eMatrix c). Крім того, визначено кардиналіти відношення (скільки елементів може бути пов'язано на кожній стороні

відношення) і правила поетапного перегляду (що станеться, коли буде створено нову ревізію елемента), а також можливі атрибути для відношень.

Модель процесу реалізована як ланцюги станів, які в eMatrix називаються політиками d). Політика є еквівалентом лінії діаграми потоку інформації в моделі процесу. Політика на рисунку 3.8 відповідає моделі процесу.

Для реалізації моделі переходу можуть використовуватися різні механізми. Наприклад, правила можуть використовуватися для перетворення значень стану в одному ланцюжку станів у значення в іншому. Можна визначити погляди, які зосереджені на певному контексті. У фактичних реалізаціях, найбільш помітним вираженням моделі переходу є так звана анатомія. Прикладом анатомії є структура робочого пакета, зображена на рисунку 1.2.

Реалізація моделей в основному здійснюється безпосередньо в інтерфейсі користувача eMatrix. Точніше, це робиться в Business Modeller в eMatrix. Розширені додатки програмуються на мовах програмування, в тому числі на інтерпретаційній (TCL). Це означає, що досить простий набір моделей, як показано на рисунку 3.8, реалізується за лічені хвилини. Крім того, реалізація виконується без зупинки бази даних, що означає, що реалізація може бути змінена в режимі онлайн, наприклад, під час поточної сесії. Це ключ до експериментального способу навчання у Framework. Якби впровадження зайняло б більше часу, скажімо, півгодини для зміни атрибута, і базу даних потрібно було закрити, експериментальний спосіб роботи був би неможливим.

Стабілізуюче ядро — це загальний термін для елементів у сфері діяльності, які мають трансіндивідуальний характер, тобто вони існують лише через діяльність індивідів, але не залежать від конкретних індивідів. Такі елементи можуть включати форми, правила, процедури, конвенції, стратегії, норми та технології. Він також включає в себе структури переконань, культурні питання, спільні значення тощо.

Основним стимулом є те, що ступінь ядра впливає на здатність домену використання адаптуватися до змін. Стабілізуюче ядро це здатність перешкоджає, якщо ядро стає занадто великим або занадто малим. Регулюючи баланс, можна знайти оптимальну здатність адаптуватися до змін.

3.3.1.Стратегія побудови домену

Мета стратегії побудови домену полягає в тому, щоб надати суб'єктам у сфері координації стратегію того, як її побудувати. Ця стратегія складається з безперервної ітерації між фазами роздумів і діями, під час яких моделі та ІС альтернативно відображаються та випробовуються на практиці. Це означає, що ми розглядаємо координаційну область як істоту в стані постійної еволюції. Зміни можуть бути більш-менш очевидними, але, незважаючи на це, вони завжди присутні.

Важливим аспектом стратегії побудови домену є те, що в результаті стратегії виникає спільне розуміння щодо домену координації. З цією метою моделі та інформаційна система задумані як знаки, які набудуть спільного значення під час застосування стратегії. Для цього вживаються такі заходи:

• Навчання на досвіді саме по собі є фундаментальним способом для людей створювати значення в контексті, де вони живуть і діють.

• Кількість суб'єктів у сфері координації має бути якомога меншою, але при цьому включати всі відповідні аспекти сфери координації. Наприклад, якщо область координації включає вимоги як елементи координації, координатор вимог повинен бути одним із учасників. Іншими учасниками можуть бути експерт з ІС та керівник проекту, чиє завдання полягає у розподілі вимог до поставок проекту тощо.

• Номенклатуру, яка використовується в моделях, слід вибирати з точки зору розуміння, а не формальності. Слід уникати номенклатури, спрямованої на формальну специфікацію. Наприклад, хоча номенклатура UML широко прийнята як стандарт для об'єктно-орієнтованого моделювання, вона менш

підходить у цьому контексті, оскільки вона зазвичай не знайома учасникам у координаційній області.

• Позначники, що використовуються в моделях і ІС, наприклад значки, повинні бути ідентичними, якщо вони позначають ті самі явища.

Отже, метою Концепції є операціоналізація концепції координації як сфери координації, запропонованої в цьому дослідженні. Область координації позначається концептуальною моделлю, моделлю процесу, моделлю переходу та стабілізуючим ядром.

Моделі реалізовані в інформаційній системі, яка використовується для підтримки координації. Область координації побудована за допомогою стратегії експериментального навчання, що означає, що моделі, їх імплементація в інформаційній системі та спільне значення виникають одночасно діалектичним способом (рис. 3.9).

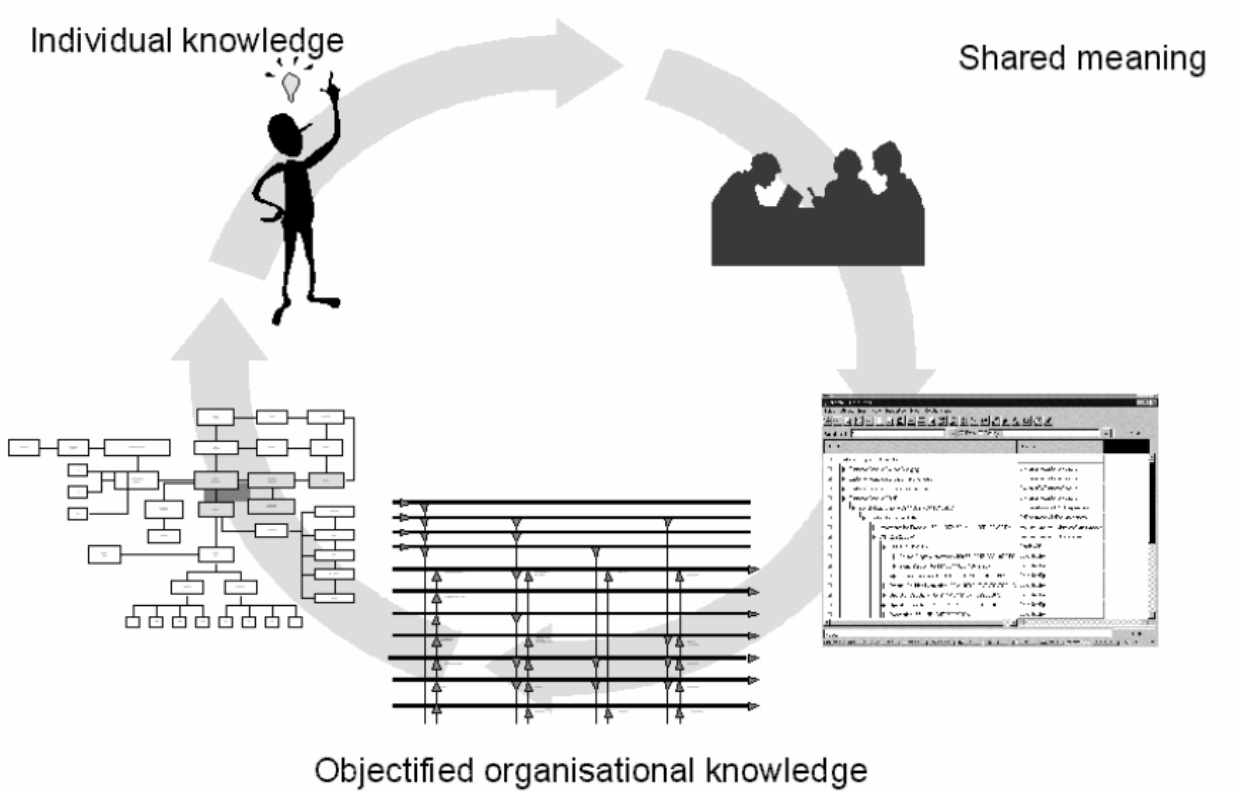

Рис. 3.9. Результати стратегії побудови домену

Працюючи з моделями та інформаційною системою, вони набувають значення для актори; тобто моделі та інформаційна система стають значущими для акторів.

3.4. Застосування методів та моделей для комплексних фреймворків інформаційних систем

Головною проблемою протягом використання комплексного фреймворку в практику був обсяг інформаційної системи. Чи має бути одна ІС для всього домену координації чи декілька? Це питання, яке межує з цілим комплексом питань (таблиця 3.1).

Таблиця 3.1.

Категорія наслідків – Архітектури інформаційних систем

Effect category	description	consequence	#
corporate IS/IT,	Issues regarding corporate IS/IT	The information system architecture suggested by the Framework did not comply with the Corporate IS/IT strategy. One consequence of this was a number of conflicts.	7
global,	Global aspects	The ISA used in the Framework was a central server with thin web-clients. This is a classical IRM architecture. However, this presumes thin clients for capacity reasons.	8
infrastructure,	Issued related to infrastructure like the Intranet, routers, PC's, etc.	The infrastructure was insufficient for the ISA used in the Framework (the IRM architectures) until the Lean Client was introduced.	3
IS architecture,	The role of eMatrix in relation to other IS	The management of all coordination items will be done in eMatrix. There are interfaces toward other IS, for example, ClearCase.	11
IS federations,	Issues regarding one or several databases for eMatrix	A number of eMatrix environments may be arranged in a federal architecture which balances the need for central control with local autonomy.	16
local development,	refers to local development of ADs (esp. tools) without coordination	Local development of a coordination is possible in a federated architecture.	8
one management tool,	Issues related to one tool (eMatrix) to support coordination	The consequence of 'one management tool' is that the ISA will be structured in such a way that eMatrix will be the sole tool in the coordination domain.	23
separation of concerns,	Categories related to hiding unimportant phenomena in a context	Separation of concerns is possible in a federated ISA	20
stabilizing core,	The stabilizing core of an activity domain	A federated ISA implies that there must be a stabilizing core which is mandatory and common to all federations.	4

Архітектури інформаційних систем (ISA - Information system architectures) широко обговорюються в літературі. Зокрема, обговорювався зв'язок між стратегічним плануванням та бізнес-стратегіями [41.43]. Тут ми зосереджуємося на ISA у зв'язку з завданням координації, яке опосередковано пов'язане з бізнес-стратегіями через систему, яка буде розроблена, виготовлена та продана на ринку.

3.4.1. Традиційна архітектура

Якщо ми візьмемо концептуальну модель, показану на рисунку 3.10, традиційна ISA базується на принципі: одна ІС на ситуацію координації (овали на рисунку). Тобто одна ІС для керування вимогами, одна для керування запитами на зміни, одна для керування тестуванням тощо. Це архітектура інформаційної системи, яку використовують багато організацій, що розробляють продукти.

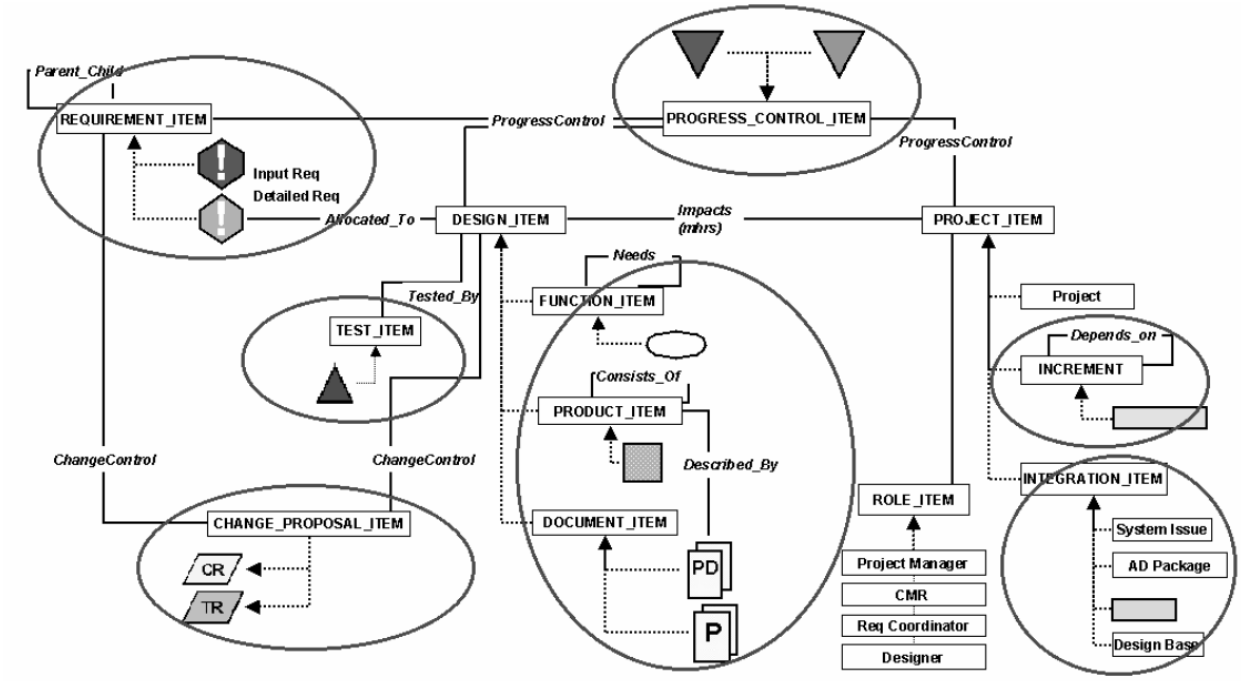

Рис. 3.10. Традиційна архітектура ІС

Існує кілька причин появи такого роду ISA:

• На ринку є ІС для конкретної ситуації координації. Наприклад, RequisitePro від Rational і DOORS від Telelogic є добре відомими продуктами для управління вимогами.

• Не існує концептуальної моделі, яка б взаємопов'язувала різні ситуації координації. Пам’ятайте, що концептуальна модель на рисунку 3.10 виникла завдяки використанню фреймворку в конкретних ситуаціях розробки, і ця модель має довгу еволюційну історію. За відсутності такої стратегії побудови, як у комплексному фреймворці, важко зрозуміти, як визначати значущу модель для всіх ситуацій. Таким чином, без загальної об’єднуючої моделі амбіції зводяться до ізольованих ситуацій координації, де світогляд кожного інструменту визначає порядок денний для взаємодії між інструментами.

• Не завжди є можливість керування всіма елементами координації в ІС. Проблеми впровадження та розгортання корпоративних систем керування даними про продукт (PDM) були успішними. Однак це стосується лише однієї з ситуацій координації на рисунку 3.10, ситуації координації продуктів і документів. Крім того, ця ситуація є досить стабільною, оскільки визначення продуктів і документів є стабільними протягом тривалого періоду. Інші ситуації координації, наприклад, поступовий розвиток, є набагато нестабільнішими.

Деякі проблеми, пов'язані з цим типом ISA:

• Необхідно реалізувати ряд інтерфейсів між окремими ІС, щоб досягти відстежуваності в кількох ситуаціях координації. Обслуговувати такі інтерфейси, як правило, важко і дорого, особливо якщо ІС надходять від різних постачальників.

• Якщо інтерфейси між ІС не реалізовані, інформація повинна передаватись між ІС вручну. Це тягне за собою великий ризик того, що інформація стане суперечливою.

• Кожна ІС має власний інтерфейс користувача. Крім того, кожна ІС має власний особливий «світогляд», тобто властивий набір концепцій,

нотацій і процедур тощо. Таким чином, уся область координації здається суб'єктам фрагментованою та неоднорідною.

• Зміни, які впливають на всі ситуації координації, важко здійснити через притаманні світогляди різнорідних ІС. Усі зміни, що впливають на перший рівень у багатошаровій моделі, є такого роду, наприклад, введення нового загального корпоративного атрибута для продуктів. Це означає, що здатність координаційної області реагувати на загальні зміни буде низькою.

3.4.2. Альтернативна архітектура

У рамках підходу Framework область координації створюється акторами, які використовують моделі та інформаційну систему в Framework як об'єкти під час побудови. Це означає, що всіма пунктами координації, які учасники сприймають як важливі, можна керувати в одній інформаційній системі, в даному випадку eMatrix. Це показано на рисунку 3.11.

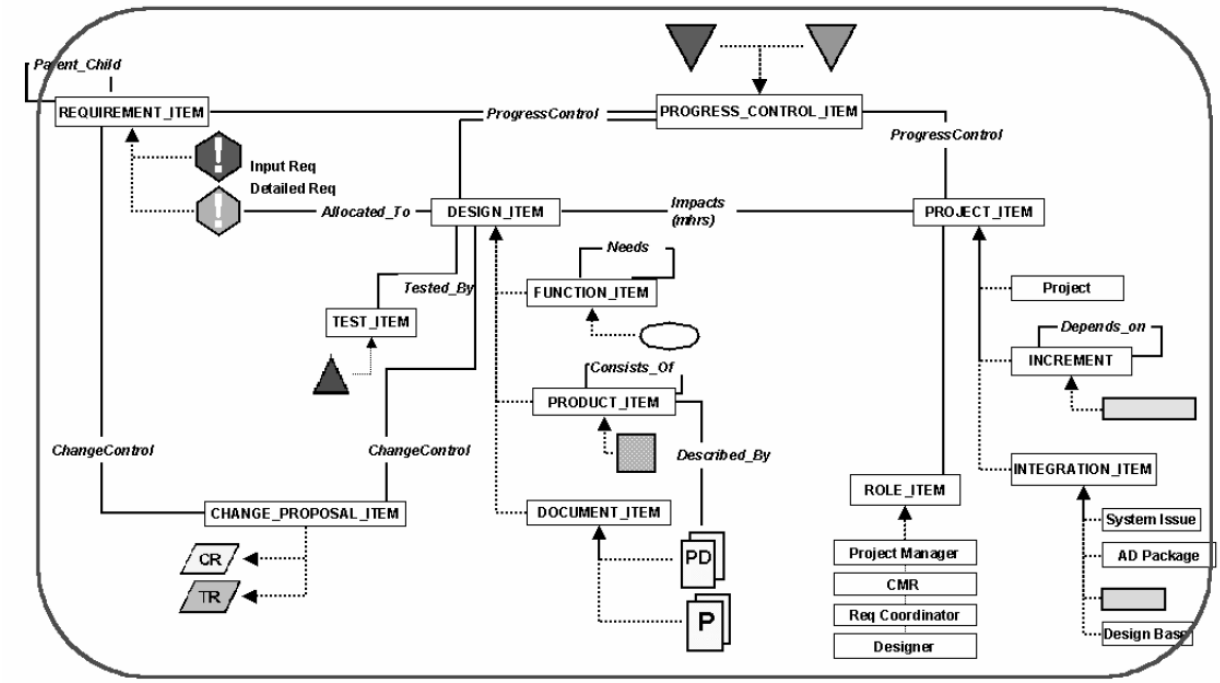

Рис. 3.11. Обсяг додатків eMatrix

Ця ідея стала відомою як концепція «єдиного інструменту управління». Він має певні переваги в порівнянні з традиційною архітектурою:

• Інтерфейси між ІС в області координації не потрібні.

• Декілька ІС можна замінити однією системою.

• Інформація про координацію узгоджується.

• Для сфери координації існує однорідний світогляд.

• Зміни, які є спільними для всіх ситуацій координації, легше зробити.

• Здатність реагувати на нав'язані зміни вище.

Проте концепція «єдиного інструменту управління» є важко реалізованою в повній мірі. Можна виділити дві основні причини цього. Незважаючи на уявну простоту, ця концепція загалом не була зрозуміла поза групою акторів, залучених до побудови різних областей координації. Інша причина полягала в тому, що деякі існуючі ІС набули стану «чорних ящиків» відповідно до теорії акторської мережі. Тобто їх використання вже було інтерналізовано та не відображено в області координації. Прикладом із домену А є використання ІС від Rational для керування запитами на зміни, хоча це з рівним успіхом можна було б зробити в eMatrix.

Концепція «єдиного інструменту управління», звичайно, не виключає використання інших ІС або інструментів, необхідних для розробки складної системи. Наприклад, керування файлами для розробки програмного забезпечення має здійснюватися спеціальним інструментом керування конфігурацією програмного забезпечення (SCM), таким як ClearCase від Rational. Це також означає, що існуватимуть інтерфейси між ІС у Framework та іншими IS. Однак ці інтерфейси не будуть лежати в області координації. Питання про те, чи можна розглядати певний елемент як елемент у координаційній області, чи ні, в кінцевому підсумку залежить від характеру залучених областей діяльності.

3.4.3. Корпоративне підключення

Основним принципом організації стала децентралізація. Компанії отримали велику свободу у прийнятті рішень щодо підтримки процесів та інформаційних систем для розробки продуктів. Коротше кажучи, лише два

механізми були спільними: корпоративний архів застарілих продуктів, де зберігаються всі випущені продукти, і загальний набір правил для того, як керувати продуктами та документами з їх описом .

Таке розташування має багато переваг. Однак основним недоліком є те, що локальні рішення можуть стати несумісними. Особливо це проявляється, коли йдеться про архіви для зберігання проектної документації. Використовується багато локальних архівів, що створює багато проблем, перш за все, коли продукти випускаються в загальний архів продуктів.

Щоб навести порядок в архіві проекту, деякі місцеві компанії почали розробляти власні рішення на основі системи C-PDM. Усі ініціативи, засновані на системі C-PDM, загалом обмежувалися керуванням продуктами та пов'язаними з ними документами з описом (рис. 3.12).

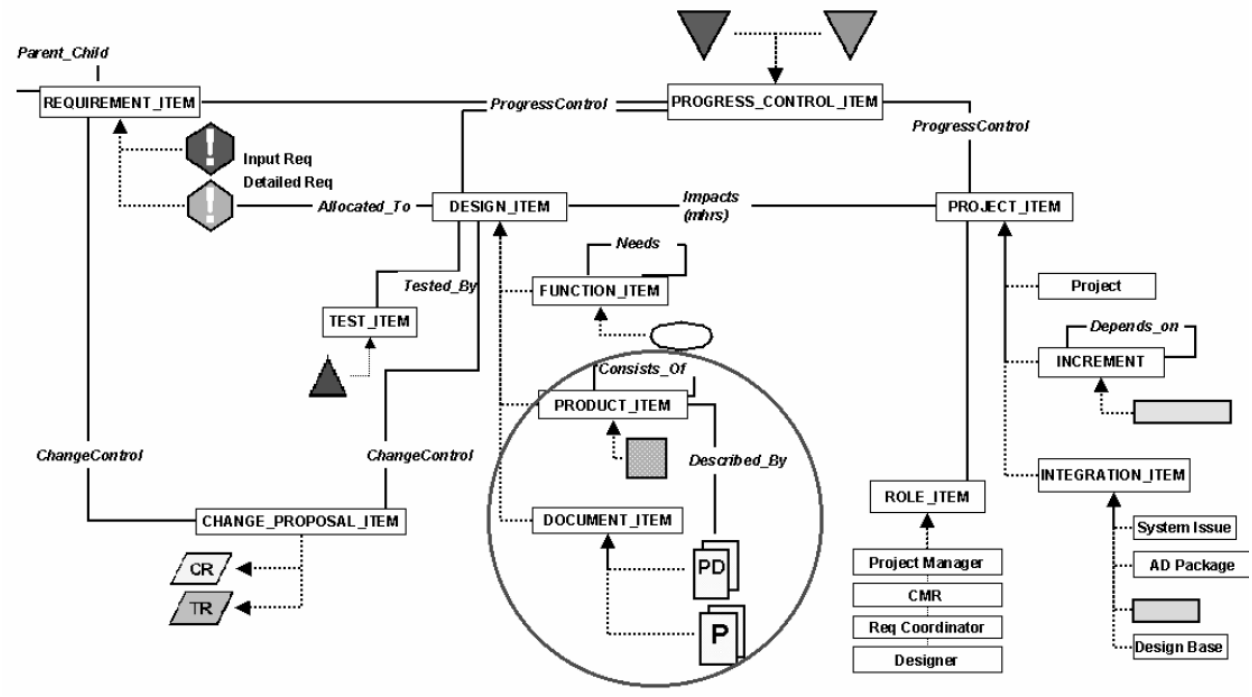

Рис. 3.12. Область застосування системи C-PDM

Оскільки система C-PDM базувалася на комерційній, добре запровадженій системі PDM, можна було розглядати її як заміну застарілої системи, яка створювалася та обслуговувалася власними силами.

Архіви проектів можуть складалися з понад 100 сайтів розробки, і консолідація цих архівів була надзвичайно важливою.

Це означає, що і система C-PDM, і eMatrix виникли з потреби архівації проектної документації, що суперечить більшості застосувань систем PDM. Звичайна сфера застосування систем PDM – це саме та сфера, яку охоплювала застаріла система, тобто управління продуктами та документами та архівування випусків.

Однак основною відмінністю між додатком C-PDM і додатком eMatrix є частини концептуальної моделі, які лежать поза овалом на рисунку 3.12, тобто управління вимогами, поетапне керування розробкою тощо. Це було включено до додатків eMatrix але не в програмах C-PDM. Це означає, що архітектура, заснована на системі C-PDM, насправді відноситься до традиційного типу на рисунку 3.10. Отже, вона матиме всі недоліки цієї архітектури.

Консолідація різних додатків C-PDM була ускладнена тим фактом, що не було спільного значення того, що являє собою продукт. Консолідація локальних додатків C-PDM була зумовлена, що продукти та документи з описом продуктів повинні розглядатися однаково в усій компанії.

Відповідно потрібне спільне стабілізуюче ядро, де визначено обов’язкові елементи для всього підприємства. Однак у той час як додатки eMatrix містять усю концептуальну модель, включаючи основну частину продукту, альтернатива C-PDM включає лише частини продукту та документа. Стверджується, що вирішальною відмінністю між двома альтернативами є простота внесення змін в eMatrix порівняно з системою C-PDM. Це дає змогу побудувати всю область координації відповідно до Стратегії побудови домену в рамках. Такий спосіб роботи неможливий у системі C-PDM.

Висновки до розділу

В даному розділі проведено опис комплексного фреймворку і основних елементів, з яких він складається. Метою застосування фреймворку є побудова домени координації. Також здійснено опис використання Framework у практиці розробки. Вважається, що на проект впливає фреймворк, якщо проект використовує або надає координаційну інформацію через фреймворк, головним чином інформаційну систему eMatrix.

ВИСНОВКИ

В магістерській роботі розглянуто та досліджено моделі, методи та засоби комплексних фреймворків розробки програмних систем та рішень.

Для того, щоб досягти спільного значення щодо домену координації, до розробки фрейворку включено стратегію побудови домену. Ця стратегія складається з безперервної ітерації між фазами роздумів і діями, під час яких моделі та інформаційна система поперемінно відображаються та випробовуються на практиці. У цьому процесі моделі та ІС сприймаються як ознаки, які опосередковують спільне значення між акторами. Крім того, ітерація між розробкою в області координації та використанням в області використання означає, що впровадження в інформаційних систем постійно змінюється. Таким чином, використовується еволюційний підхід до розробки інформаційної системи. Конкретний синтаксис і семантика кожної моделі, а також те, яку конкретну інформаційну систему використовувати, не прописані в рамках розробки комплексного фреймворку. Результатом застосування стратегії побудови домену є поступове створення елементів фрейворку, включаючи спільне значення між учасниками в контексті координації.

Фреймворк, який розглядається як артефакт, можна вважати внеском знань у тому сенсі, що він є результатом діяльності з мотивом забезпечення координації.

Теорію сфери діяльності, викладену в роботі можна розглядати як альтернативну теоретичну відправну точку для розробки інформаційних систем, де розглядаються як соціальні, так і технічні аспекти. Також надається оцінка деяких інших теорій.

Концепцію координації як області діяльності, викладену в магістерській роботі можна розглядати як внесок, що стосується координації дій при розробці комплексного фрейворку.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Allen B R, Boynton A C: Information Architecture: In Search of Efficient Flexibility, MIS Quarterly, December 1991.
2. Axelsson K: Metodisk systemstrukturering - att skapa samstämmighet mellan informationssystemarkitektur och verksamhet, Dissertation No. 1, Department of Computer and Information Science, Linköping University, Linköping (in Swedish). 1998
3. Alvesson M, Sköldberg K : Tolkning och Reflektion - Vetenskapsfilosofi och kvalitativ metod, Studentlitteratur (in Swedish), Lund. 1994.
4. Baskerville R, Wood-Harper T: A Critical Perspective on Action Research as a Method for Information Systems Research, Journal of Information Technology 11,1996, pp. 235-246. 1996.
5. Berger P, Luckmann T : The Social Construction of Reality, Reprint by Penguin Books, London 1991.
6. Braf E: Organisationers kunskapsverksamheter - en kritisk studie av “knowledge management”, (in Swedish), Licentiate thesis no 37, Department of Computer and Information Science, Linköping University, Linköping, Sweden 2000.
7. Broadbent M, Weill P: Management by Maxim: How Business and IT Managers Can Create IT Infrastructures, Sloan Management Review, Spring. 1997.
8. Chalmers A F : What is this thing called science?, University of Queensland Press, St Lucia, Queensland 1976.
9. Checkland P : Systems Thinking, Systems Practice, Wiley, Chichester 1981.
10. Checkland P: From Framework through Experience to Learning: the essential nature of Action Research, in Information Systems Research, Contemporary Approaches and Emergent Traditions, Nissen H E & Hirschheim R (Editors), 1991.

11. Crnkovic I, Asklund U, Persson Dahlqvist A : Implementing and Integrating Product Data Management and Software Configuration Management, Artech House, London, planned in June 2003.

12. Dahlbom B, Mathiassen L: Computers in context : the philosophy and practice of systems design, Cambridge, Blackwell, Mass,1993.

13. Davenport T, Eccles R, Prusak L : Information Politics, Sloan Management Reviev, fall 1992.

14. Durkheim É : Selected writings, transl., and with an introd. by Anthony Giddens, Cambridge University Press, Cambridge 1972.

15. Earl M J : An Organizational Approach to IS Strategy Making, in Earl M J (Ed, 1996), Information Management - The Organizational Dimension, Oxford University Press, 1996.

16. Ehn P: Work-Oriented Design of ComputerArtefacts. Arbetslivscentrum, Stockholm, Sweden, 1988.

17. Engwall M : Jakten på det effektiva projektet, Stockholm: Nerenius & Santerus (in Swedish), 1995.

18. Eppinger S D, Whitney D E, Smith R P, Gebala D A : A Model-Based Method for Organizing Tasks in Product Development, Research in Engineering Design, 1994.

19. Engels F (): On the part played by labour in the transition from ape to man, Dialectics of Nature, Foreign Languages Publishing House, Moscow, 1954. Eriksson M, Lilliesköld J, Jonsson N, Novosel D (2002): How to Manage Complex, Multinational R&D Projects Successfully , Engineering Management Journal, Vol.14, No.2 June 2002.

20. Erixon G: Modular Function Deployment - A Method for Product Modularisation, Doctoral Thesis, Royal Institute of Technology, Stockholm, 1998.

21. Erixon G, Erlandsson A, Yxcüll A-v, Östgren B : Modularisera produkten, in Swedish, Industrilitteratur 1994.

22. Engeström Y: Activity theory and individual and social transformation, in Engeström Y, Miettinen R, Punamäki RL (eds. 1999)

Perspectives on Activity Theory, Cambridge University Press, Cambridge UK., 1999.

23. Freire P: Pedagogy of the Oppressed, Penguin Books, London, 1996.

24. Freyd J: Shareability: the social psychology of epistemology, Cognitive Sciences, 7, 1983.

25. Gandhi M, Robertson E L: A Specification-based Data Model, Indiana University Computer Science Department Technical Report TR344, Indiana University, Indiana. Available at Oct 2002.

26. Gandhi M, Robertson E L : SDBM as a Model for Codesign Data, in Rozenblit J and Buchenrieder K (eds.) Computer Aided Software/Hardware Engineering, IEEE Press, Piscataway, 1995.

27. Gedenryd H: How designers work, Dissertation 75, Lund: Lund University Cognitive Studies,1998.

28. Gilb T: Principles of Software Engineering Management, Addison-Wesley Longman, 1989.

29. Giddens A: The Constitution of Society, Polity Press, Cambridge, 1984.

30. Golden-Biddle K, Locke K: Appealing Work: an Investigation of how Ethnographic Texts Convince, Organization Science, Vol. 4, No. 4. November 1993.

31. Goldkuhl G: Kunskapande, Institutionen för Datavetenskap, Linköpings Universitet (in Swedish), Linköping, 1998.

32. Goldkuhl G, Röstlinger A: Praktikbegreppet. En praktikgenerisk modell som grund för teoriutveckling och verksamhetsutveckling, CMTO, Linköpings universitet (in Swedish), 1998.

33. Goldkuhl G: Anchoring scientific abstractions – ontological and linguistic determination following socio-instrumental pragmatism, in Proceedings of European Conference on Research Methods in Business, Reading 2002.

34. Grinter R E: Systems Architecture: Product Designing and Social Engineering. In Georgakopulos, D, Prinz, W and Wolf, A L eds. Proceedings of the Work Activities Coordination and Collaboration (WACC '99), 1999.

35. Gärdenfors P:Blotta tanken, (in Swedish), Nya Doxa, Nora, Sweden 1992.

36. Gärdenfors P: Conceptual Spaces: The geometry of thought, MIT Press,Cambridge, Massachusetts, 2000.

37. Gärdenfors P: Hur Homo blev sapiens. Om tänkandets evolution, (in Swedish), Nya Doxa, Nora, Sweden, 2000.

38. Habermas: The Theory of Communicative Action, Volume One, Reason and the Rationalization of Society, Beacon Press, Boston, 1984.

39. Henderson A: A Conversation with Austin Henderson, ACM Interactions, Nov./ Dec., 1998.

40. Hoopes D, Postrell S: Shared Knowledge, "Glitches" and Product Development Performance, Strategic Management Journal, 20, 1998.

41. Huber G: Organizational learning: the contributing processes and the literatures. Organization Science, vol 2 (1), 1991.

42. Hugoson M-Å : Verksamhetsbaserad systemstrukturering, NordDATA90, Göteborg, 1990.

43. von Humboldt W: On language : on the diversity of human language construction and its influence on the mental development of the human species, edited by Michael Losonsky, translated by Peter Heath, Cambridge University Press, Cambridge, 1999.

44. Humphrey W, Kellner M: Software Process Modelling: Principles of Entity Process Models, CMU/SEI-89-TR-2, Software Engineering Institute, Carnegie Mellon University, 1989.

45. Iden J: Six Essays on Business Process Reengineering, dissertation, Department of Information Science, University of Bergen, J994.

46. Iivari J, Lyytinen, K: Research on Information Systems Development in Scandinavia-Unity in Plurality, Scandinavian Journal of Information Systems, 10 (1&2), 1998.

47. Innis R: Semiotics. An Introductionary Anthology (edited by Innis R), Indiana University Press, Bloomington, USA, 1985.

48. Jones M: Structuration Theory, in Rethinking management information systems : an interdisciplinary perspective, Currie, Wendy Galliers, Robert (eds), Oxford Univ. Press, Oxford, 1999.