Business Plan Design Document

BYB Team

Purpose

The motivation for the project is to create an architecture to store data needed for a business/assessment plan that represents the relationships between the abstract goals and the concrete action plans. For example, we should be able to represent an assessment plan at college with a vision statement, the mission statements to achieve the vision, and the concrete action plans to fulfill the missions.

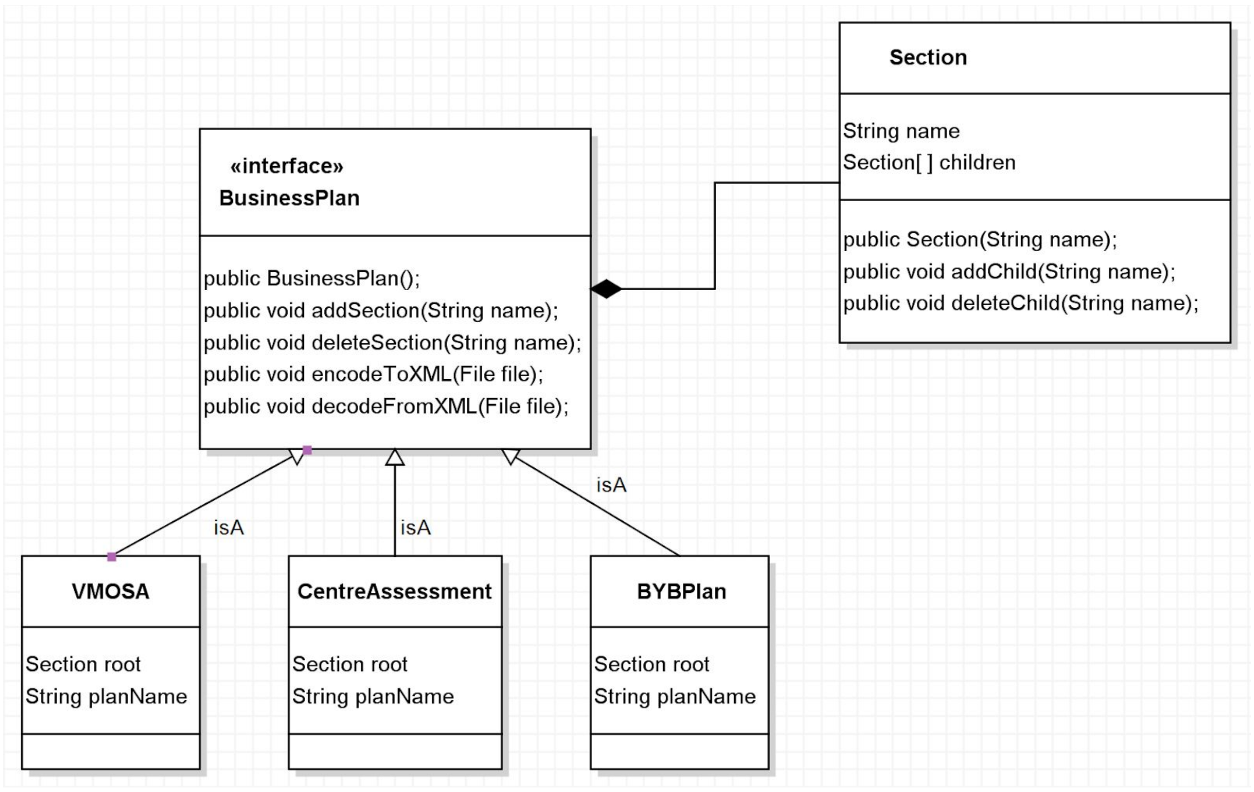
Specifications

Dr. Bradshaw needs an application that allows him to review the Centre College Assessment Plan in a way that can be reused every year. The users will fill out the assessment plan in an manner that establishes concrete relationships between objectives/goals and the strategies and plans of action for fulfilling them.

Our design is very flexible and very extendable, as it is able to represent a large family of business plans and can be further extended with little effort to assess the success of those plans. This is due to the basic tree structure which naturally establishes a hierarchical relationship; for example, each strategy is dependent on an objective (or ‘tied’ to an objective) and each objective is dependent on a mission statement or a vision.

This design represents the VMOSA business plan, the Centre College Assessment Plan and the Beyond Borders (BYB) Business Plan. These plans can be stored in an XML file and can be read back into memory.

Design Overview



«interface»
BusinessPlan
public BusinessPlan();
public void addSection(String name);
public void deleteSection(String name);
public void encodeToXML(File file);
public void decodeFromXML(File file);
Section
String name
Section[ ] children
public Section(String name);
public void addChild(String name);
public void deleteChild(String name);
isA
isA
isA
VMOSA
Section root
String planName
CentreAssessment
Section root
String planName
BYBPlan
Section root
String planName

Figure 1: Class Diagram for Business Plan

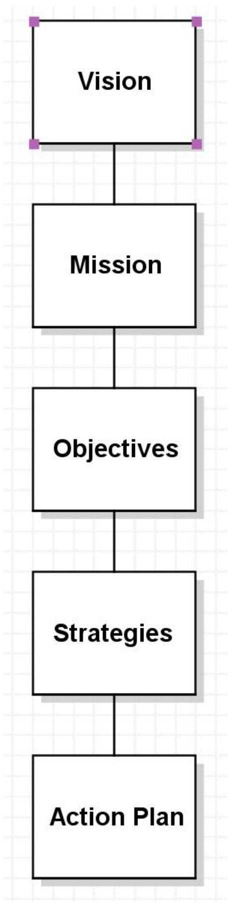


Figure 2: The Tree Structure of Nodes

We use the composite pattern for our design to compose objects into tree structures to show the hierarchies between objects. For example, vision statements are always the very first thing in a business plan, so it should be on the top of the hierarchy (in Figure 2), which means there will be nothing without first creating vision statements.

Section is basically a node in a tree data structure (on the right-hand side in Figure 1). Every node has its own name and a list of children. For example, in VMOSA, there will be five sections and they are called Vision, Mission, Objectives, Strategies, and Action Plan. The methods addChild() and deleteChild() are used to modify the children of each node. Except for the first section in each of these plans, the other sections are both children and parents at the same time.

BusinessPlan is an interface because there will be more concrete plans in the future. In this way, more plans can be created without altering the existing code. There are three classes that implement the business plan for now - VMOSA, Centre's Assessment Plan, and our team plan. The concrete classes of the BusinessPlan interface implement the methods of adding and deleting sections in order to be able to modify multiple sections for each level of the business plan. For example, there could be two Mission sections. Since this architecture is based on a tree structure, each concrete business plan class has its own root section as an attribute. For example, Vision is the root node in VMOSA. They also have an attribute called planName that indicates the name of groups that utilize the business plan, such as Centre's Assessment Plan: Political Department.

Subsystems Specifics

There are two parts of our design that are worthy to talk about in detail: the BusinessPlan Interface and Each BusinessPlan concrete classed.

- BusinessPlan Interface has:
    - Attribute: Node: root, which is an abstract variable
    - Constructor to create all the necessary nodes & set their head to the root.
    - Skeleton methods for adding and deleting sections.
- Each BusinessPlan concrete classes:
    - Implement the BusinessPlan Interface, so they have all the methods and the root from the interface but with concrete codes. However, the planName variable will be set by passing into constructor when we actually create objects.
    - May have future features which is an array of Elements.

Future Features

In addition to what we have for now, we also considered some future features which may be added to the current structure.

The first feature is the changeable boolean. When we discuss the structure of the business plan, we talk about how we wanted to keep the structure of the business plan complete which means we want each plan to have all of its sections that are pre-defined. For example, for the VMOSA business plan, we want to make sure that it always has five sections, but if users delete some sections and do not add new sections back to the plan, the structure will be incomplete. In order to solve this problem, we think of adding a boolean attribute called changeable to the node class to indicate if the user can delete the section. We will make part of the tree as the main tree which is used to maintain the structure of the plan complete. If the boolean changeable is set to false, then when we call the delete method, we cannot delete it but modify its current content. If the boolean is set to true, we can delete it since the deletion will not affect the structure of the tree.

The other feature we think of is the Element interface. Although we currently have a string for the content of each section, adding the Element interface expands the number of options that could be used in the content of sections, such as text boxes, diagrams, and, tables. We have discussed if we want to create a data structure to represent such elements in each section, however, if we add the Element interface, then things get more complicated since we need to maintain the structure of the elements complete, so we just keep this plan and see if we need it when we actually implement it.