1. Класові стереотипи «Boundary», «Control», «Entity».

Стереотипи дозволяють створювати нові будівельні блоки (точніше, похідні від існуючих), які є специфічними для розв’язуваної задачі. Стереотипи розширюють словник UML. Стереотипи класів для використання на етапі аналізу ПС: прикордонні (boundary) або інтерфейсні класи; класи-сутності (entity); управляючі (control) класи (класи-менеджери).

Прикордонні (boundary) або інтерфейсні класи моделюють взаємодію (інтерфейс) між основним актором та прецедентом (зовнішньо залежна частина ПС, яка навряд чи може бути використана в інших системах).

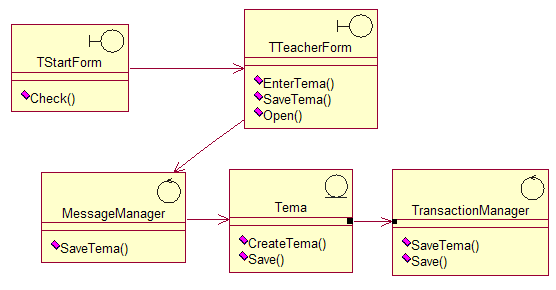
Управляючі (control) класи або класи-менеджери відповідають за координацію дій, поведінки (об'єктів) у процесі реалізації деякої функціональності ПС, зокрема у процесі реалізації функціональності деякого прецеденту.

Класи-сутності (entity) моделюють ключові абстракції предметної області, пов'язані з обробкою та збереженням інформації програмною системою (такі ключові абстракції, як правило, є незалежними від конкретної ПС, а отже можуть успішно використовуватись в інших ПС). Класи-сутності часто містять інформацію, що має постійно (тривалий час) зберігатись в одній чи декількох таблицях БД.

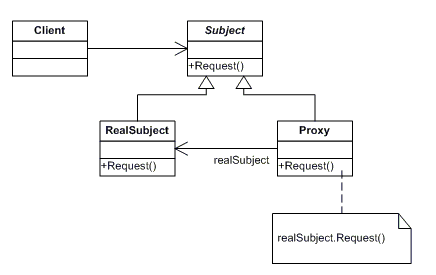
2. VOPC-діаграми.

Графічна реалізація прецедентів полягає у створенні однієї чи декількох діаграм взаємодії (послідовності чи кооперації), розроблених у відповідності до основних сценаріїв прецедентів; діаграм класів-учасників – VOPC (View of Participating Classes). Призначення діаграм VOPC - відображення класів, що беруть участь у реалізації варіанту використання та зв’язків між цими класами. Зазвичай вона використовується після опису потоку подій в описі варіанту використання.

Діаграма класів-учасників VOPC (View of Participating Classes) до сценарію «Викладач формулює тему дипломної роботи»:



3. Патерн «Proxy» та його використання для віддаленої взаємодії.

Патерн – це типове вирішення типової проблеми у даному контексті. Proxy («Заступник») виступає сурогатом (заступником) іншого об'єкта, «контролюючи» доступ до нього. 

Проблема: необхідно контролювати доступ до об'єкта, не змінюючи при цьому поведінку клієнта; необхідно мати доступ до об'єкта так, щоб не створювати реальні об'єкти безпосередньо, а через інший об'єкт, який може мати додаткову функціональність.

Рішення: створити сурогат реального об'єкта. «Заступник» зберігає посилання, яке дозволяє заступникові звернутися до реального суб'єкту (об'єкт класу «Заступник» може звертатися до об'єкта класу «Суб'єкт», якщо інтерфейси «Реального Суб'єкта» і «Суб'єкта» однакові). Оскільки інтерфейс «Реального Суб'єкта» ідентичний інтерфейсу «Суб'єкта», так що «Заступника» можна підставити замість «Реального Суб'єкта», контролювати доступ до «Реального Суб'єкта», відповідати за створення або видалення «Реального Суб'єкта». «Суб'єкт» визначає загальний для «Реального Суб'єкта» і «Заступника» інтерфейс так, що «Заступник» може бути використаний скрізь, де очікується «Реальний Суб'єкт». При необхідності запити можуть бути переадресовані «Заступником» «Реальному Суб'єкту».

4. Прозорість віддаленої взаємодії з об’єктами.

Під прозорістю розподіленої системи розуміють її здатність приховувати свою розподілену природу, а саме розподіл процесів і ресурсів по безлічі комп’ютерів та представлятися для користувачів і розробників додатків у вигляді єдиної централізованої комп'ютерної системи.

COM забезпечує як прозорість доступу, так і прозорість місцерозташування в розподілених (клієнт-серверних) системах. Як програміст, так і користувач клієнтської програми не бачать (прозорість!) різниці між різними варіантами COM-серверів при використанні останніх, не зважаючи на необхідність у різних засобах (механізмах) COM та Windows для підтримки таких варіантів серверів.

Прозорість доступу (англ. Access transparency). Незалежно від способів доступу до ресурсів і їх внутрішнього подання, звернення до локальних і віддалених ресурсів здійснюється однаковим чином. На базовому рівні приховується різниця архітектур обчислювальних платформ, але, що більш важливо, досягається угода про те, як ресурси різнорідних машин будуть представлятися користувачам розподіленої системи єдиним чином.

Прозорість розташування (англ. Location transparency) дозволяє звертатися до ресурсів без знання їх фізичного розташування. У цьому випадку ім'я запитуваного ресурсу не повинно давати жодного уявлення про те, де ресурс розташований. Тому важливу роль для забезпечення прозорості розташування грає іменування ресурсів.

Прозорість перенесення: приховується факт переміщення ресурсу в інше місце.

Прозорість зміни місця розташування: приховується факт переміщення ресурсу в процесі обробки в інше місце.

Прозорість реплікації: дозволяє приховати той факт, що існує кілька копій ресурсу.

Прозорість паралельного доступу: приховується факт можливого спільного використання ресурсу декількома конкуруючими користувачами.

Прозорість відмов: приховується від користувача, що він використовує ресурс, який не в змозі правильно працювати і знаходиться тимчасово в процесі відновлення.

Прозорість збереження: приховується зберігається ресурс на диску чи знаходиться в оперативній пам'яті.

5. Маршалінг і демаршалінг даних у випадку віддаленої взаємодії.

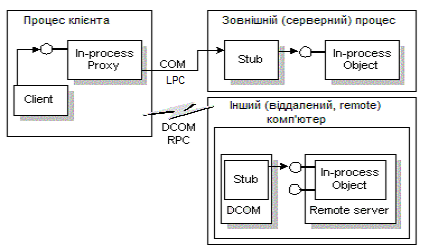
Proxy отримує виклик від клієнта і передає повідомлення об'єкту сервера, здійснюючи перетворення (маршалінг) даних для їх передачі (в інший процес чи по мережі). Маршалінгом (marshaling) прийнято вважати процедуру перетворення даних з метою їх подальшої передачі за межі процесу.

Два proxy-об'єкти обмінюються собою пакетами ("marshaling packet”).

Один proxy-об'єкт здійснює маршалінг даних (функцією COM API CoMarshalInterface) і пакет передається іншому процесу – іншому proxy-об'єкту, який здійснює демаршалінг (функцією CoUnMarshalInterface).

Подібна технологія з використанням proxies та marshalling packet є традиційною при здійсненні викликів віддалених процедур (RPC – Remote Procedure Calls).

(Обмеження: типи, сумісні з RPC- маршалінгом).



LPC чи LRPC (Lightweight Remote Procedure Call) – тип міжпроцесного зв'язку, який є, по суті, спеціальним випадком RPC.

У багатьох випадках при розробці серверів автоматизації можна не перейматися проблемами маршалінгу, покладаючись на так званий стандартний маршалінг автоматизації (зі стандартними proxy). Зокрема, майстер Delphi генерує модулі для підтримки об'єктів автоматизації з орієнтацією саме на використання стандартного маршалінгу. Але поруч з плюсами є й мінуси – обмеження на типи, що можуть використовуватись інтерфейсними методами (по суті, треба враховувати два обмеження: типи, що підтримуються маршалінгом RPC та типи, сумісні з автоматизацією). Загалом, майстер обмежується такими типами Delphi: Smallint, Integer, Single, Double, Currency, TDateTime, WideString, IDispatch, SCODE, WordBool, OleVariant, IUnknown, Byte. Тип String – відсутній.