Уніфікована мова моделювання *UML*. Діаграми прецедентів

(Курс “Інформаційні технології”)

2004

Уніфікована мова моделювання *UML*

UML (***Unified Modeling Language*** – Уніфікована Мова Моделювання) – це ***стандартна нотація візуального моделювання*** програмних систем (але не тільки програмних систем!).

Прийнята консорціумом *Object Managing Group* (*OMG*) восени 1997р.

На сьогодні вона підтримується багатьма об'єктно-орієнтованими *CASE* системами, включаючи *Rational Rose*.

UML – дійсно уніфікована мова, вона:

- не залежить від методології, що використовується при розробці проекту;
- може підтримувати будь-яку об'єктно-орієнтовану мову програмування.

На *UML* можна змістовно описувати класи, об'єкти й компоненти в різних предметних областях, що можуть суттєво відрізнятись одна від одної.

Важлива характеристика *UML* – її відкритість, *UML* має засоби розширення свого базового ядра.

Передумови виникнення *UML*

У середині 90-х існувало більше ***50*** різних ***об'єктно-орієнтованих методів чи мов моделювання***.

У цей же період часу оновлюються версії таких досить розповсюджених методів як: Booch'93, OMT-2 (Object Modelling Technique), Fusion, OOSE (Object-Oriented Software Engineering).

І розроблювачів ПС, і замовників охоплювало занепокоєння при виборі метода проектування ПС, кожен із яких до того ж, як правило, спирався на власну нотацію.

Отже, на часі визріла проблема в ***стандартизації й уніфікації*** підходів до моделювання.

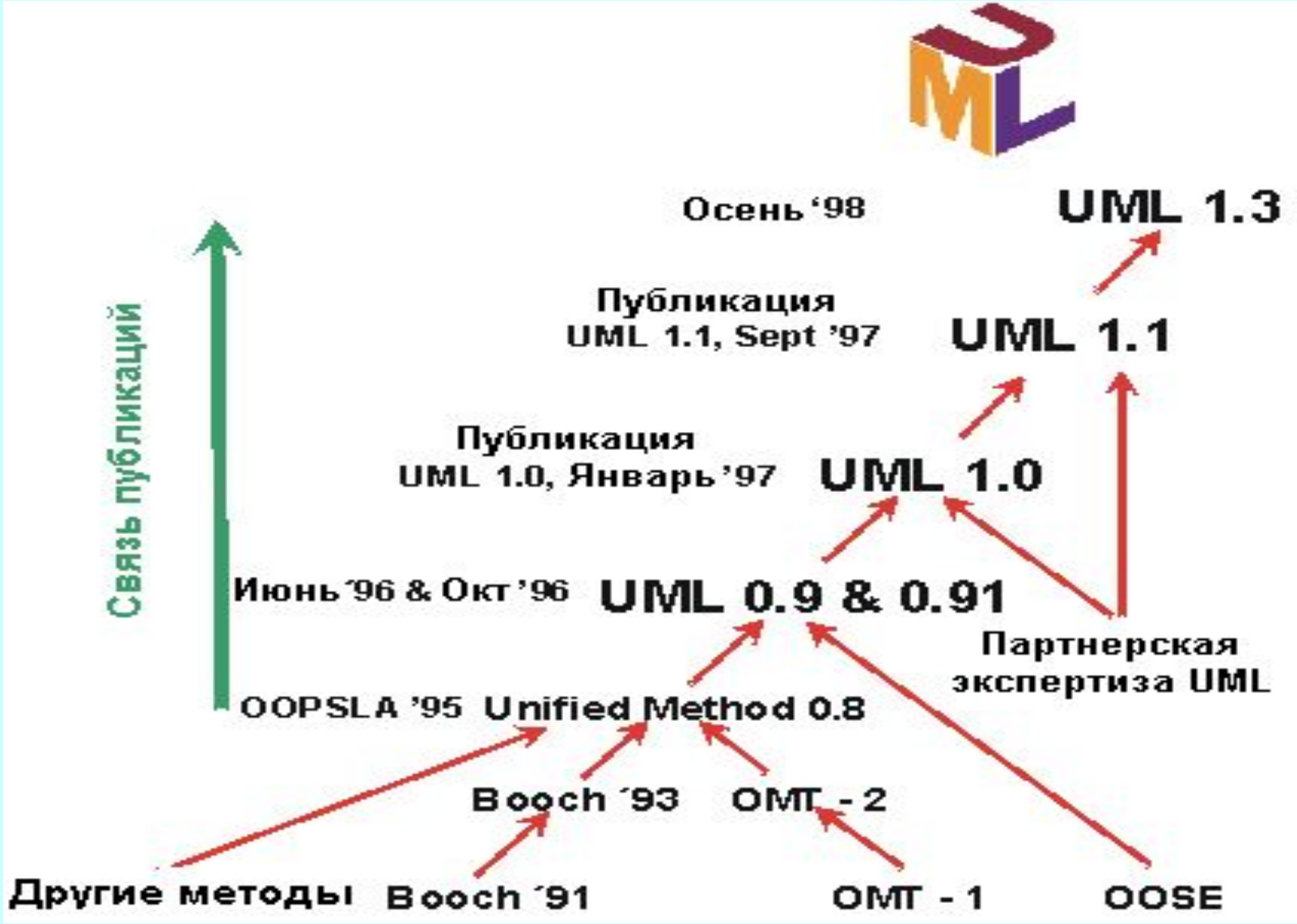
Виникнення *UML*

Початком розробки *UML* вважається жовтень 1994 року, коли у *Rational Software Corporation* силами ***Греді Буча*** (***Grady Booch***) і ***Джима Рамбо*** (***Jim Rumbaugh***) була започаткована робота з уніфікації їх власних методів Booch'93 та OMT. Перша версія Уніфікованого Метода (Unified Method 0.8) була опублікована в жовтні 1995.

Трохи згодом, у тому ж 1995 році, до роботи приєднався ***Айвер Якобсон*** (***Ivar Jacobson***), залучаючи до процесу інтеграції й уніфікації ще один метод – власний метод *OOSE*.

Таким чином, на першому концептуальному етапі *UML* отримав трьох авторів: Буча, Рамбо і Якобсона, кожен із яких був ідеологом свого власного об'єктно-орієнтованого метода візуального моделювання.

Виникнення *UML*



Статичні та динамічні моделі

В *UML* ***інтегровані*** різноманітні відомі засоби візуального моделювання, які добре зарекомендували себе на практиці, зокрема, забезпечується можливість опису двох визначальних видів об'єктних моделей:

- ***структурних*** (або ***статичних***) ***моделей*** – описується структура сутностей системи, включаючи класи, інтерфейси, відношення, атрибути;
- моделей ***поведінки*** (або ***динамічних моделей***) – описується поведінка (функціонування) об'єктів системи, включаючи методи, взаємодію, процес зміни станів окремих компонент чи всієї системи.

Призначення *UML*

- Надати користувачу ***засоби візуального моделювання*** систем різного призначення з акцентацією на можливості їх розробки та отримання документації. (*UML* містить як абстрактні конструкції для представлення моделей, так і цілком конкретні, які дозволяють описувати деталі реалізації програмних систем).
- Забезпечити користувачів ***засобами розширення та специфікації*** з метою більш точного опису конкретних предметних областей. (Хоча у більшості випадків для побудови моделей цілком достатньо базових конструкцій *UML,* все ж в *UML* уведено механізм ***розширення*** базових понять. Крім того, можлива ***спеціалізація*** базових понять, шляхом доповнення останніх новими атрибутами чи властивостями).
- Підтримувати таку специфікацію моделей, яка, з одного боку, була б ***незалежною*** від конкретних мов програмування і, з іншого боку, забезпечувала б потенційні ***можливості реалізації*** у таких мовах.

Діаграми *UML*

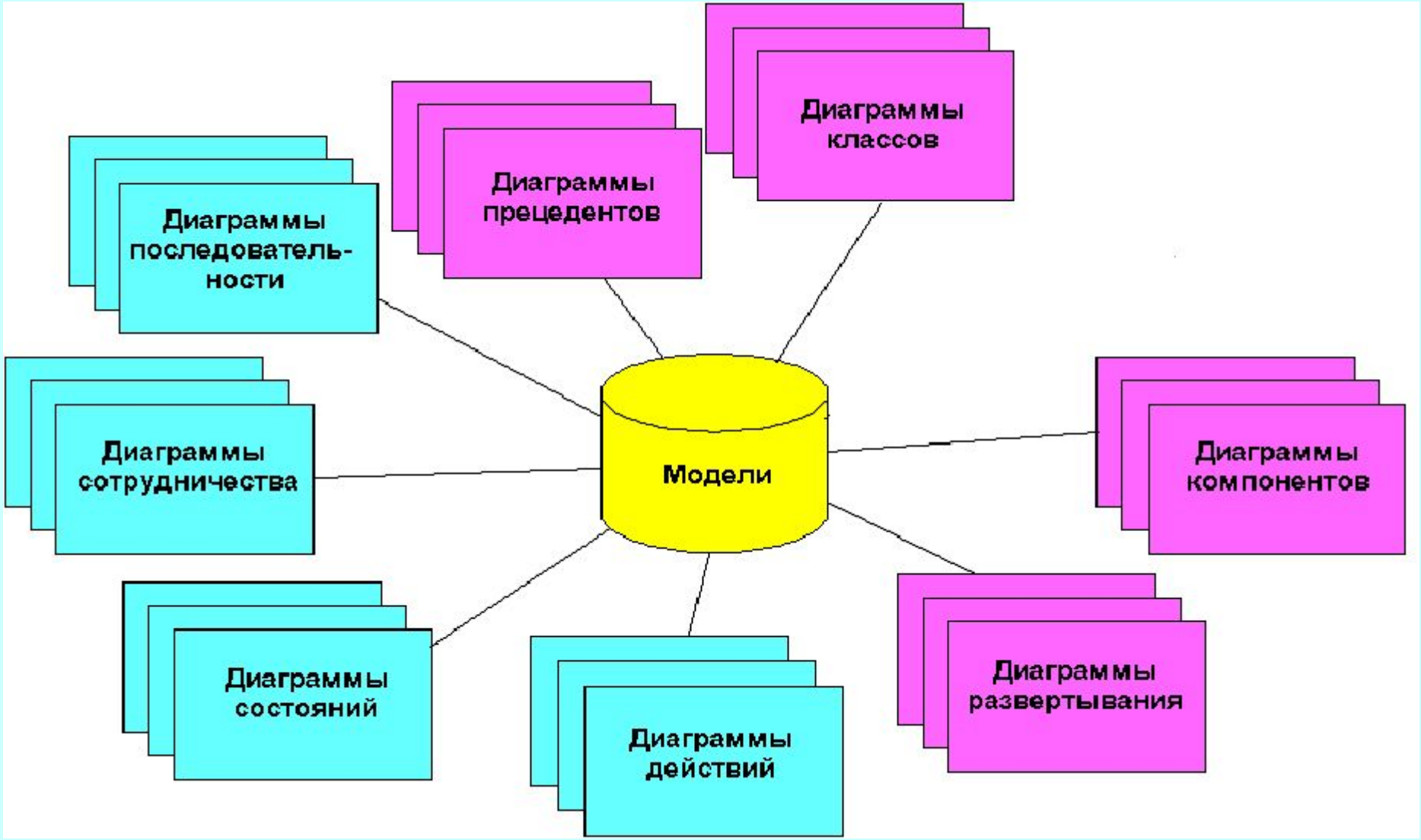
Діаграма *UML* – це зображення у вигляді ***графа з вершинами (сутностями) і ребрами (відношеннями)***.

Основна ***мета*** діаграм – ***візуалізація архітектури*** розроблюваної системи з різних точок зору.

Діаграма – деякий зріз системи. Звичайно діаграми дають згорнуте представлення елементів, із яких складається розроблювана система. При цьому один і той самий елемент може бути присутнім у декількох (а іноді й в усіх) діаграмах.

При візуальному моделюванні з *UML* використовуються ***вісім видів діаграм***, кожна з яких може містити елементи певного типу. Типи можливих елементів і відношень між ними залежать від виду діаграми.

Діаграми *UML*: діаграми структури та діаграми поведінки



Види діаграм *UML*. Коротка характеристика

Діаграми прецедентів ***або діаграми використання (use case diagrams).*** Такі діаграми описують ***функціональність***, яка буде надаватись користувачам системи, котра проектується. Представляються шляхом використання ***прецедентів*** та ***акторів***, а також ***відношень між ними***. Набір усіх прецедентів діаграми фактично визначає ***функціональні вимоги***, за допомогою яких може бути сформульоване ***технічне завдання***.

Діаграми класів ***(class diagrams)*** описують статичну структуру класів. Дозволяють (на концептуальному рівні) формувати "словник предметної області" та (на рівні специфікацій і рівні реалізацій) визначати структуру класів у програмній реалізації системи. Можуть використовуватись для генерації каркасного програмного коду (в реальній мові програмування).

Види діаграм *UML*. Коротка характеристика

Для опису динаміки використовуються ***діаграми поведінки (behavior diagrams)***, що підрозділяються на

- ***діаграми станів (statechart diagrams);***
- ***діаграми діяльності (активності) (activity diagrams);***
- ***діаграми взаємодії (interaction diagrams),*** що у свою чергу підрозділяються на
  - ***діаграм послідовності (sequence diagrams);***
  - ***діаграм кооперації (співробітництва) (collaboration diagrams).***

І, нарешті, ***діаграми реалізації (implementation diagrams)*** поділяються на

- ***компонентні діаграми· (діаграми компонентів) (component diagrams);***
- ***діаграми розгортання· (deployment diagrams).***

Спрощена стратегія використання *UML*-діаграм при моделюванні програмних систем

Спочатку для програмної системи серією ***діаграм прецедентів*** визначається її ***зовнішня функціональність*** (виділяються всі прецеденти та актори, а також відношення між ними).

Уся подальша робота над проектом має здійснюватись на основі прецедентів: для кожного прецеденту формується опис його динаміки у вигляді серії ***діаграм взаємодії*** та ***діаграм діяльності***.

З отриманих описів шляхом виявлення об'єктів, що задіяні в реалізації прецедентів, будуються ***діаграми класів***.

Для визначення поведінки класів із складною динамікою реагування на події можуть формуватись ***діаграми станів***.

Розміщення об'єктів по програмних модулях описується в ***компонентних діаграмах***, а розміщення програмних модулів по вузлах комп'ютерам мережі – у ***діаграмах розгортання***.

Деякі рекомендації при побудові діаграм

Кожна діаграма має бути ***завершеним*** представленням фрагмента предметної області, який моделюється, тобто має враховувати всі істотні сутності обраного рівня абстрагування (***повнота моделі***).

Усі сутності діаграми мають бути ***одного концептуального рівня*** (відповідно до обраного рівня абстрагування).

Діаграми не повинні містити ***суперечливу інформацію*** (наприклад, не повинно бути елементів з однаковими іменами, але з різними атрибутами – омонімія, так само не повинно бути циклів у графах, що представляють відношення узагальнення чи агрегування).

Діаграми, в сукупності, мають бути ***самодостатніми***, тобто не повинні вимагати додаткового текстового пояснення стосовно семантики чи інтерпретації окремих елементів діаграм. (Важлива роль при цьому покладається на побудову ***ієрархії моделей***).

Не обов'язково у проектуванні використовувати ***всі види*** діаграм (наприклад, для локальних програмних систем не доцільно розробляти діаграму розгортання).

Діаграми прецедентів або діаграми використання (*use case diagrams*)

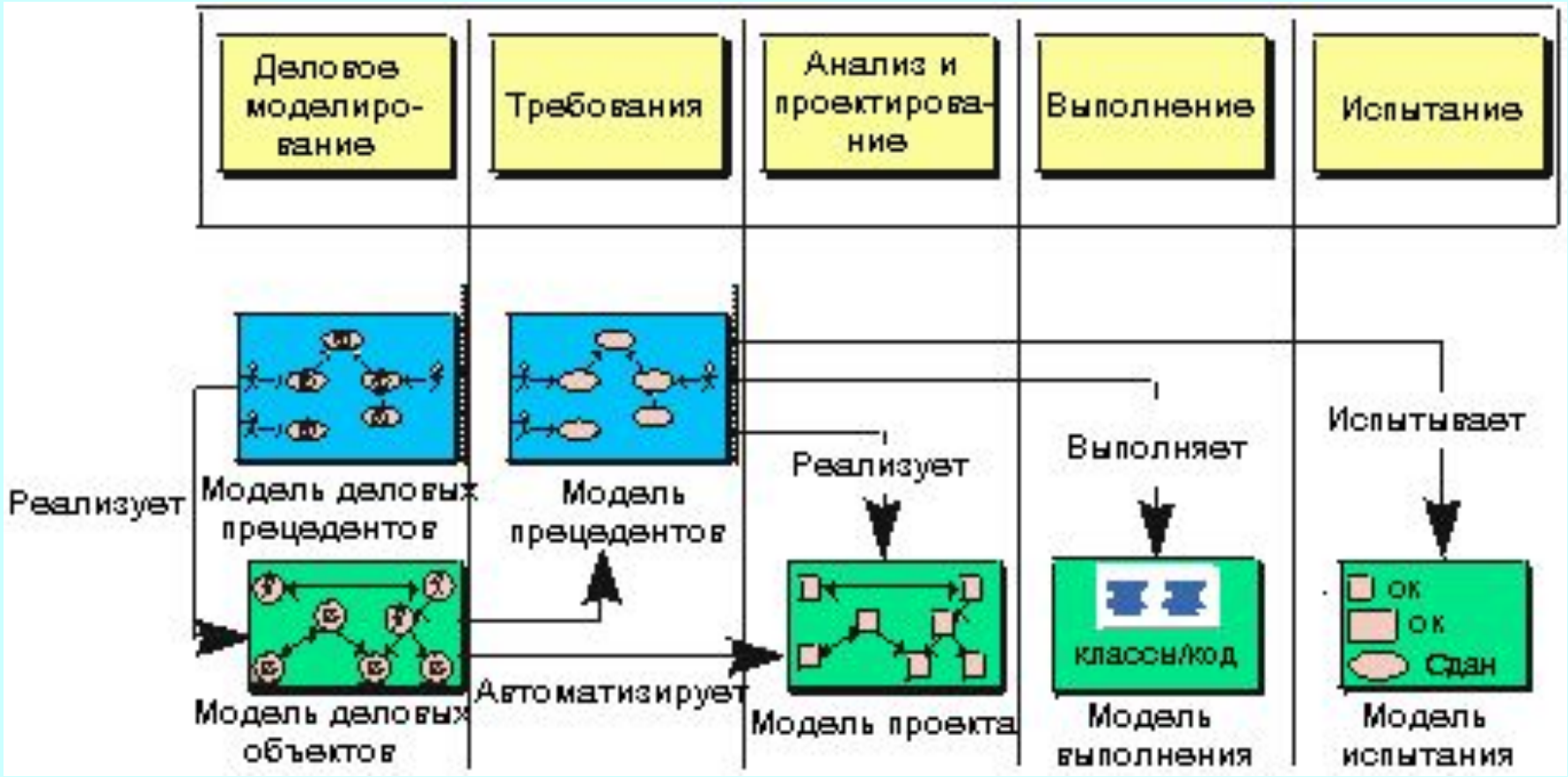
Значення. Діаграми варіантів використання є ***первісним, концептуальним представленням*** (***концептуальною моделлю***) ПС в процесі її проектування й розробки. Вони виступають основою ***подальшої деталізації*** системи у формі логічних і фізичних моделей.

Розробка діаграм варіантів використання переслідує такі конкретні цілі:

- визначити загальні ***кордони*** та ***контекст*** системи;
- сформулювати загальні ***вимоги*** до ***функціональної поведінки*** системи;
- отримати ***первісну*** документацію для предметної взаємодії розроблювачів системи з її замовниками й користувачами.

Управління прецедентами є одним з ***наріжних каменів*** *RUP*.

Управління прецедентами в *RUP*



Діячі *(actors)*. Моделювання контексту системи

Діаграми варіантів використання створюються на ***етапі аналізу вимог*** до системи.

Аналіз починається з ідентифікації ***діячів (діючих осіб, акторів, актантів) (actors),*** як потенційних користувачів системи. Діячі позначаються на діаграмі стилізованими людськими фігурками.

Для визначення діячів, необхідно розглянути ситуації, типові для використання системи. Користувачами системи не обов'язково мають бути ***люди***; це можуть бути інші (зовнішні) ***системи***, що звертаються до даної системи чи до яких звертається дана система.

Варто зауважити, що між діячами й користувачами системи є важлива відмінність: ***діячі***, по суті, ***– це типи***, тоді як користувачів слід розглядати як конкретних ***екземплярів*** таких ***типів***.

Сутності, що знаходяться поза системою і взаємодіють із нею, складають її ***контекст***. Таким чином, визначення ***діячів*** діаграм прецедентів дозволяє ***моделювати контекст*** системи.

Варіанти використання, прецеденти (*use case*). Моделювання вимог до системи

Прецеденти є засобом специфікації системи с точки зору отримання ***діячами*** деяких істотних для них ***результатів***. Тобто, прецедентами задаються ***сервіси*** (***функціональні можливості***) системи, якими може скористатись діяч.

Як саме визначати ***прецеденти?*** Відповідь може бути такою. Стосовно кожного діяча слід розглянути ***типові*** (***узагальнюючі***) варіанти сервісів (функцій), які мають підтримуватись системою і якими може скористатись діяч. (Можливо, для цього доведеться спочатку провести ділове моделювання.) Виділеним сервісам і мають співставлятись ***прецеденти.***

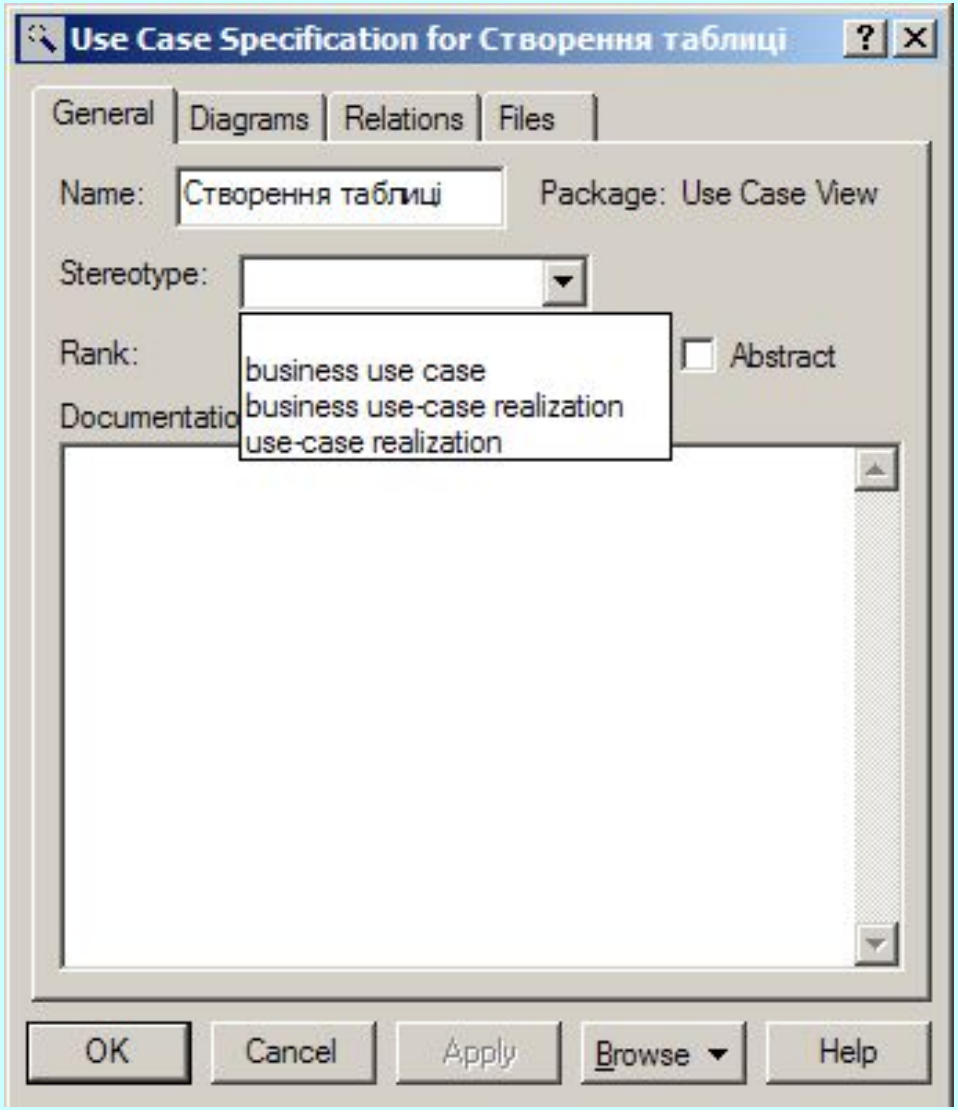
Фактично визначенням прецедентів забезпечується ***моделювання вимог*** до системи.

Прецеденти специфікують ***поведінку*** розроблюваної системи (***що має робити система***), ***не визначаючи реалізацію*** цієї системи (***яким чином вона це має робити***). Вони дозволяють досягти взаєморозуміння між розроблювачами, замовниками, експертами і кінцевими користувачами. Крім того, прецеденти допомагають перевіряти й контролювати архітектуру системи у процесі її розробки.

Прецеденти. Специфікація прецедентів у *Rational Rose*

У діаграмах прецедент зображується ***еліпсом***.

Як правило, для іменування прецедентів використовуються дієслова чи короткі дієслівні фрази (“Створення таблиці”, “Створити таблицю”).



Прецеденти. Потоки подій

Прецедентам рекомендується співставляти ***потоки подій***, які описують поведінку системи в процесі отримання необхідного результату.

Потоки подій описуються мовою предметної області, а не в термінах реалізації.

Найчастіше для опису потоку подій пропонується наступна структура:

- заголовок (наприклад, “Потік подій для прецеденту <Зняти гроші>”);
- короткий опис потоку (наприклад, “Дозволяє клієнту зняти гроші з його рахунку”);
- передумови (*pre-conditions*) (діаграми прецедентів не дозволяють відображати послідовний характер використання прецедентів!);
- головний потік подій та, можливо, його підпотоки;
- альтернативні потоки подій;
- постумови (*post-conditions*).

Потоки подій. Ще одна структура опису

X. Потік подій для прецеденту <Зняти гроші>.

// X - номер потоку (прецеденту)

X.1. Передумови (*pre-conditions*).

X.2. Головний потік.

X.3. Підпотоки (S-1, S-2, …).

X.4. Альтернативні потоки (E-1, E-2,).

X.5. Постумови (*post-conditions*).

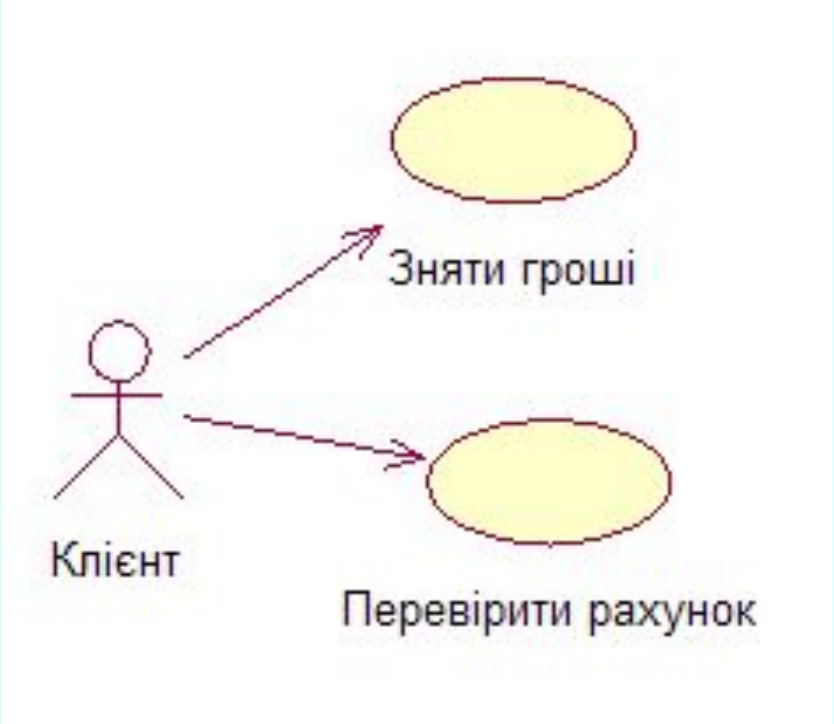
Приклад головного потоку подій для прецеденту “Зняти гроші” (система “Банкомат”)

1. Прецедент починає виконуватись, коли клієнт вставляє картку.
2. Банкомат пропонує обрати мову спілкування, а за цим - увести код.
3. Клієнт уводить код.
4. Система перевіряє код (E-1);

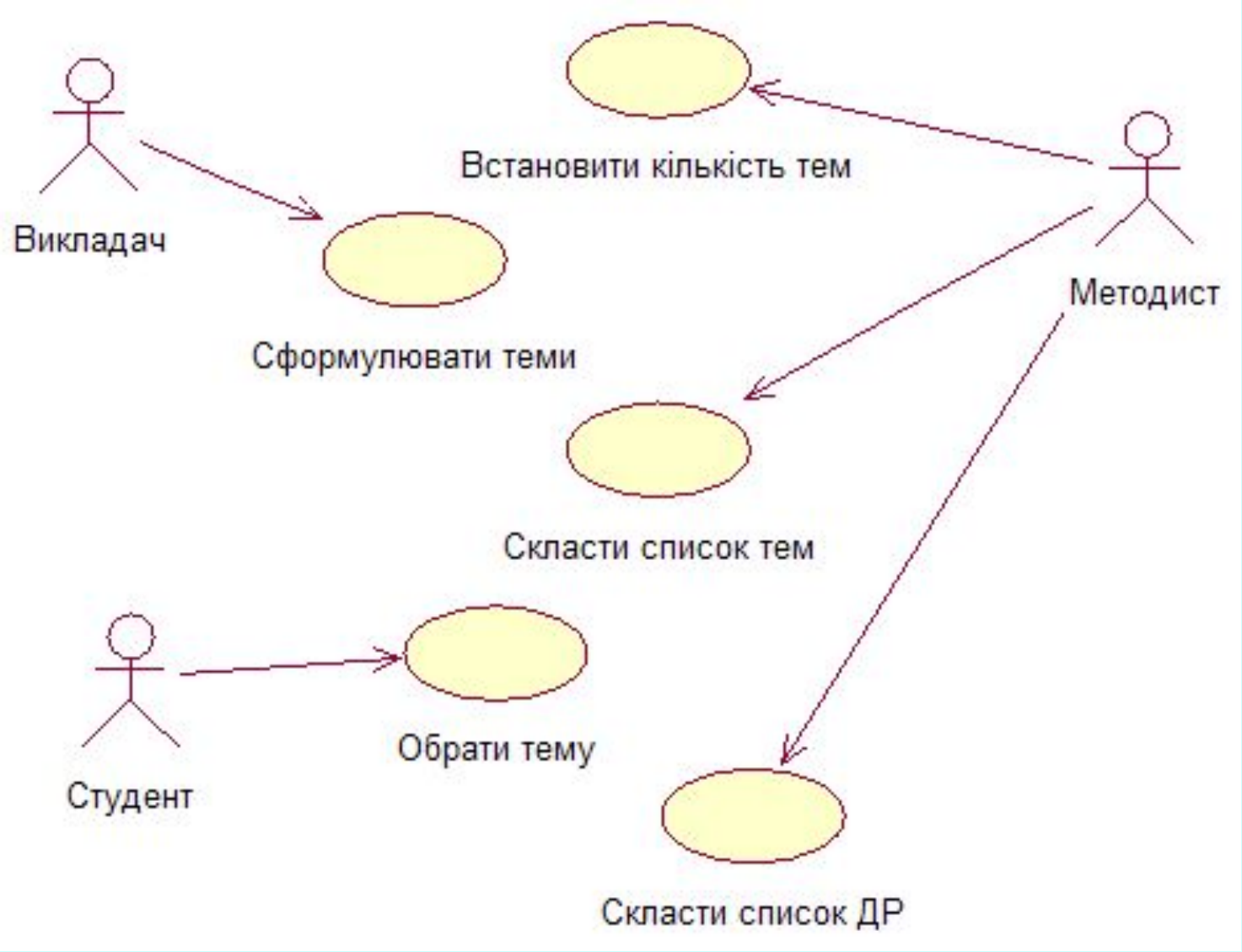
   // E-1 - альтернативний потік “Невірний код”.
5. Банкомат пропонує обрати одну з двох транзакцій:
   - зняти гроші;
   - залишок на рахунку;
6. Клієнт обирає “Зняти гроші”
7. Банкомат пропонує кілька конкретних варіантів значень (5, 20, 50, 100, …) та варіант “Інша сума”.
8. . . .

Діаграма прецедентів. (Банкомат)

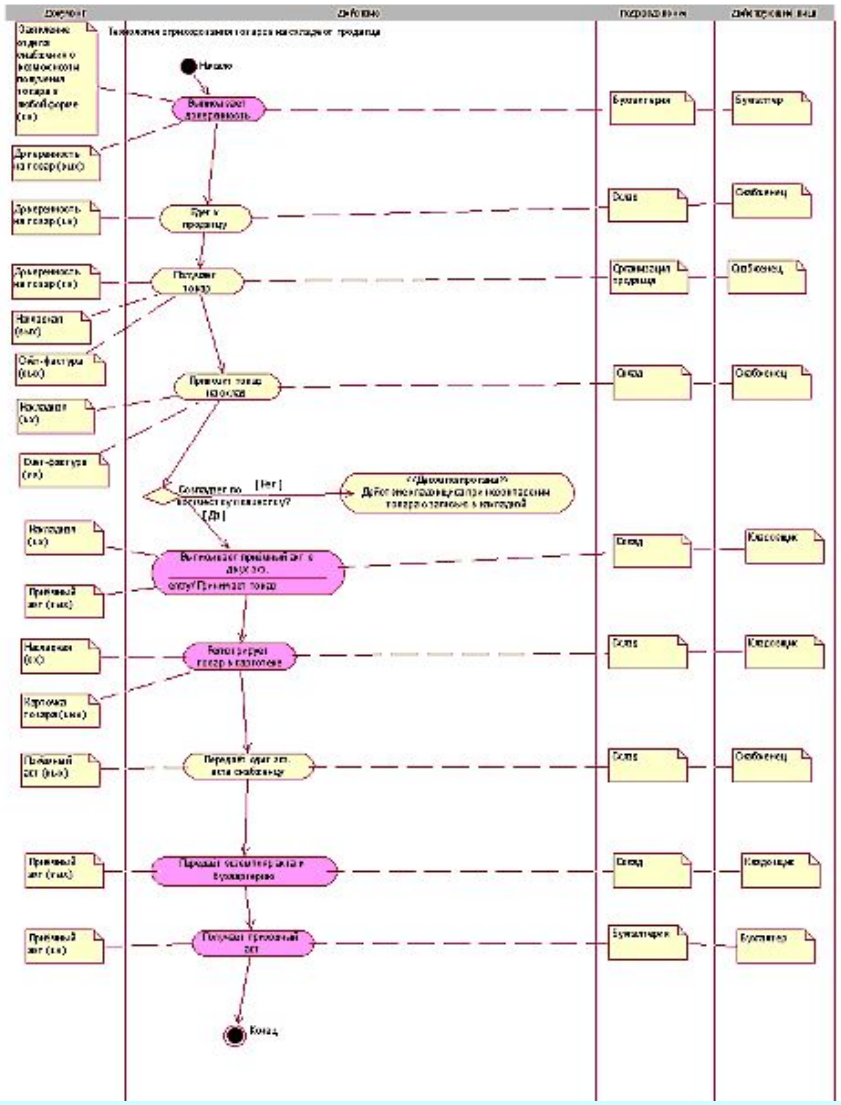
Зв'язки між акторами та прецедентами (***комунікації***) на діаграмах прецедентів представляються ***однонаправленими асоціаціями*** (відповідний стереотип - ***<<communication>>***) і зображаються суцільною стрілкою у напрямку від “ініціатора зв'язку”. Це єдиний тип діаграмних відношень, які можливі між акторами та прецедентами, тому можна цей стереотип і не задавати.



Діаграма прецедентів. (Система “Дипломні роботи”)



Використання ділових моделей



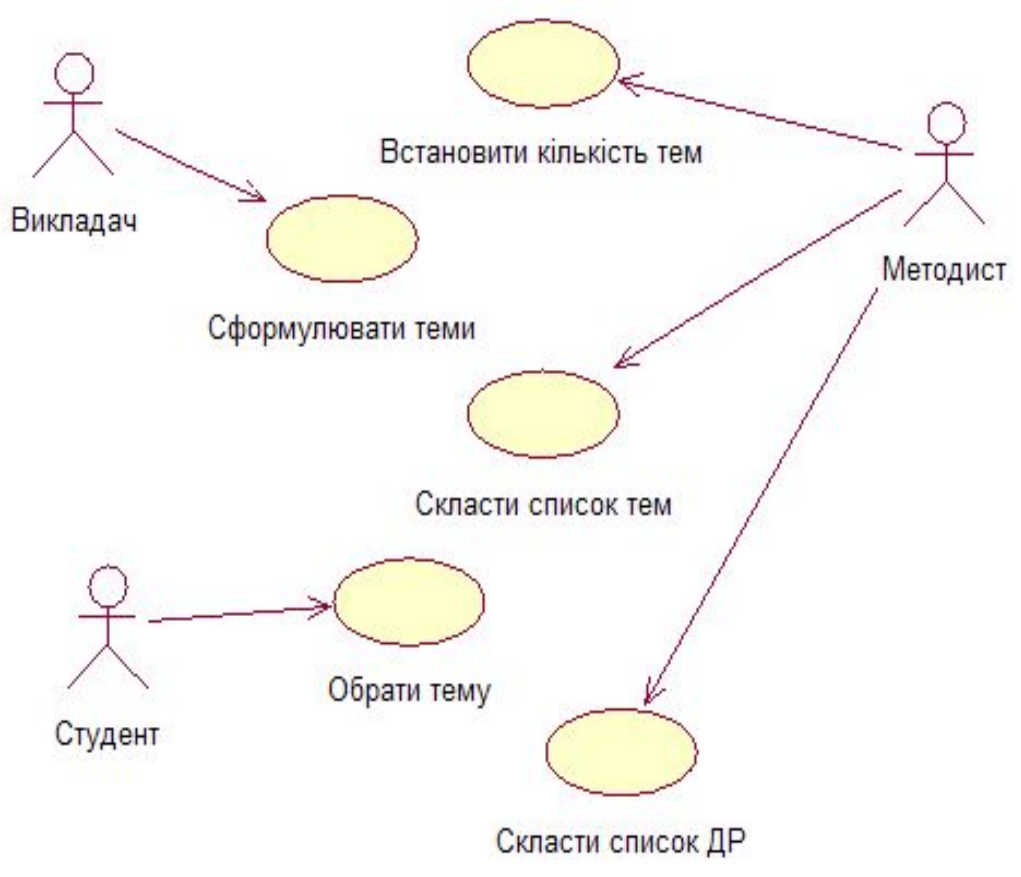
Постановка задачі на автоматизацію: “Оприбуткування товару на складі підприємства від продавця”.

На основі опису бізнесів-процесів з використанням діаграми діяльності (***activity diagram***) визначаються (і позначаються ***кольором***) діяльності, які варто автоматизувати:

1. Виписує доручення;
2. Виписує прийомний акт у двох екземплярах;
3. Реєструє товар у картотеці;
4. Передає екземпляр акта в бухгалтерію;
5. Одержує прибутковий акт.

Е.Б. Золотухина, Р.В. Алфимов **Пример описания предметной области с использованием Unified Modeling Language (UML) при разработке программных систем,** *Interface Ltd., 2001*

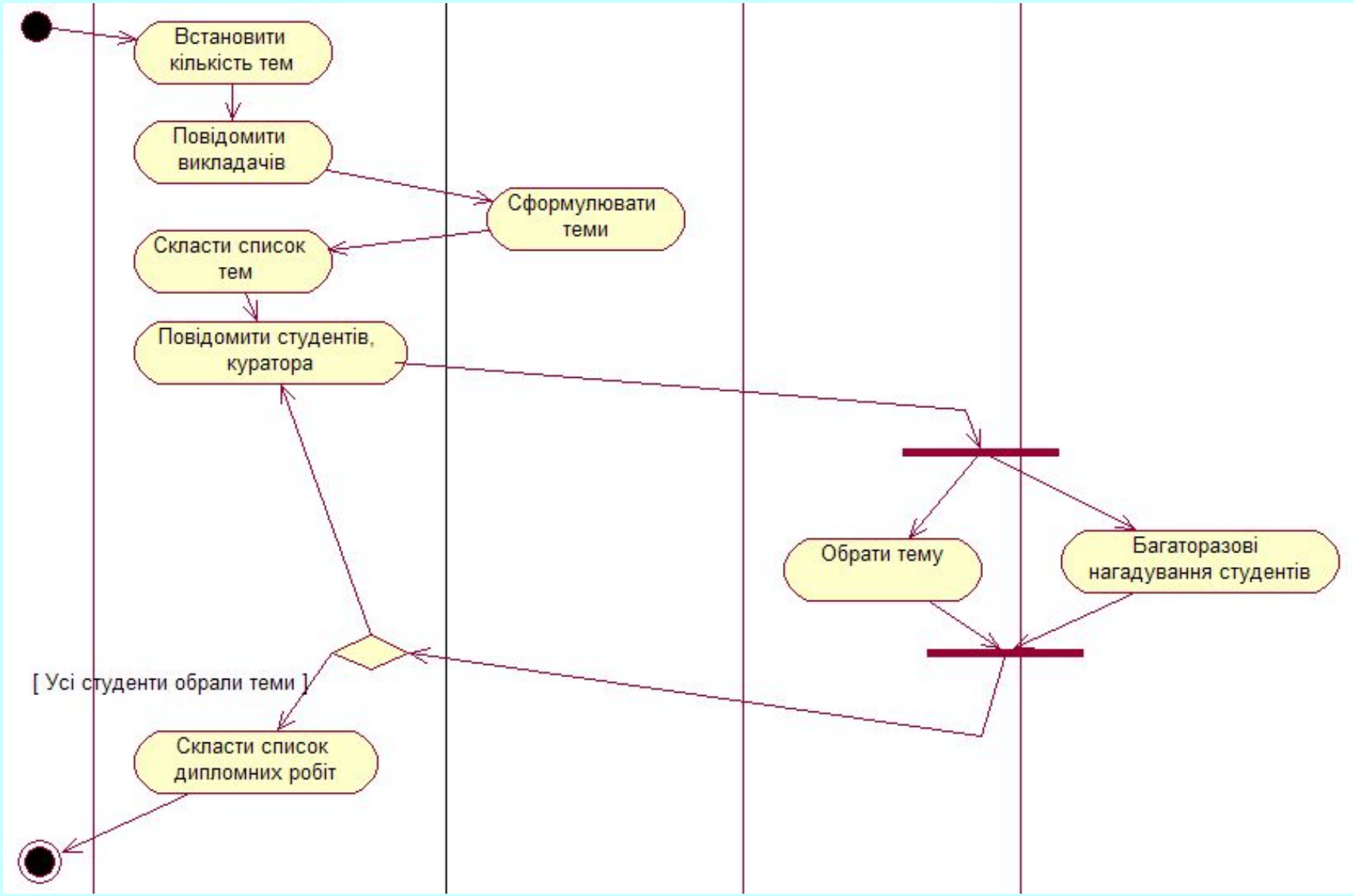
Система “Дипломні роботи”



Можливі додаткові прецеденти “Відправити електронні листи викладачам”, “Відправити електронні листи студентам”.

Мабуть, не підлягає автоматизації ***діловий*** прецедент “Багаторазові нагадування куратором студентів про необхідність обрати теми дипломних робіт”.

Ділове проектування (система “Дипломні роботи”)



Потоки подій та сценарії як типи й екземпляри

Стосовно прецедентів так само, як і у випадку діячів, залучаються поняття на зразок “типи - екземпляри”.

Коли користувач (як екземпляр діяча) взаємодіє із системою, виконується (спрацьовує) деякий ***сценарій (scenario)*** – послідовність деяких дій відповідного прецеденту.

Зрозуміло, що одному прецеденту можуть відповідати одразу кілька різних сценаріїв, і, якщо на потік подій подивитись з “позиції типу”, то його “екземплярами” виступатимуть ***сценарії***.

Іноді відношення “типи - екземпляри” застосовується також до характеристики пар ***прецеденти - сценарії***.

Використання сценаріїв

Архітектура ПС розробляється при розгляді “архітектурно істотних” сценаріїв (для підмножини прецедентів, що представляють найбільш перспективну поведінку, яку повинна підтримати система).

Гарним індикатором стабільності архітектури може бути досвід роботи з архітектурою: якщо зміни в архітектурі малі і залишаються малими, коли вводяться ***нові сценарії***, є всі підстави думати, що архітектура стабільна.

Та найголовніше - саме до сценаріїв застосовуються подальші кроки на шляху моделювання, використовуючи діаграми взаємодії (у першу чергу - ***діаграми послідовності***), а також при потребі діаграми діяльності та діаграми станів.

Використання сценаріїв при плануванні версій

Г.Буч: Сценарії повинні групуватись таким чином, щоб для чергової версії вони спільно забезпечували ***реалізацію значної частини поведінки*** ПС та вказували на необхідність розгляду ***наступного найбільшого ризику***.

(Пріоритет - Rank - можна встановлювати для прецедентів при їх специфікації).

Г.Буч рекомендує проектувати ***5±2*** проміжних випусків системи.

Між прецедентами можуть існувати семантичні ***залежності***, які доцільно представляти у діаграмах (для зображення ***відношень залежностей*** використовуються ***пунктирні стрілки***).

Найважливішими випадками ***відношень залежності*** є відношення ***включення*** та ***розширення.*** Вони мають важливе значення стосовно до, мабуть, найголовнішої стратегії у програмування - стратегії повторного використання коду.

Організація прецедентів. Відношення включення (*include*)

Відношення включення (***include***) між прецедентами означає, що в деякій "точці" базового прецеденту як складова частина використовується поведінка іншого прецеденту. Прецедент, що включається, ніколи не використовується автономно (з точки зору UML він розглядається як ***абстрактний***, - див. специфікацію прецедентів в *UML*, а саме прапорець *Abstract*), він використовується тільки як частина більш загального прецеденту. Можна вважати, що один прецедент запозичає, використовує поведінку (функціональність) іншого прецеденту (того, що включаються, абстрактного). (Зауважимо, що імена абстрактних прецедентів зображаються курсивом).

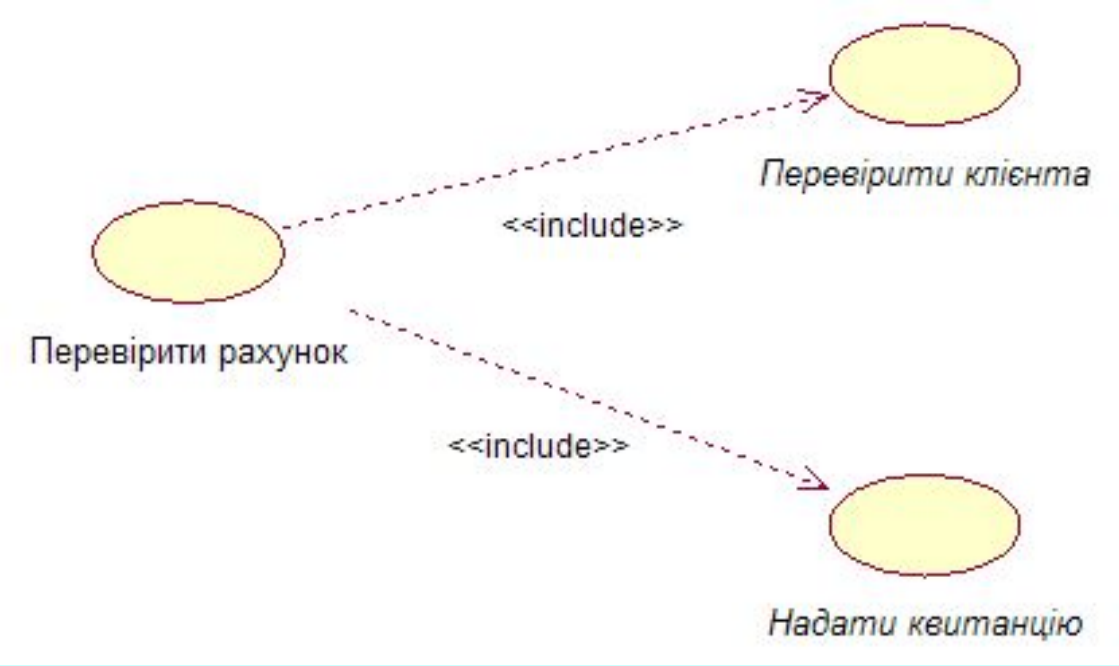
Завдяки наявності відношення включення вдається уникнути багаторазового опису потоків подій, оскільки спільну поведінку можна описати у вигляді самостійної поведінки, що включається в інші. (Зрозуміло, що це має безпосереднє відношення до стратегії ***повторного використання коду***).

Організація прецедентів. Відношення включення (*include*)

Щоб специфікувати місце в потоці подій, де саме базовий прецедент включає поведінку іншого, просто пишеться слово *include*, за яким іде ім'я прецеденту, що включається.

Приклад: `include("Перевірити клієнта")`.

На діаграмі прецедентів відношення включення зображують у вигляді ***залежності*** зі ***стереотипом include*** (пунктирна стрілка спрямована ***від*** базового прецеденту до того, що включається, абстрактного).



Організація прецедентів. Відношення розширення (*extend*)

Відношення розширення (extend) застосовують для моделювання таких частин прецеденту, які користувач сприймає як необов'язкову поведінку системи. Тим самим можна розділити обов'язкову й необов'язкову поведінку. Відношення розширення використовується також для моделювання окремих субпотоків, виконуваних тільки при певних обставинах.

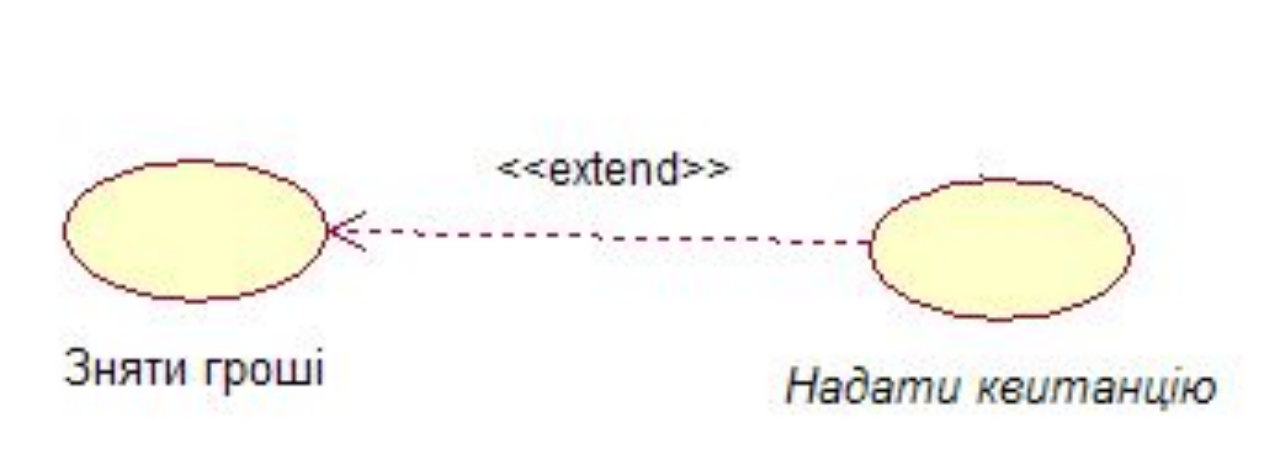
Мотивації цього відношення ті ж самі, що і у випадку відношення включення.

Організація прецедентів. Відношення розширення (*extend*)

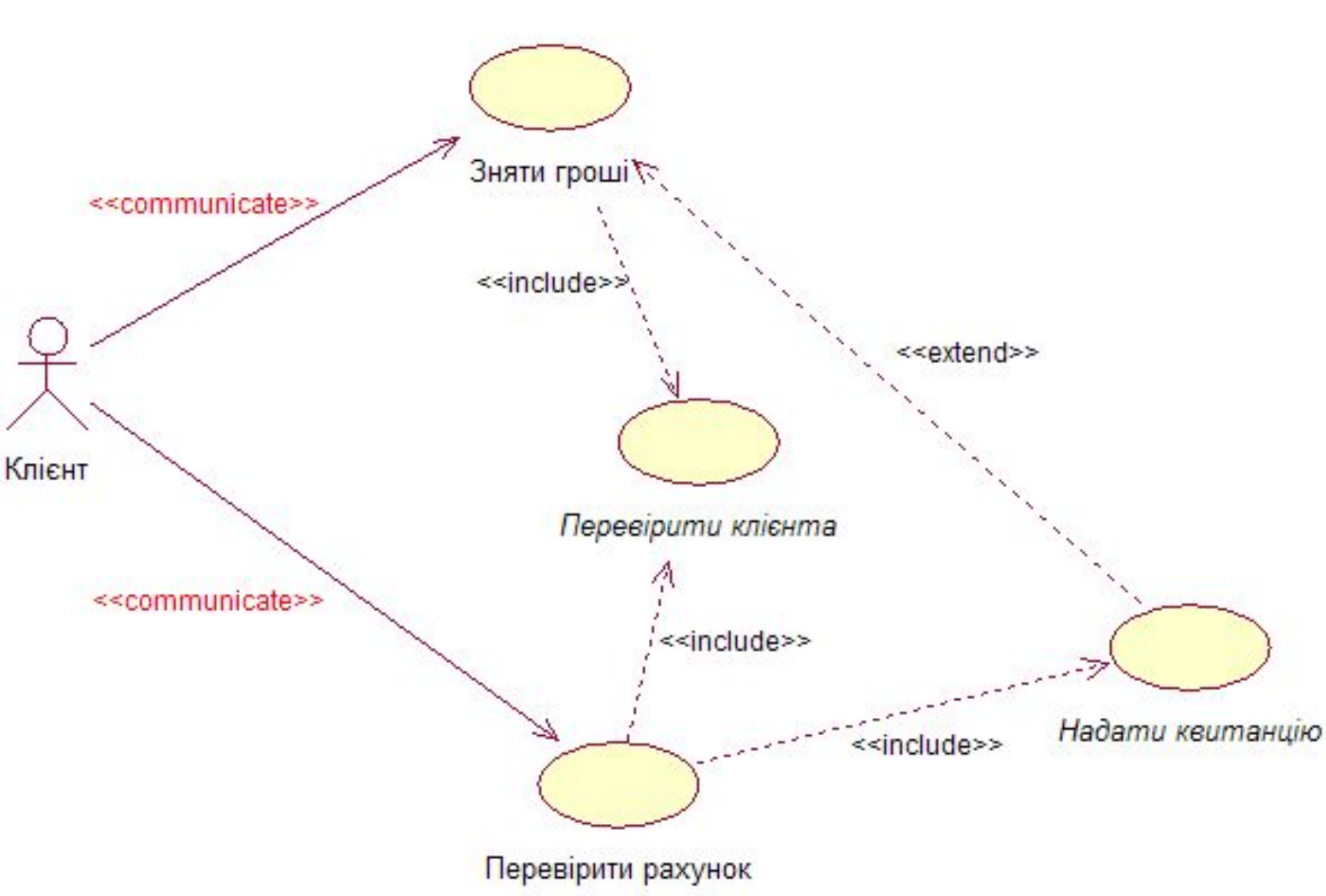
У потоці подій вказуються ***точки розширення.*** Потік може містити кілька точок розширення, ідентифікованих іменем прецеденту, що використовується для розширення.

Приклад: `При обранні . . . extend("Надати квитанцію").`

На діаграмі прецедентів відношення розширення зображують у вигляді ***залежності*** зі ***стереотипом extend***. Пунктирна стрілка має бути спрямована ***до*** базового прецеденту (до прецеденту, який розширюється), ***від*** абстрактного (того, що розширює).



Організація прецедентів. Відношення включення і розширення



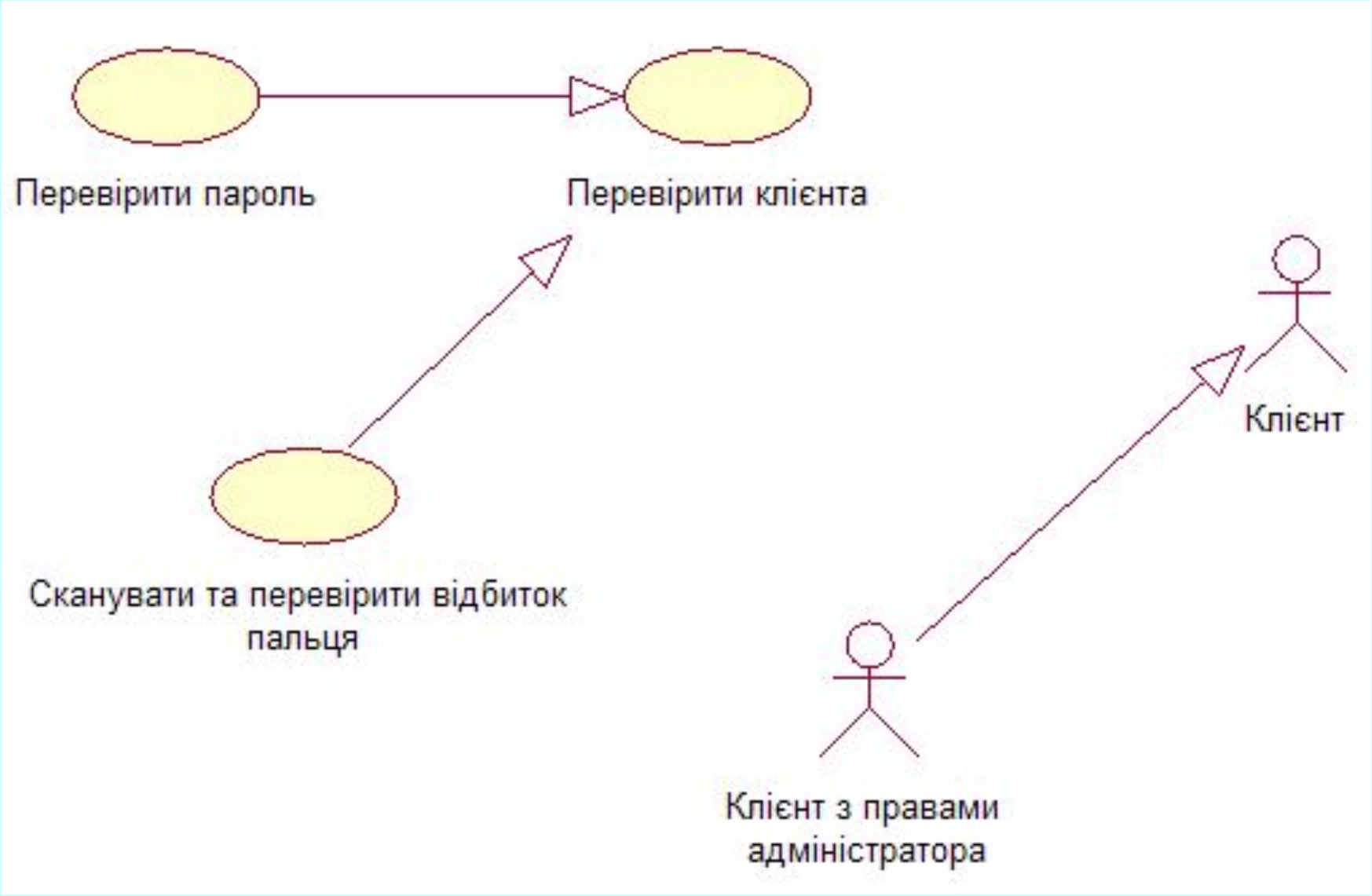
Відношення узагальнення

Відношення узагальнення (generalization) між ***прецедентами*** аналогічне відношенню узагальнення між класами в ООП і означає, що прецедент-нащадок успадковує поведінку й семантику свого батька, може заміщувати чи доповнювати його поведінку та, крім того, може бути підставлений усюди, де з'являється його батько. Відношення узагальнення між прецедентами зображуються у вигляді лінії з не зафарбованою стрілкою (так само на діаграмах зображуються узагальнення між класами, діячами).

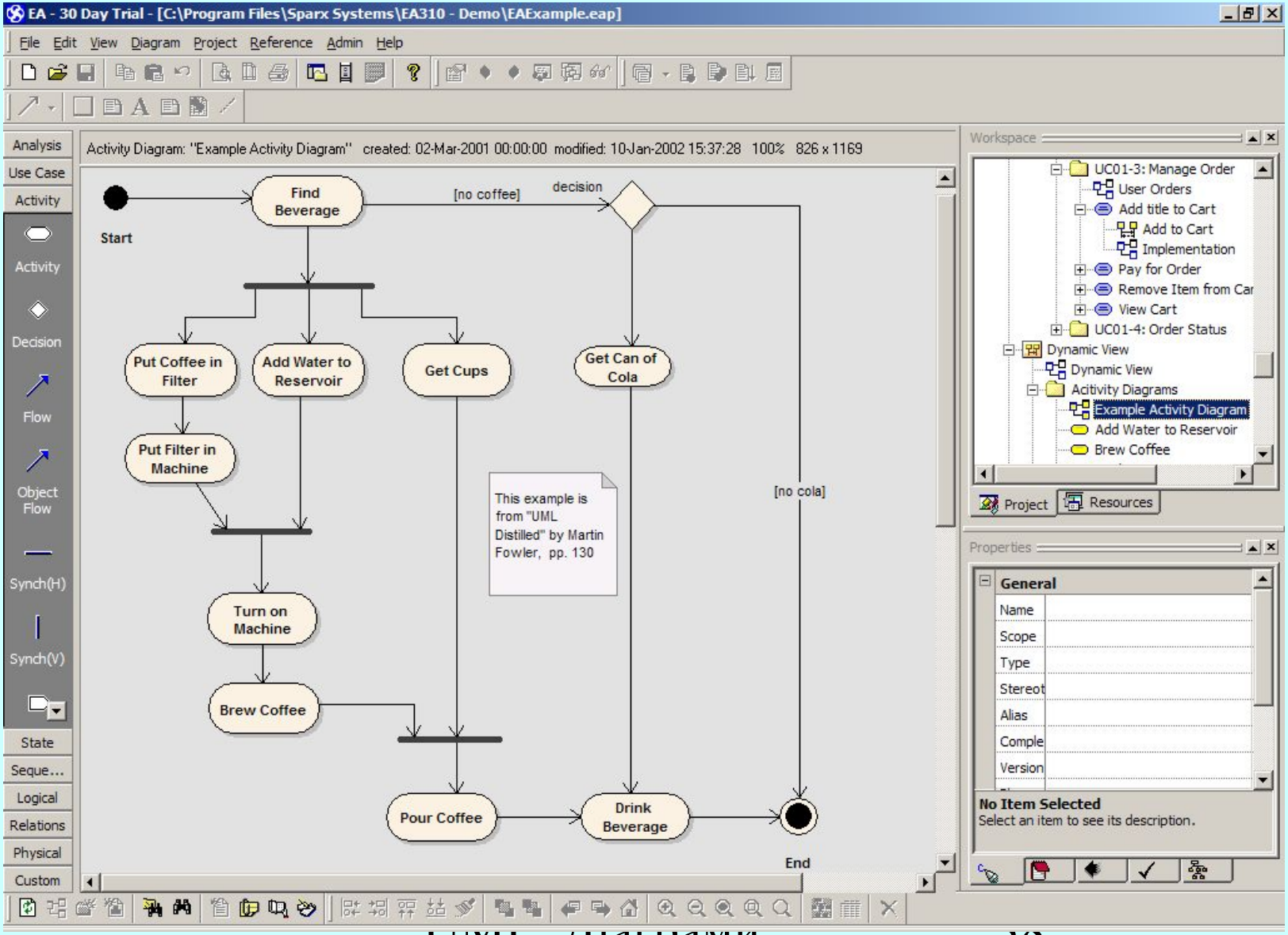
Відношення ***узагальнення*** можуть застосовуватись і між акторами. Це, можливо, більш звично. До того ж, у Rational Rose для специфікації акторів використовується та ж сама форма, що і у випадку класів (тобто актори по суті розглядаються як окремі випадки класів).

Узагальнення - це єдиний тип відношень, який може задаватись між ***акторами***. (Можна, наприклад, визначати загальні типи акторів, а потім спеціалізувати їх, створюючи різновиди.)

Актори та відношення



Діаграма діяльності




About Enterprise Architect
Enterprise Architect
Version 3.10
parx ystems
Enterprise Architect
Author: Geoffrey Sparks
Program Version: 3.10 (Build: 497)
Copyright © Sparx Systems 1998-2002
Database Details:
Database Version: 3.11
Version Date: Jan-14-2001
Registration Details:
Registered Enterprise Architect
(Professional Edition)
Licensed User
Thank you for registering this product
Close

Діаграма діяльності