### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Favorites**  
  Our application allows marking posts in user’s friends news feed as favorites. Those posts are displayed in additional tab in our application’s view. User can remove specific post from favorites later. In order to mark/unmark a post as a favorite user can use context menu (right-click around post area) or click on favorite’s sign – star (image) – that located on the left side of the post message.
* **Translation**  
  Our application allows instant translating of a specific post to desired language using one of the implemented translators. In this exercise we allowed selection of one of three languages – Hebrew, English, and Russian – but it could be extend to dynamic list supported by translators.  
  We have implemented real translator by integration with the Microsoft’s Bing translation service. Both translators and supported languages options are loaded dynamically to the applications menu (*Translation*). There are 2 additional translators implemented for demo purposes – Dummy (adds “translated” work to the text) and Base64 (translates text to base64).

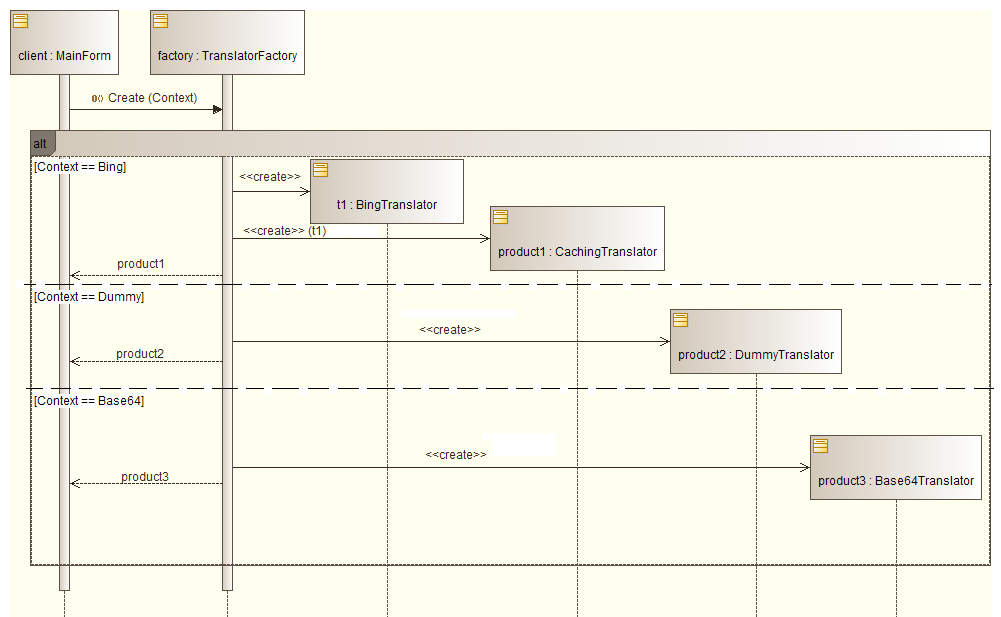
### תבנית מס' 1 – FactoryMethod

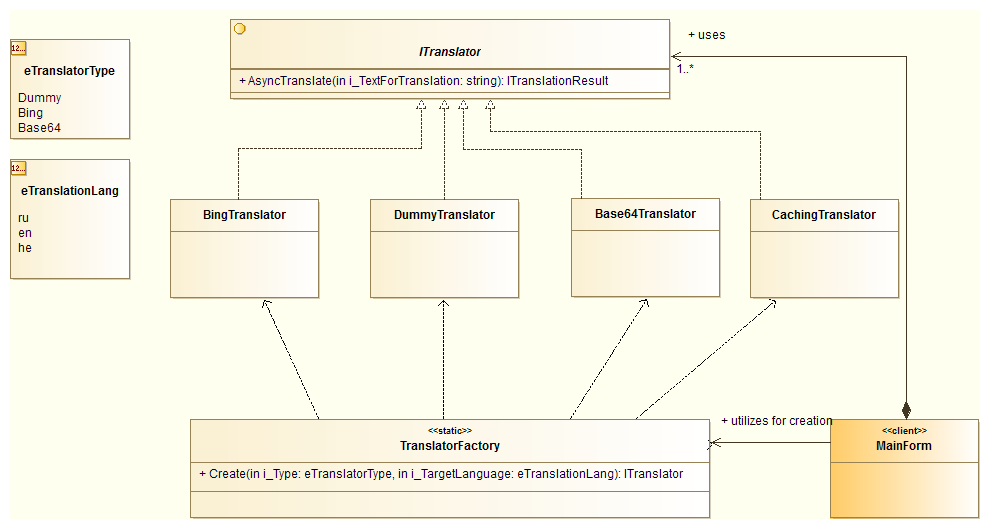
* **סיבת הבחירה / שימוש בתבנית:**

Our application supports different translators (implementations of ITranslator interface). In application’s main menu user can select a translator that he would like to use. When translation required, we need to create the concrete translator based on user’s choice (context of creation). We decided to place the creation logic (based on context) in a single place – factory method pattern.

* **אופן המימוש:**

We implemented static method (defined in static class) that returns the result as a reference to the ITranslator interface. Based on provided context (translator type) we create a concrete instance of the translator and return it. In case the type is unexpected - exception will be thrown.  
The implementation is located in the **FacebookApp.Model** project, in the **TranslatorFactory.cs** file under the **Translator** folder.

* **Sequence Diagram**  
  
* **Class Diagram**



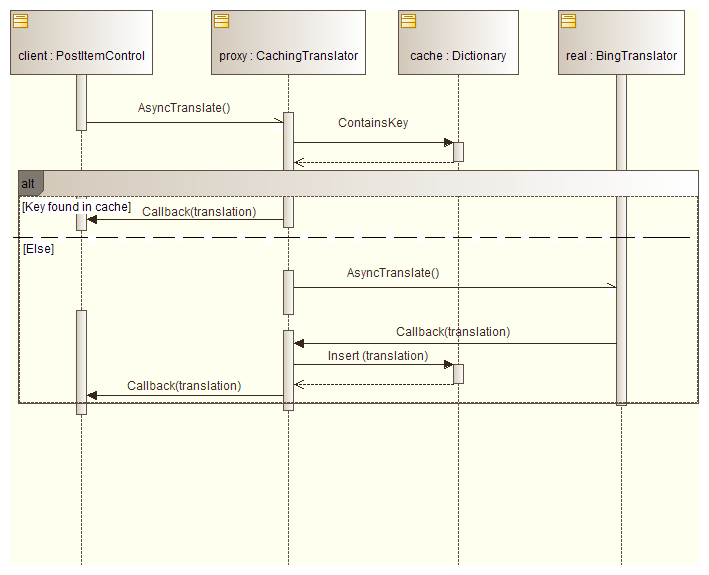
### תבנית מס' 2 – Proxy

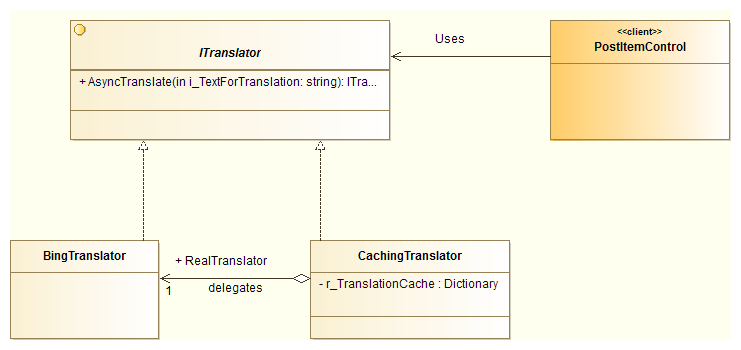
* **סיבת הבחירה / שימוש בתבנית:**

Some of translators (in our implementation – the “real” Bing translator) have “costy” translation operation – in terms of both time and money. In order to reduce those costs we decided to cache the results of text translation, however we want to make that caching logic transparent to the client – from the client’s perspective it works with a regular translator (ITranslator interface).

* **אופן המימוש:**

Our proxy translator – **CachingTranslator** – gets a reference to the real translator in the ctor and stores it as an instance member. Another stored member is a dictionary with key that is hash (md5) of the source text, and value which is the result of the previous translation of that text. When client asks to translate a text, first proxy computes the hash and checks whether it exists in the dictionary. If it does (cache hit) – the value returned from the dictionary. Otherwise proxy invokes the real translator, stores the result in the dictionary (and returns it as well).  
The implementation is located in the **FacebookApp.Model** project, in the **CachingTranslator.cs** file under the **Translator** folder.

* **Sequence Diagram**  
  
* **Class Diagram**



### תבנית מס' 3 –Facade

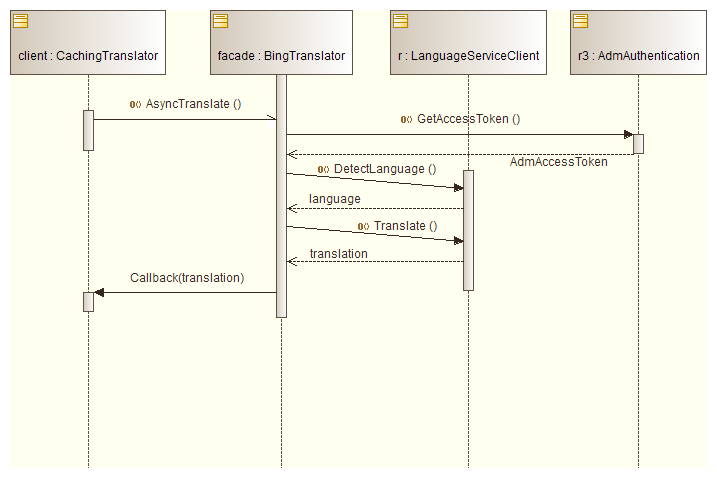
* **סיבת הבחירה / שימוש בתבנית**:

Getting translation from Bing Translation Service requires some additional set of operations to be performed in the specific order – authorization, getting access token, detecting source language and finaly the translation. We decided to hide all these implementation and process details from the client and provide it with the simple ITranslator interface, while the BingTranslator (façade) performs all the required operations in the correct order.

* **אופן המימוש**:

BingTranslator creates an instance of the LanguageServiceClient (client of the WCF real Bing translator service) and an instance of AdmAuthentication which initializes an AccessToken by sending appropriate HTTP request to the translator service. When the translator is asked to translate some text, it requests authentications from the service using AccessToken, asks the service to detect the text’s source language and then asks it to translate the text to the target language. Then it returns the result to the client using provided callback.  
The implementation is located in the the **FacebookApp.Model** project under the **Translator** folder. Files: BingTranslator.cs, AdmAuthentication.cs, AdmAccessToken.cs.

* **Sequence Diagram**



* **Class Diagram**   
  