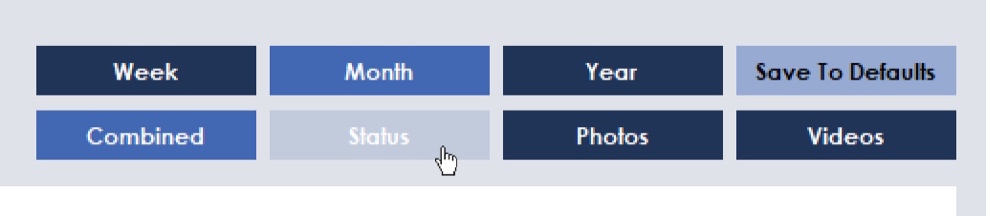
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

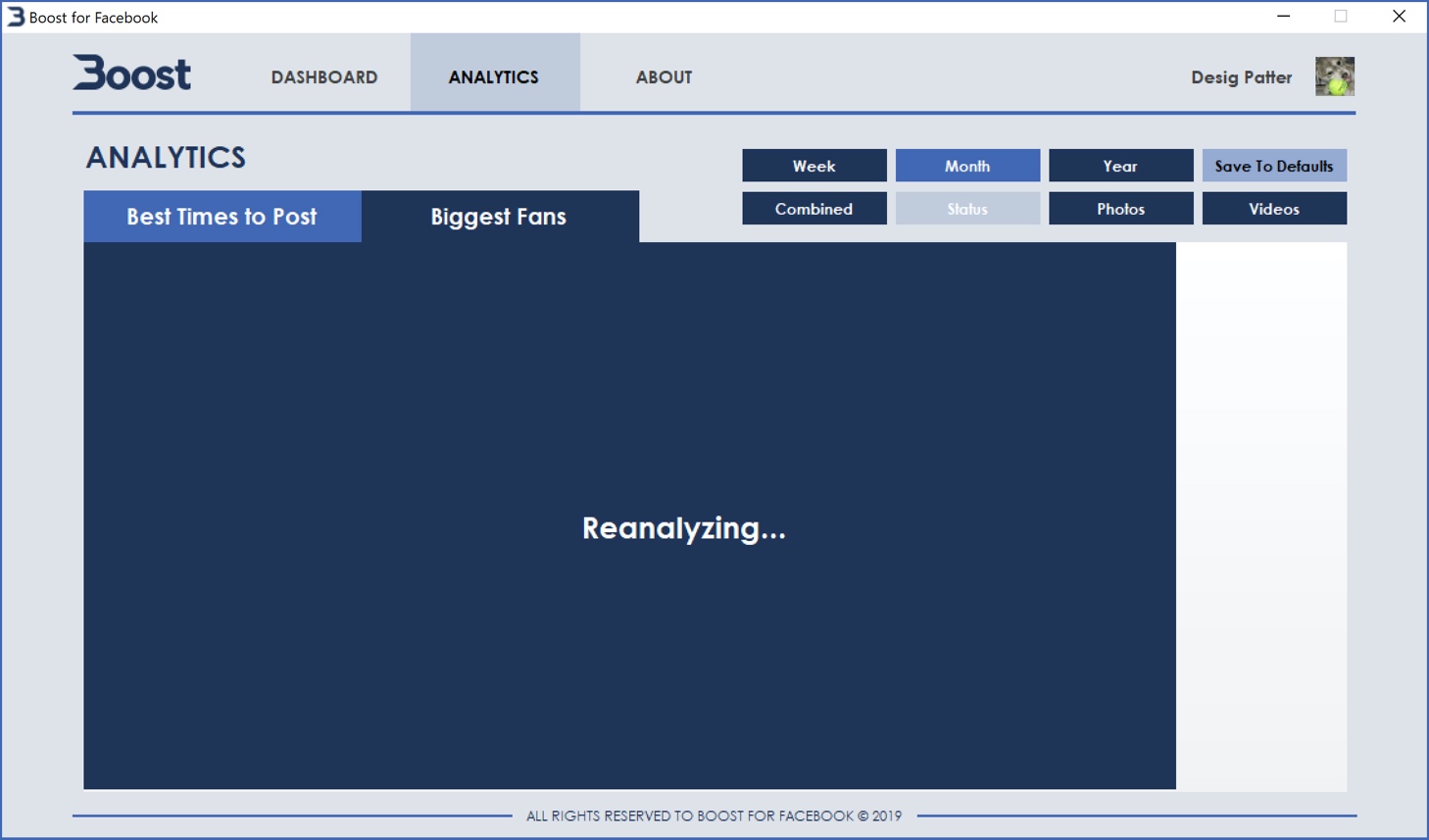
* - **Best Times to Post**מציאת הזמן הכי טוב לפרסם פוסט בחתך של תמונות, סטטוסים ווידאו ולפי קבוע זמן. בנוסף לכך, הצגה של מפת חום אשר מציגה את הזמן הטוב ביותר על גבי המפה לפי חתך כולל של כל הפרמטרים המוזכרים לעיל.
* **Biggest Fans** - מציאת החברים אשר עושים הכי הרבה לייקים לפוסטים של המשתמש, אותם אנו מגדירים כ״מעריצים הגדולים ביותר״ לפי חתך כולל של פוסטים.

### תבנית מס' 1 – Observer

* סיבת הבחירה / שימוש בתבנית:

כחלק מחוויית המשתמש, רצינו לשלב מרכיב ויזואלי בולט שיופיע לאחר בחירת אלמנט חדש לאנליזה בעזרת הכפתורים הבאים:

המרכיב הויזואלי שבחרנו ליישם הוא אנימציה,אשר מציגה למשתמש שנעשית אנליזה מחדש. אנימציה זו איננה שטחית:

* בשלב הראשון, ״אנימציית הכניסה״ ההתחלתית מופיעה בגלישה/צמיחה למסך עד שגודלה ישתווה לגודל חלון האב שלו.
* בשלב השני, ״אנימציית היציאה״ תחל רק אם התקבלה התרעה שהאנליזה החדשה הסתיימה.

לפי ההסבר הנ״ל, הבחירה בתבנית Observer ברורה.

* אופן המימוש:

בחרנו לממש את התבנית בעזרת Delegates ו-Events כאשר המחלקה בה נמצאת מתודת האנליזה (AnalyticsViewModel) מחזיקה ב-Delegate הנקרא AnalysisFinishedEventHandler() ו-Event הנקרא m\_AnalysisFinishedEvent. מתודת האנליזה, Analyze, מפעילה את האנליזות הספציפיות של שני הפיצ׳רים שלנו שלנו (BestTimesToPost ו-BiggestFans) בשני ת׳רדים נפרדים, כאשר הת׳רד הראשון (בעל עדיפות גבוהה יותר) עושה אנליזה לפיצ׳ר שחלונו כרגע מוצג למשתמש ובהתאמה, האנליזה השנייה נעשית בעדיפות נמוכה יותר ברקע.

מתודת ה-Invoke של ה-Event שהוזכר קודם לכן, מחוברת לסופו של הת׳רד הראשון (כסוג של מתודת Callback) ובכך תפעיל את כל המתודות הרשומות לאירוע בסופה של האנליזה (במקרה הספציפי שלנו, רק מתודה אחת).

תחת AnalyticsView, המכיל את חלון האנימציה reanalyzingOverlay (מופע של ReanalyzingView), אנו רושמים לאירוע m\_AnalysisFinishedEvent את המתודה האחראית להודעה לאנימציה על סיום האנליזה: AnalysisFinishedNotification.

המתודה הנ״ל, משנה ערך בוליאני במחלקת האנימציה ReanalyzingView במהלך האנימציה כפי שהוסבר בסעיף הקודם. אם האנליזה הסתיימה, ההודעה מתקבלת והשלב השני של האנימציה יחל, אחרת, האנימציה תישאר בשלב הראשון עד לקבלת ההודעה.

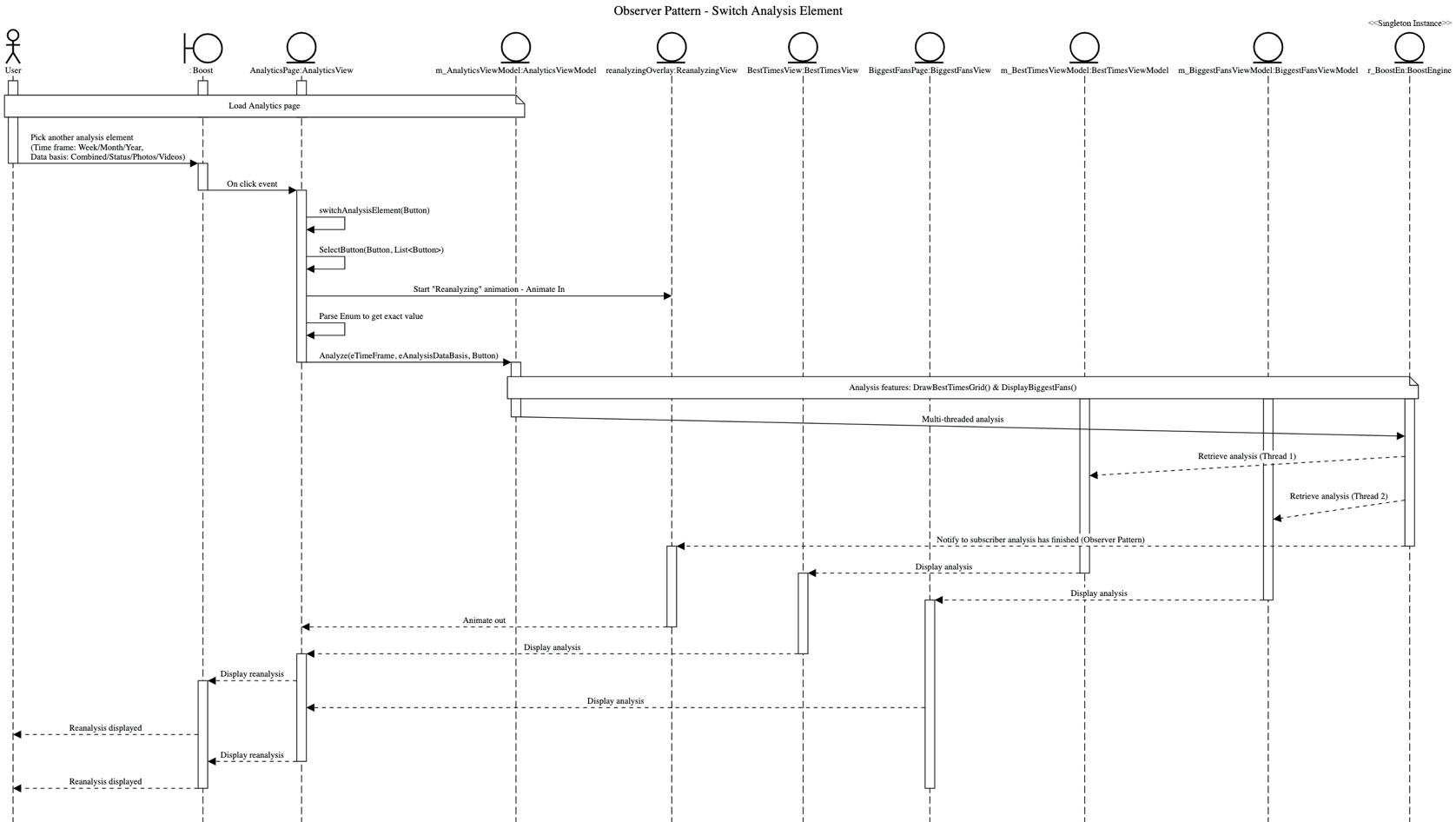
מיקום התבנית בקוד:

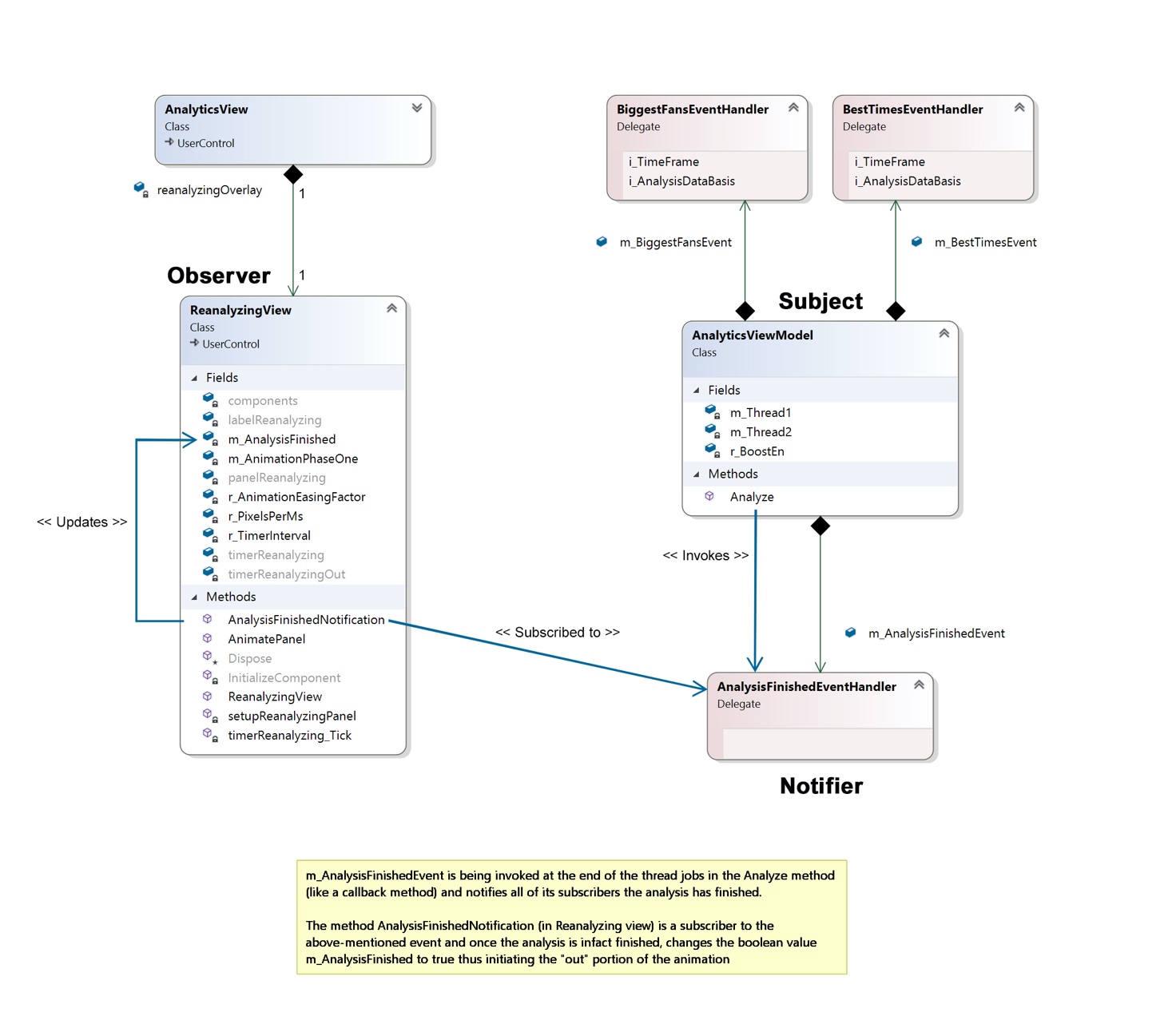
**Observervable Subject & Notifier:**

ViewModels🡺AnalyticsViewModel.cs

**Observer:**

View🡺ReanalyzingView.cs

* Sequence Diagram

Class Diagram

### תבנית מס' 2 – Decorator

* סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית עיצוב זו לצורך מיון של ערכי ה-Value ב-Dictionaries בהסתכלות עתידית אל מקרים בהם נרצה להוסיף סוג נוסף של מיון, או מיון בתוך המיון הנוכחי, לאנליזות שלנו.

הסיבה לפיה לא בחרנו להשתמש בתבנית Proxy היא המקרה בו נרצה לעשות סינון לסינון קיים בצורה שתמשר תחזוקתיות עתידית אופטימלית.

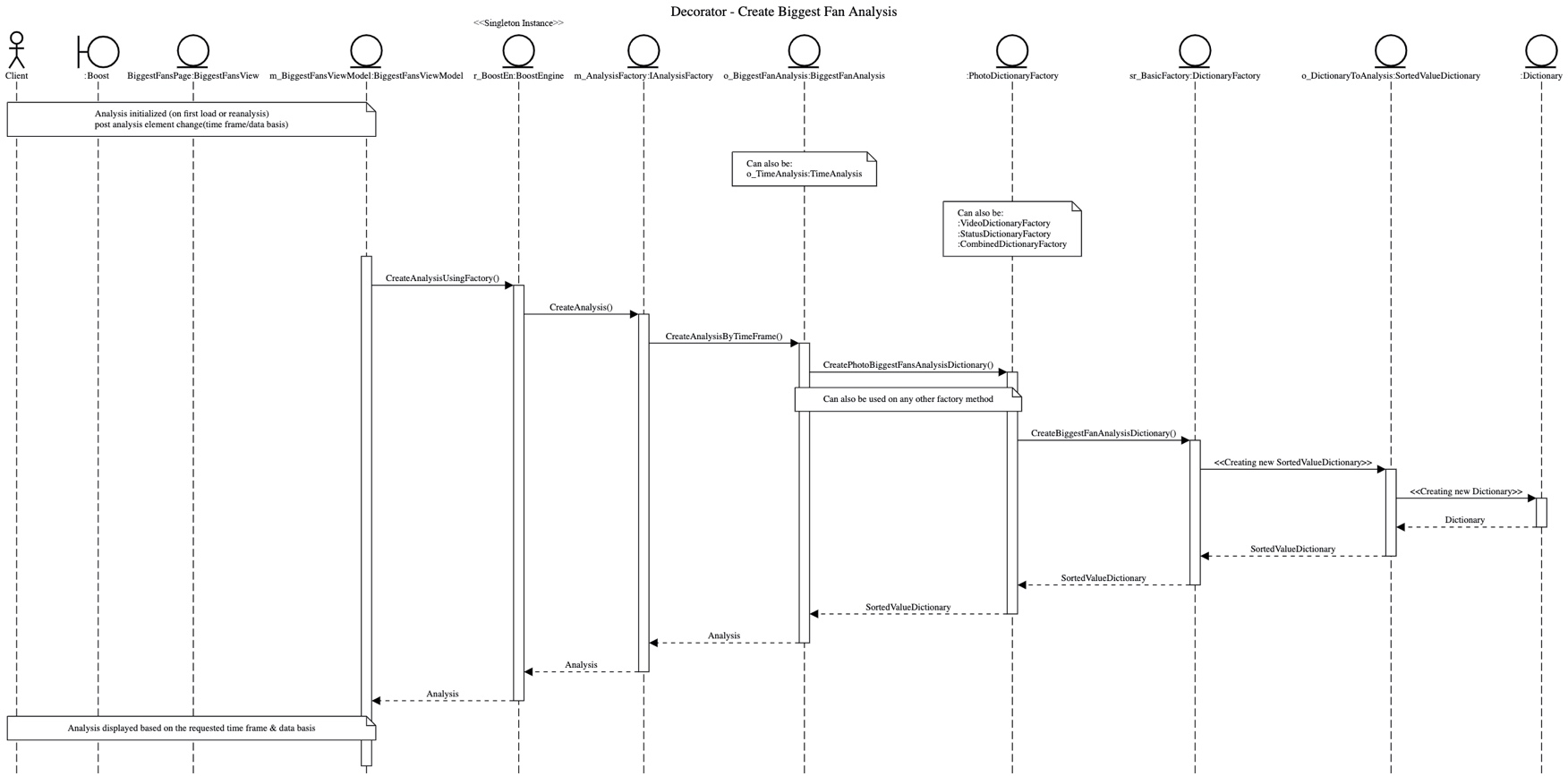
* אופן המימוש:

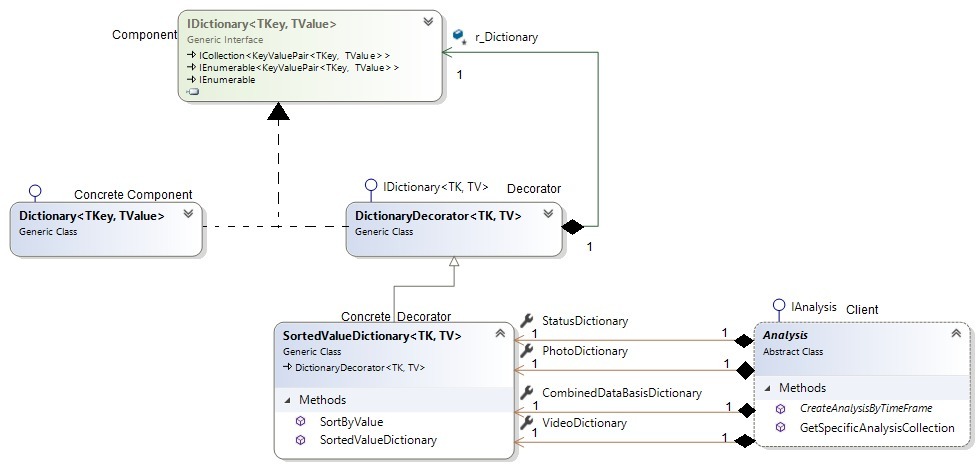
התבצע מימוש מהאינטרפייס IDictionary וקומפוזיציה אצל המחלקה המממשת DictionaryDecorator, מחלקה זו מממשת את Dictionary בצורה גנרית. בנוסף, יצרנוConcrete Decorator בשםSortedValueDictionary אשר מממש את המיון לפי ערכי ה-Value בהם חפצנו.

מיקום התבנית בקוד:

Model🡺DesignPatterns🡺Decorator.cs

* Sequence Diagram



* Class Diagram

### תבנית מס' 3 – Strategy

* סיבת הבחירה / שימוש בתבנית:

במהלך בניית ה-Factory Method נתקלנו בבעיה פונטציאלית של שכפול ובעייתיות בתחזוקה עתידית של הקוד. על מנת להתגבר על כך, מימשנו בעזרת פיצ'ר מהשפה שלמדנו במסגרת הקורס (Func) את תבנית Strategy Method. השתמשנו בתבנית כדי לגרום ל­-DictionaryFactory לדעת מה הוא צריך לפרסר (במקרה שלנו, אילו סוגי פוסטים) לפני החזרת Dictionary ממויין.

סיבה נוספת היא שימוש חוזר עתידי של הקוד אם נרצה לקבל Dictionary לפי סוג נוסף של מיון.

* אופן המימוש:

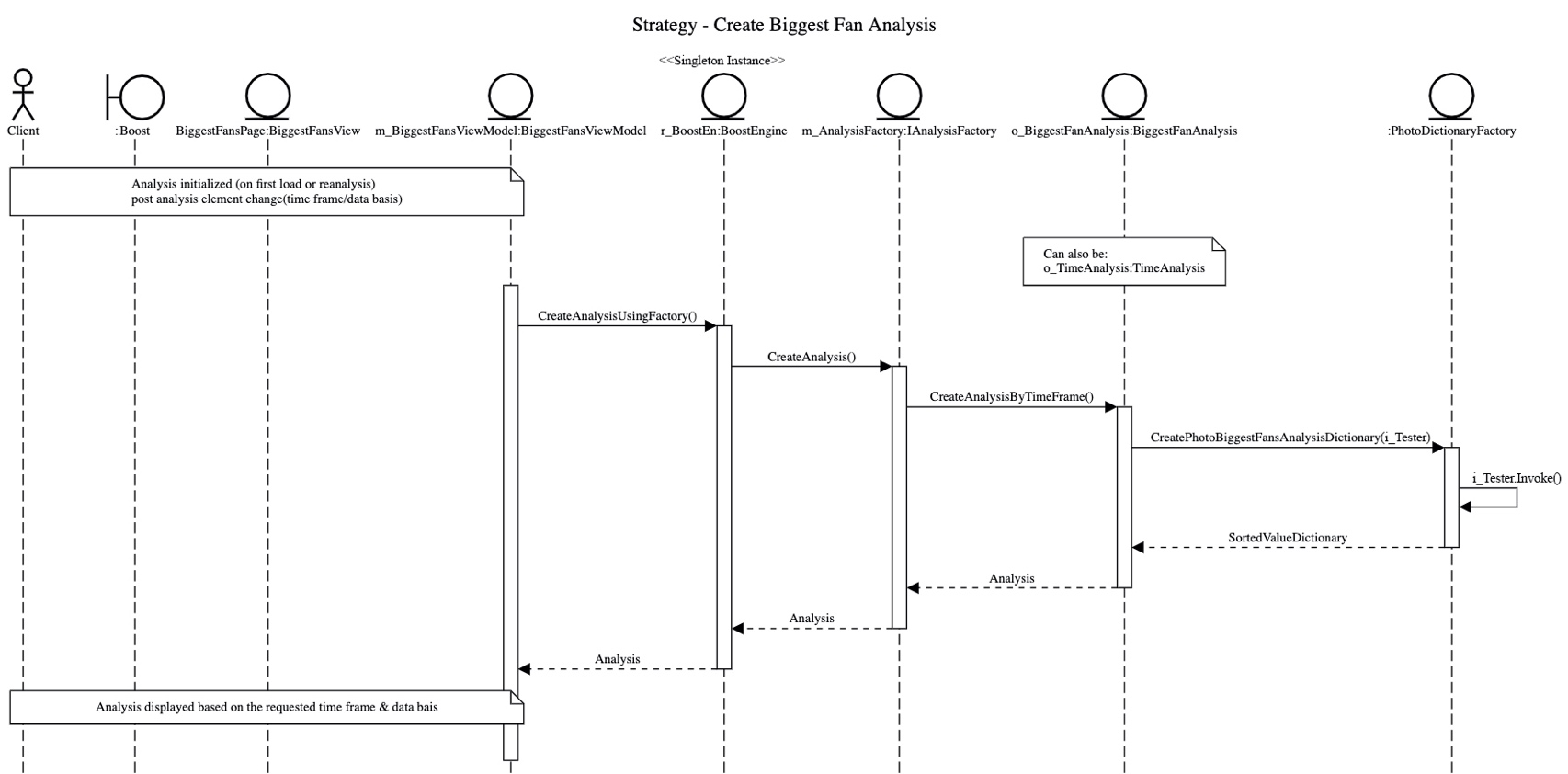
המימוש מבוצע על ידי שימוש במתודה המקבלת פרמטר מסוג Func<Post.eType?, bool> בתוכו מתבצעInvoke למתודת ה-Startegy בה רצינו להשתמש. הקריאה לאותה המתודה התבצעה על ידי שימוש ב- Lambda Expression(Inline Annonymous Function) המונע שימוש בבניית מתודה חיצונית ל-Strategy מאחר והיא בעל שורת קוד אחת בלבד.

מיקום התבנית בקוד:

Model🡺DesignPatterns🡺FactoryMethod🡺 DictionaryFactory.cs

והקריאה לאותה מתודה מתבצעת בשאר ה-Factories באותה התיקייה

* Sequence Diagram



* Class Diagram

