1) *Create the factory hierarchy*. This is accomplished by (a) creating an abstract factory classifier (an abstract class or interface that is intended to realize the target SRM *AbstractFactory* role) for each realization of the source SRM *AbsPrimaryProduct* role, (b) creating a concrete factory class (an intended realization of the target SRM *ConcreteFactory* role) for each realization of *ConcPrimaryProduct* role in the source model, and (c) linking them (using generalization or realization relationships) in accordance with the product hierarchy.

2) *Migrate the create operations from the realization of the Client role in the source model to the appropriate factory classes*. The allocation of create operations to factories is determined by the associations between the primary parts and their subparts : the create operations for each subpart linked to a primary part are placed in the factory corresponding to the primary part. This results in the removal of the create dependencies between the *Client* realization and the product classifiers, and creation of create dependencies between the factories and the product classifiers.

3) *Link the factory classes to the Client realization using associations or usage dependencies*.

Fig. 5 shows a class diagram that reflects the static, structural aspects of a design for a maze game (this design is an adaptation of an example given in [GAM 95]). In this design, the *MazeGame* (the client) is responsible for creating the different types of mazes and their parts. If a new type of maze or maze part is added, the *MazeGame* class would have to undergo significant change. Incorporating the Abstract Factory pattern into this design will result in a more flexible design in which the maze creation aspects are localized in factories that can be accessed by the MazeGame.

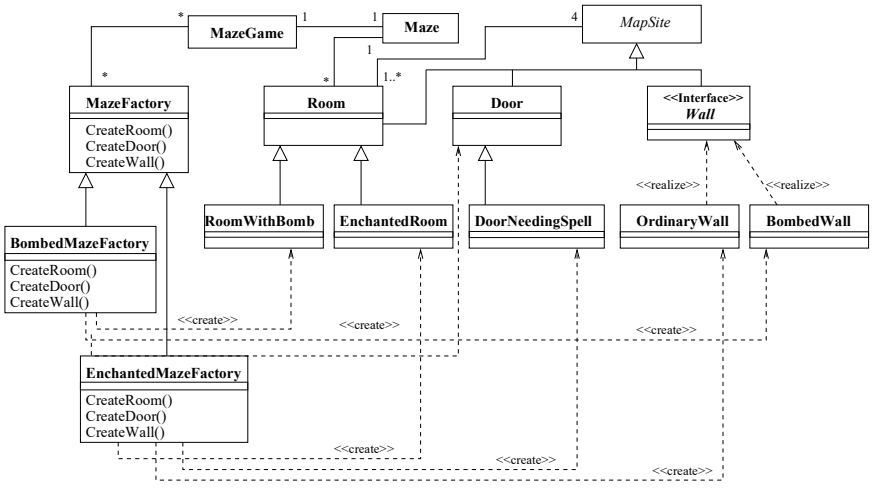

Figure 7. *Maze Game refactored with Abstract Factory pattern*