Руслан Богатырев

О расширяющем программировании в языке Оберон

В чем практическая выгода использования расширяющего программирования (extensible programming) в трактовке языка Оберон?

Прежде всего, следует сказать, что расширяющее программирование является симбиозом модульного программирования и ООП, направленным на создание расширяемых программных компонентов.

Рассмотрим преимущества расширяющего программирования на примере достаточно распространенной на практике задачи.

Есть некоторое программное устройство/приспособление (плеер, калькулятор, текстовый редактор и т.п.). Запрограммирована и отлажена логика его работы. Управление осуществляется через интерфейс, на который опираются другие (внешние) устройства/приспособления (от термина "компонент" пока воздержимся).

Предположим, что в ходе эксплуатации нашего устройства выявились некоторые нюансы, которые требуют введения дополнительных "ручек" управления (для тонкой настройки, индикации режимов, адаптации к особенностям использования и т.п.). Встает задача такого расширения функционала устройства, при котором не потребовалось бы вносить изменения в зависящие от него устройства. Более того, в идеале хотелось бы подменить "на лету" старый вариант устройства на его расширенный вариант.

Как быть? Оберон дает простой и эффективный ответ на подобный вопрос: функционал данного устройства можно расширять бесшовно путем добавления в интерфейс новых процедур, при этом не затрагивая другие устройства (без их перекомпиляции) и без использования какого бы то ни было объектно-ориентированного программирования (это достигается за счет средств раздельной компиляции, использующих понятие основного и расширенного интерфейса).

Остается добавить, что весь этот процесс осуществляется в режиме жесткого контроля соответствия интерфейсов по экспорту/импорту на уровне языка программирования со статической типизацией (т.е. Оберона) и что подобные устройства (компоненты), характеризующиеся длительным временем жизни (persistence), в системе Oberon System 3 (ETH Oberon) называются словом gadget (приспособление).

Почему для таких задач расширяющее программирование (включающее в себя в случае надобности и средства ООП) лучше подходит, нежели традиционное ООП на основе классов? Как известно, любое специальное устройство можно реализовать эффективнее соответствующего универсального. И если нет необходимости жертвовать эффективностью и надежностью во имя пресловутой универсальности, то выбор всегда будет в пользу специализации.

Как пример. Я как потребитель не хочу покупать костюм на вырост. Мне не нужно заботиться и о том, что в будущем на его основе кто-то где-то будет выпускать собственную коллекцию одежды. Но мне может потребоваться после приобретения костюма внести несущественные изменения, не требующие универсальности (обобщения на уровне классов), например, заменить пуговицы. Расширяющее программирование справляется с этим очень эффективно. Ну а если я (уже не как потребитель, а как дизайнер) все-таки прийду к мысли, что костюм вполне созрел для создания на его основе целой коллекции (но для этого осмысления должно пройти достаточное время), то в том же расширяющем программировании можно будет задействовать как средства универсализации (ООП), так и средства настраиваемой/расширяемой специализации (модульное программирование).

Наконец, расширяющее программирование в трактовке классического Оберона выступает в двух разновидностях: расширение в большом, т.е. на уровне модулей (о котором говорилось выше), и расширение в малом, на уровне типов данных (type extension). Если учесть, что класс в Обероне трактуется как расширяемый тип, заключенный внутрь модуля, то нетрудно сделать вывод, что подобная схема (варьирования специализацией/универсализацией) более надежна и одновременно более гибка, нежели традиционное промышленное ООП.