Journal of the KIMST, Vol. 27, No. 6, pp. 744-753, 2024
ISSN 1598-9127(Print) · 2636-0640(Online)
DOI https://doi.org/10.9766/KIMST.2024.27.6.744

Research Paper 정보통신 부문

비동기 함수를 사용한 MVVM 형태의 모듈화된 UI 소프트웨어의 시각적 동기화 기법

김진솔*,1) · 신나나1) · 이신영1)

1) LIG넥스원㈜ PGM통제기술연구소

Visually Synchronizing Modular UI Software Using Asynchronous Function

Jinsol Kim*,1) · Nana Shin1) · Shinyoung Lee1)

1) *PGM Control Technology Research Center, LIG Nex1, Korea*

(Received 20 June 2024 / Revised 3 September 2024 / Accepted 12 September 2024)

Abstract

As the modern warfare continuously evolves into one that is network-centric, the amount of data a weapon system has to handle is rapidly growing, which applies not just to the internal data processing software but also to the UI software. Not only is the amount of data growing, but also is the complexity of handling such data, which when combined together creates a significant delay in updating relevant components in a UI software. In the context of the defense industry, split-second delay in visual component update can lead to catastrophic failure in making strategic decisions. This paper presents a novel method that utilizes asynchronous functions to control the program flow and reduce the delay in updating multiple visual components of a modular software. We apply this method to our modular UI software which uses MVVM pattern, where we were able to reduce the delay by an average of 78.4 %.

Key Words : Synchronization(동기화), Modularization(모듈화), User Interface(사용자 인터페이스), MVVM Pattern(MVVM 패턴)

1. 서 론

현대 전쟁은 더 이상 단순하게 더 좋은 무기와 더 많은 병력이 승리를 가져다주지 않는다. 육군, 해군, 공군 모두가 각각의 무기 체계들에서 획득한 정보를 공유함을 통해 시너지 효과를 내고 있다. 아무리 좋은 무기를 갖고 있어도 전쟁 상황에 대한 정보가 없으면 어느 지점을 타격할지 모르고, 아무리 안 좋은 무기를 가지고 있어도 정보가 있다면 큰 효과를 낼 수 있다. 이와 같이 전쟁이 점점 더 네트워크 기반의 전쟁으로 진화하고 무기 체계들이 더 많고 다양한 정보를 주고

* Corresponding author, E-mail: jinsol.kim@lignex1.com

받으면서 이런 정보들을 보관하고 처리하는 소프트웨어적 부담 또한 많이 증가했다.

이에 따라 무기체계의 소프트웨어도 복잡한 정보처리 과정을 체계적으로 관리해야 하므로, 소프트웨어를 기능별로 모듈화하는 사례가 늘고 있다[1,2]. 소프트웨어 모듈화의 다양한 장점 중 첫 번째로 꼽을 수 있는 것은 우선 업무 분담이 명확해진다는 점이 있다[3,4]. 모듈화하지 않은 단일 형태의 소프트웨어일 경우, 여러 개발자가 하나의 코드를 수정해야 하므로 업무 분담에 지장이 생긴다. 그뿐만 아니라 기존의 개발자가 아닌 새로운 개발자가 업무를 맡게 되면 한 번에 너무 방대한 양의 코드를 분석해야 하므로 개발에 소모되는 시간이 비약적으로 증가할 수 있다. 그리고 기존의 개발자라도 시간이 지나고 코드를 수정할 일이 생겼을 때 내용을 다시 파악해야 하는 가능성이 있기 때문에 유지보수성이 떨어진다.

모듈화의 또 다른 장점은 새로운 무기 체계의 소프트웨어를 개발할 때 코드를 재사용하기 쉬워진다는 점이다[4]. 단일 형태의 소프트웨어를 재사용하려면 코드 내 결합도가 높아서 필요한 부분만 분리하기 쉽지 않고, 오히려 새롭게 코드를 작성하는 게 더 효율적일 때가 많다. 그 반면에 모듈화된 소프트웨어를 사용한다면, 필요한 기능의 모듈만 재활용할 수 있고, 모듈 간 의존성이 명확하게 정리되어 있어 추가적인 구현이나 수정을 최소화할 수 있다.

모듈화에는 장점이 많지만, 단점 또한 존재한다. 소프트웨어의 복잡도 증가, 오버헤드 발생 등 여러 가지 문제점들이 있지만 본 논문에서는 모듈화에 따르는 동기화 문제에 중점을 둔다. 모듈화를 하게 되면 하나의 정보처리 요청에 대해서도 여러 개의 모듈, 그리고 UI 소프트웨어의 경우 여려 화면 영역들이 해당 요청에 대해 반응하고 갱신해야 한다. 하지만 모듈마다 정보처리 요청에 대해서 수행해야 할 작업의 양이 다르고 이에 따라 각 영역이 갱신되는데 걸리는 시간이 다르다. 화면 영역 간 갱신 시간의 차이는 아무리 미세하더라도 사용자에게 불편함을 줄 뿐만 아니라 업무 수행 시간 증가 및 조작실수를 할 확률을 증가시킨다[5,6]. 만약에 실제 전쟁 상황에서 이러한 문제를 가진 무기체계 운용 UI 인터페이스를 사용하다가 잘못된 판단을 하거나 조작 실수로 인해 의도하지 않은 명령을 내리게 된다면 낭비된 무기 자산에서 오는 금전적인 손실, 실제 위협을 제거하는 데 사용될 무기가 줄어드는 데서 오는 전략적 불리함, 그리고 최종적으로 이러한 실수들로 인해 인명 피해까지 생길 수 있다.

기존 연구에서는 시각적 동기화를 위해 두 가지 방법을 사용한다[7]. 하나는 흐름 동기화로서 단순한 순차적인 작업을 통해 인터페이스를 갱신한다. 사용자가 정보를 입력하거나 혹은 외부에서 정보를 갱신했을 때 해당 정보를 읽어오는 함수가 있고, 함수에서는 특정 인터페이스 부분을 갱신한다. 이 방식은 정보 처리 함수에서 정보를 저장하고 있는 도메인도 갱신하고 관련된 인터페이스를 동시에 갱신해서 코드가 단순하고 이해하기 쉽다는 장점이 있다. 두 번째 방법은 관찰자 동기화로서 정보 갱신에 대해 알림을 보내고 인터페이스 구성 요소들은 이런 알림에 구독하는 방식을 사용한다. 이 방식은 도메인 관련 처리와 인터페이스 관련 처리를 완전히 분리할 수 있어서 여러 인터페이스 구성 요소를 동시에 갱신시킬 때 유용하다. 하지만 흐름 동기화는 도메인 관련 처리와 인터페이스 갱신이 서로 얽혀있어서 복잡한 프로그램일수록 코드의 유지보수성이 떨어지고, 관찰자 동기화는 정보 처리를 하는 모듈이 여러 개일 때를 고려하지 않아서 인터페이스 구성요소 간 갱신 시각의 차이가 생길 수 있다.

이에 따라 본 논문에서는 비동기 함수를 활용해서 흐름 동기화와 관찰자 동기화의 장점을 합친 새로운 동기화 기법을 제시한다. 이 기법은 흐름 동기화 기법처럼 특정 정보 갱신에 관련된 모든 인터페이스 구성 요소를 동시에 갱신시켜서 구성 요소 간 갱신 시각 차이를 최소화하면서 관찰자 동기화 기법의 알림과 구독 방식을 유지해서 도메인 처리와 인터페이스 처리를 분리한다. 제시한 새로운 기법을 MVVM (Model - View - ViewModel) 형태의 모듈화된 UI 소프트웨어에 적용해서 시각적 오차가 얼마나 줄었는지 측정하는 실험을 해서 시각적 오차를 78.4 % 줄이는 데 성공했다. 2장에서는 MVVM 패턴의 주요 구성 요소와 개념을 설명하고, 3장에서는 타깃 소프트웨어를 분석해서 시각적 동기화가 안 되는 부분과 그 원인을 찾는다. 4장에서는 타깃 소프트웨어에 새로운 동기화 기법을 구현하는 방법을 설명하고 동기화 효과를 측정하기 위한 설정들을 소개한다. 5장에서는 동기화 기능을 사용해서 구간마다 처리되는 시간 차이를 측정하여 실험 결과를 정리하고, 마지막으로 6장에서는 실험 결과를 바탕으로 결론을 내린다.

2. MVVM 패턴

새로운 동기화 기법을 소개하기에 앞서 타깃 소프트웨어의 구조인 MVVM 패턴을 소개하겠다. MVVM 패턴에는 3개의 주요 구성 요소가 있다: 정보들을 저장하고 비즈니스 로직을 수행하는 모델(model)과, 화면의 디자인적 구조를 정의해놓은 뷰(view)와, 마지막으로 모델과 뷰 사이에서 모델의 정보를 가공해서 뷰가 가져갈 수 있게 준비하거나, 뷰의 입력을 받아 모델에 정보처리 요청을 보내는 뷰모델(viewmodel)이 있다. 각 요소는 앞서 서론에서 소개한 관찰자 동기화 기법을 사용해 동기화한다. 모델은 데이터를 갱신해서 해당 데이터에 구독한 뷰모델들에 알림을 보내고, 알림을 받은 뷰모델은 정보를 모델에서 가져와서 처리하고 비슷하게 뷰에 알림을 보낸다. 사용자가 인터페이스를 사용해 뷰에 입력하게 되면 뷰는 직접적으로 뷰모델의 데이터를 수정하고, 뷰모델은 이에 반응해 모델의 데이터를 수정한다. Fig. 1은 MVVM의 형태를 나타내는 그림이다.

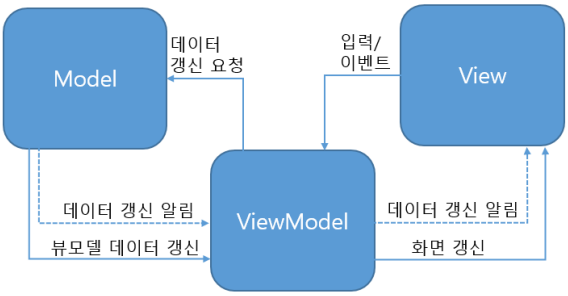

Fig. 1. The MVVM pattern

모델과 뷰모델의 차이를 간단히 표현하자면 모델은 더 큰 단위의 데이터베이스이고 뷰모델은 다른 말로 '뷰의 모델'로서 특정 뷰에서 필요한 데이터만 모델에서 가져오고 가공해서 갖고 있다[7-9]. 하지만 그렇다고 해서 뷰모델과 뷰가 1:1의 관계를 이루고 있는 것은 아니다. 하나의 뷰모델에 대해 여러 개의 뷰가 존재해서 하나의 뷰모델에 대해서 여러 개의 뷰를 사용해 다양한 방식으로 표현할 수가 있는 것이다. 마찬가지로 하나의 모델에 대해 여러 개의 뷰모델이 존재할 수 있다. 예를 들면 동물원 관리 소프트웨어라고 하면 모델에 모든 동물들에 대한 데이터가 담겨있고, 두 개의 뷰모델에는 각각 육식동물하고 초식동물에 대한 데이터를 갖고, 육식동물/초식동물의 위치를 나타내는 그래프의 뷰와 목록을 나타내는 뷰를 구현하면 위 설명대로 모델과 뷰모델의 관계도 1:N이고 뷰모델과 뷰의 관계도 1:N이 된다.

모델은 자신의 데이터를 갱신하고 알림만 보내는 요소로서 뷰모델과 뷰의 존재를 모르고, 오직 정보처리 요청이 들어왔을 때 데이터를 저장 및 수정하고 비즈니스 로직을 수행하기만 한다. 따라서 Fig. 1이 표현하는 것은 모델이 뷰모델에 데이터를 건네주는 것이 아니라 뷰모델이 모델의 데이터를 가져간다고 봐야 한다. 뷰모델은 모델을 관찰하면서 모델의 데이터가 변경됐을 때 모델의 데이터를 가져와서 가공하고 자신의 데이터를 갱신한다. 마지막으로 뷰도 비슷하게 뷰모델을 관찰하면서 뷰모델의 데이터가 변경됐을 때 뷰모델의 데이터를 가져와서 화면의 내용을 갱신한다.

MVVM 패턴은 이렇게 양방향 결합이 아닌 단방향 결합을 사용해 느슨한 결합을 이루고 있다[4,7-9]. 느슨한 결합에는 많은 장점들이 있는데 그중 하나는 관심사를 분리할 수 있어서 디자이너가 뷰를 담당하고 프로그래머가 모델과 뷰모델을 담당해서 흔히 말하는 프론트앤드와 백앤드가 분리된다. 이렇게 명확하게 분리가 된다는 것은 모델과 뷰가 서로 개별적으로 검증이 가능하다는 것이고, 만약 모델이나 뷰에 변경 사항이 생겨도 서로 영향을 주지 않는다. 뷰에서 UI를 완전히 새롭게 디자인한다 해도 모델은 변경할 필요가 없고, 모델에서 데이터를 가공하는 방식을 전부 바꿔도 뷰를 새롭게 만들 필요가 없다.

다음 장에서는 시각적 동기화가 안 되는 부분을 찾기 위해 타깃 소프트웨어를 분석하는 과정에 대해 설명한다.

3. 타깃 소프트웨어 분석

타깃 소프트웨어는 WPF 프레임워크에 C# 언어를 사용하고, 여러 개의 모듈이 존재하며 각 모듈은 MVVM 패턴을 따른다. 따라서 하나의 정보처리 요청에 대해 여러 개의 모델이 비즈니스 로직을 수행하게 되고 이에 대해 여러 뷰모델과 뷰들이 갱신된다. 타깃 소프트웨어를 가지고 실험하기 위해 Fig. 2와 같이 여러 모듈에 영향을 주는 임의의 정보처리 요청을 선택하였다.

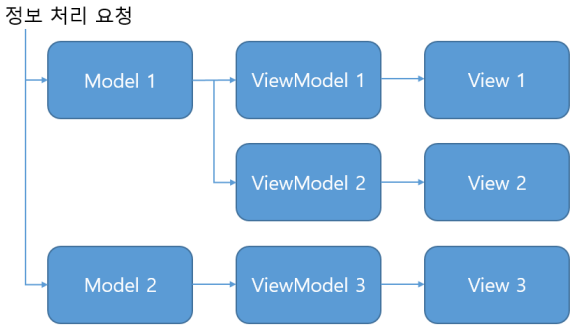

Fig. 2. Event flow structure of data update request

Table 1. Definition of time variables

변수명	의미
t_{m1s}	모델 1이 정보처리 요청을 받은 시각
t_{m1e}	모델 1이 데이터 갱신을 끝낸 시각
t_{m2s}	모델 2가 정보처리 요청을 받은 시각
t_{m2e}	모델 2가 데이터 갱신을 끝낸 시각
t_{vm1s}	뷰모델 1이 갱신 알림을 받은 시각
t_{vm1e}	뷰모델 1이 데이터 갱신을 끝낸 시각
t_{vm1s}	뷰모델 2가 갱신 알림을 받은 시각
t_{vm1e}	뷰모델 2가 데이터 갱신을 끝낸 시각
t_{vm1s}	뷰모델 3이 갱신 알림을 받은 시각
t_{vm1e}	뷰모델 3이 데이터 갱신을 끝낸 시각

선택한 정보처리 요청으로 2개의 모델이 갱신되고 그에 반응해 3개의 뷰모델과 뷰들이 갱신되는데, 각 모델, 뷰모델, 뷰가 알림을 받고 갱신이 완료되는 시각이 다르다. 이를 비교하기 위해 Table 1에 측정할 시각의 변수명을 선언하고 의미를 정의한다.

Table 1을 보면 뷰에서는 시각을 기록하지 않는데 그 이유는 WPF 프레임워크 상 뷰에서 시각을 기록하는 코드를 작성하는 데 어려움이 있기 때문이다. WPF 프레임워크에서 모델과 뷰모델은 앞서 서술한 것처럼 C# 언어를 사용하지만, 뷰는 HTML과 비슷한 XAML이라는 마크업 언어를 사용해서 구현되기 때문에 뷰 내부에서 해당 정보처리 요청에 대한 갱신만 골라서 시각을 기록하는 것은 간단하지 않다. 이에 따라 데이터를 분석하고 시각을 기록하기 용이한 모델과 뷰모델에만 Fig. 3과 같이 코드를 추가했다.

```
Logger.Logger.AddData(ELog.M1S);

public static void AddData(ELog position)
{
    if (!IsRecording) return;
    string time = DateTime.Now.ToString("ffffff");
    string label = position.ToString();
    Debug.WriteLine($"{label}: {time} us");
    Log[position] = time;
}
```

Fig. 3. Implementation of time recording method

Table 2. Relative time chart of each time variable compared to t_{m1s} tested on target software

	M1S	M1E	M2S	M2E	VM1S	VM1E	VM2S	VM2E	VM3S	VM3E
Event 1	0	3.670	11.017	13.383	5.188	6.767	8.268	9.663	14.770	16.075
Event 2	0	1.219	8.029	9.311	2.464	3.762	5.059	6.372	10.575	11.747
Event 3	0	2.173	12.365	13.733	4.379	7.265	9.359	10.825	15.059	16.344
Event 4	0	2.260	15.838	18.403	4.638	7.525	10.258	13.321	20.884	23.225
Event 5	0	1.748	9.096	11.019	3.438	4.911	6.199	7.761	12.643	14.222
Event 6	0	2.615	11.585	12.803	5.306	7.237	9.124	10.348	14.149	15.309
Event 7	0	1.403	7.819	9.238	2.588	3.873	5.117	6.341	10.756	12.312
Event 8	0	1.489	9.379	10.605	4.028	5.442	6.814	8.094	12.038	13.328
Event 9	0	1.429	8.129	9.486	2.887	4.124	5.455	6.868	10.696	12.085
Event 10	0	1.254	8.406	10.006	2.477	3.884	5.305	6.675	11.389	12.819
Event 11	0	1.438	9.713	10.835	3.221	4.890	6.790	8.489	12.063	13.457
Event 12	0	2.840	12.837	13.973	5.521	7.985	10.491	11.651	15.093	16.180
Event 13	0	2.168	11.459	13.068	4.360	6.633	8.875	10.299	14.769	16.010
Event 14	0	2.538	9.767	10.870	5.216	6.406	7.516	8.562	12.146	13.287
Event 15	0	1.541	7.970	9.229	2.745	4.103	5.350	6.719	10.397	11.554
Event 16	0	1.178	7.724	8.980	2.567	3.764	4.978	6.148	10.103	11.303
Event 17	0	2.866	12.836	13.916	5.643	7.971	10.363	11.747	15.085	16.213
Event 18	0	1.504	8.593	9.796	2.856	4.438	6.005	7.451	10.918	12.075
Event 19	0	1.264	7.096	8.156	2.435	3.613	4.744	5.989	9.359	10.501
Event 20	0	3.288	10.776	12.177	5.268	6.833	8.057	9.281	13.381	14.707
Event 21	0	2.404	13.797	15.707	4.386	6.374	9.138	11.712	17.767	20.288
Event 22	0	1.280	7.162	9.053	2.550	3.720	4.850	6.076	10.207	11.399
Event 23	0	2.632	9.910	11.355	4.424	5.941	7.063	8.489	12.546	13.851
Event 24	0	1.721	10.452	12.742	3.499	4.988	6.388	8.364	13.878	14.935
Event 25	0	2.667	11.144	12.288	5.034	7.336	8.877	9.968	13.419	14.582
Event 26	0	2.416	13.456	15.357	4.612	6.864	8.807	11.091	16.512	17.714
Event 27	0	1.308	6.941	8.071	2.474	3.628	4.696	5.817	9.199	10.636
Event 28	0	2.200	12.322	13.749	4.197	6.641	9.092	11.129	14.993	16.374
Event 29	0	1.211	8.055	9.608	2.531	3.985	5.422	6.650	11.047	12.347
Event 30	0	1.209	7.600	9.003	2.345	3.743	4.951	6.327	10.133	11.314
Average	**0**	**1.964**	**10.042**	**11.531**	**3.776**	**5.488**	**7.114**	**8.608**	**12.866**	**14.206**

시각을 기록하는 코드를 추가하고 데이터를 수집하기 위해 앞서 선택한 정보처리 요청을 30번 보내봤다. Table 2는 기록된 시각들을 모두 t_{m1s} 기준으로 지난 시간을 계산한 차트고, Fig. 4는 30번의 정보처리 중 첫 번째 5개의 정보처리 요청에 대해 프로그램이 각 위치를 지나는 시간을 t_{m1s} 기준으로 그린 그래프다. 평균이 아닌 첫 번째 5개를 표현한 이유는 모든 개별적인 정보처리 요청이 똑같은 순서로 시각 기록 위치들을 지나는 것을 중요하게 생각해서이다. 실제로 Table 2를 보면 프로그램 흐름이 각 위치를 지나는 순

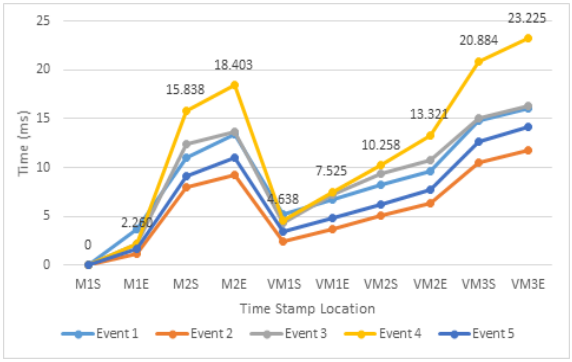

Fig. 4. Relative time graph of each time variable compared to t_{m1s} tested on target software (shows first 5 events out of 30)

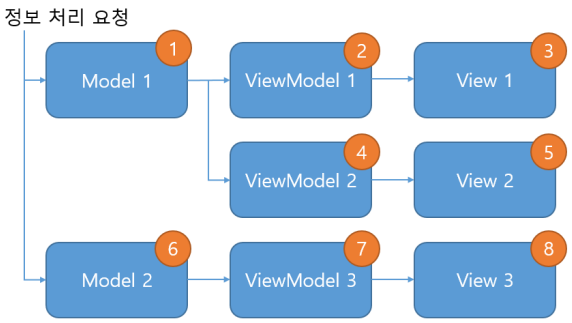

Fig. 5. Order of event flow of selected data update request without synchronization

서가 항상 같은 것을 볼 수 있다. 선택한 정보처리 요청에 대해 모델 1이 가장 먼저 갱신하고, 모델 1을 보고 있는 뷰모델 1과 뷰모델 2가 순차적으로 갱신한다. 그리고 모델 2가 다음으로 정보처리 요청에 대해 갱신하고 모델 2를 보고 있는 뷰모델 3이 마지막으로 갱신된다. Fig. 5에서 이 정보처리가 모듈들의 각 부분을 지나는 순서를 표현했다. 각 뷰가 처리되는 순서를 보면 3, 5, 8번째로서 뷰들이 갱신되는 시각의 차이가 생길 수밖에 없다.

위 현상은 WPF 프레임워크의 설계상 하나의 정보처리 요청(이벤트)에 대한 갱신은 끝까지 하나의 스레드가 담당하게 돼서 그런 것으로 파악된다. 해당 정보처리를 담당하는 스레드는 일단 모델 1에서 정보 갱신을 완료하고, 이에 대한 알림을 보낸다. 모델 1에 구독을 하고 있는 뷰모델 1과 뷰모델 2는 알림을 받고 각각 정보처리를 시작하는데 이 역시 같은 스레드가 순차적으로 진행하게 된다. 하지만 뷰모델 1의 처리가 끝나면 바로 다음에 뷰모델 2의 처리가 시작되는 것이 아니라 뷰 1이 뷰모델 1에 구독하고 있기 때문에 뷰 1의 처리가 먼저 진행된다. 마찬가지로 모델 2는 스레드가 모델 1, 뷰모델 1, 뷰모델 2에 대한 갱신을 하고 나서 정보처리를 하게 된다.

다음 장에서는 타겟 소프트웨어에서 동기화 기능을 만들기 위해 구현한 코드 구조와 기능을 구현한 위치에 대해 설명한다.

4. 동기화 기능 구조

앞서 사용했던 프로그램 흐름도를 원하는 대로 수정할 수 있다면 Fig. 6처럼 수정할 수 있다. 모델과 뷰모델들의 처리 순서가 어떻게 되는지는 상관없고 최종적으로 뷰들이 순서대로 갱신되는 것이 목표라 할 수 있다. 이 목표를 달성하기 위해 선행 연구들[10,11]이 제안한 것처럼 타깃 소프트웨어의 프로그램

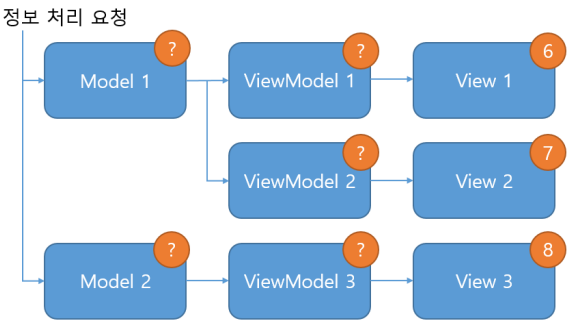

Fig. 6. Ideal order of event flow with synchronization

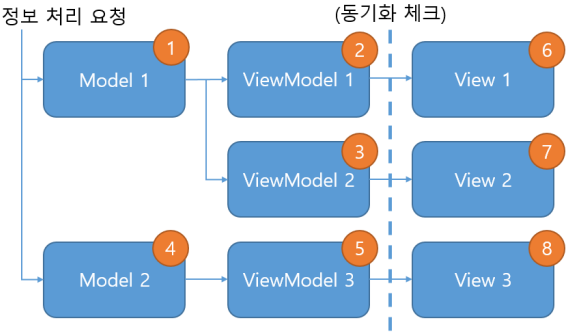

Fig. 7. Order of event flow with synchronization mechanism

```
private async void Update()
{
    if (Logger.Logger.IsActive)
    {
        Logger.Logger.AddData(ELog.VM3S);
        Logger.Logger.IsWaitingVM3.SetResult(true);
        await Logger.Logger.IsWaitingVM1.Task;
        await Logger.Logger.IsWaitingVM2.Task;
    }
    // update logic
    if (Logger.Logger.IsActive)
    {
        Logger.Logger.AddData(ELog.VM3E);
    }
}
```

Fig. 8. Asynchronous method of viewmodel 3

흐름이 특정 위치를 지날 때 잠시 정지했다가 다른 주요 위치들도 지났다는 것을 확인하고 재개하는 방법으로 동기화 기능을 구현하였다. 만약 이를 구현하게 되면 Fig. 7처럼 각 뷰를 갱신하기 직전에 동기화 체크를 해서 모든 뷰모델이 뷰를 갱신시킬 준비가 됐는지 확인하고 나서 최종적으로 뷰들을 순차적으로 갱신시키는 형태가 된다. Fig. 8은 뷰모델 3에서 동기화 기능을 구현한 코드를 보여준다.

먼저 동기화가 필요한 위치에 있는 함수를 비동기 함수로 선언해서 이벤트를 발생시킨 스레드가 아닌 별도의 스레드가 이 함수를 수행하게 만들어야 한다. 그리고 해당 함수 안에서 동기화가 필요한 위치에 도달하면 동기화 모듈에 현재 위치에 도달했다는 것을 알리고, await 키워드를 사용해서 동기화 모듈이 동기화가 필요한 다른 위치들도 도달됐는지 알려주기를 기다린다. 이런 방식으로 구현하면 프로그램 흐름이 동기화 기능을 구현한 모든 위치에 도달해야지만 실행을 계속하게 돼서 프로그램 흐름 순서를 제어할 수 있게 된다.

Table 1에서 정의한 시각 중 각 모델 혹은 뷰모델이 정보처리 요청을 받은 시각을 동기화하거나 데이터 갱신을 끝낸 시각을 동기화할 수 있는데 이 중 뷰모델이 데이터 갱신을 끝내는 지점은 동기화 하면 동기화 효과를 측정할 수가 없다. 왜냐하면 뷰모델이 데이터 갱신을 하면 뷰에 알림을 보내는데 앞서 설명한 대로 뷰 쪽에 시각을 기록하는 코드를 넣기 어려워서 뷰가 화면 갱신을 했다는 것을 알 수 없기 때문이다. 따라서 Table 3과 같이 4가지 상황에서 시간을 측정하였다.

Table 3. Configurations of target software

상황	소프트웨어 구조
C_{none}	동기화 기능이 없는 구조
C_{ms}	모델들이 정보처리 요청을 받는 시간을 동기화함
C_{me}	모델들이 데이터 갱신을 끝낸 시간을 동기화함
C_{vms}	뷰모델들이 정보처리 요청을 받는 시간을 동기화함

다음 장에서는 앞서 3장에서 한 것처럼 상황마다 정보 갱신 요청을 30번 해서 프로그램 흐름을 기록하는 실험을 한다. 이 결과를 가지고 어떤 소프트웨어 구조가 동기화 효과가 제일 크고 그 원인에 대해 알아본다.

5. 실험 결과

Table 4. Relative time chart of each time variable compared to t_{m1s} for C_{ms}

	M1S	M1E	M2S	M2E	VM1S	VM1E	VM2S	VM2E	VM3S	VM3E
Event 1	0	10.060	6.719	19.211	11.433	12.992	14.586	15.980	20.538	21.870
Event 2	0	2.587	1.455	9.174	3.831	5.060	6.268	7.406	10.673	12.140
Event 3	0	2.662	1.402	9.265	3.912	5.813	6.930	8.100	10.368	11.596
Event 4	0	3.055	1.570	10.435	5.183	6.427	7.990	9.147	11.831	13.440
Event 5	0	3.119	1.787	13.304	5.339	8.224	10.655	11.937	14.620	16.370
Event 6	0	3.652	1.989	13.167	5.691	8.011	10.010	12.051	14.355	15.528
Event 7	0	6.077	2.861	12.279	7.264	8.430	9.614	10.824	13.572	14.813
Event 8	0	2.600	1.393	10.634	4.769	6.261	7.921	9.226	11.840	13.113
Event 9	0	2.622	1.360	9.247	3.833	5.619	6.685	7.957	10.655	11.919
Event 10	0	4.362	2.687	10.608	5.634	6.763	7.997	9.166	11.822	13.029
Event 11	0	4.258	2.152	10.587	5.540	6.897	8.199	9.341	11.773	12.983
Event 12	0	2.890	1.475	10.577	4.615	6.895	8.137	9.334	11.780	13.015
Event 13	0	5.218	2.463	13.418	7.497	9.273	10.724	12.068	14.574	15.850
Event 14	0	5.527	3.158	13.186	8.006	9.064	10.424	11.619	14.478	15.790
Event 15	0	2.610	1.449	8.899	3.755	5.240	6.593	7.696	10.035	11.305
Event 16	0	2.609	1.345	11.257	3.732	5.501	7.496	10.063	12.479	13.603
Event 17	0	2.617	1.400	9.390	3.873	5.519	6.926	8.139	10.532	11.674
Event 18	0	2.637	1.495	9.234	3.749	4.970	6.226	7.940	10.917	12.014
Event 19	0	4.302	1.467	12.217	7.080	8.356	9.607	10.873	13.446	14.748
Event 20	0	5.193	2.531	12.215	7.288	8.640	9.908	11.048	13.372	14.627
Event 21	0	2.667	1.443	12.453	4.117	6.499	8.439	11.032	13.601	14.943
Event 22	0	5.117	2.745	14.487	7.526	10.175	11.951	13.253	15.593	16.808
Event 23	0	3.201	1.775	10.148	5.524	6.676	7.924	9.056	11.271	12.532
Event 24	0	4.238	2.481	10.570	5.637	6.889	8.099	9.315	11.842	13.097
Event 25	0	2.572	1.411	8.532	3.794	4.922	6.116	7.286	9.770	11.057
Event 26	0	5.093	2.692	14.300	7.535	9.930	11.894	13.068	15.450	16.649
Event 27	0	5.182	2.654	14.725	7.557	9.919	12.387	13.517	15.862	17.062
Event 28	0	4.298	1.734	13.941	7.641	9.483	10.947	12.384	15.398	16.934
Event 29	0	4.609	2.800	10.738	5.857	7.086	8.303	9.505	11.910	13.005
Event 30	0	5.395	2.771	12.943	7.910	9.385	10.559	11.797	14.319	15.724
Average	0	**4.034**	**2.155**	**11.705**	**5.837**	**7.497**	**8.984**	**10.338**	**12.956**	**14.241**

Table 5. Relative time chart of each time variable compared to t_{m1s} for C_{me}

	M1S	M1E	M2S	M2E	VM1S	VM1E	VM2S	VM2E	VM3S	VM3E
Event 1	0	6.653	9.778	12.429	13.996	15.187	16.339	17.496	18.828	20.000
Event 2	0	2.561	5.392	7.943	9.936	11.430	12.788	14.257	15.440	16.676
Event 3	0	1.301	2.635	3.846	5.063	6.166	7.445	8.583	9.766	10.890
Event 4	0	1.294	3.145	5.128	6.927	14.408	16.032	17.455	18.831	20.145
Event 5	0	1.334	2.543	3.820	4.968	6.481	7.621	8.859	10.047	11.298
Event 6	0	1.567	2.899	4.166	5.272	6.643	7.932	9.309	10.642	11.802
Event 7	0	1.254	2.765	3.861	5.072	6.269	7.437	8.503	9.686	10.799
Event 8	0	1.546	3.181	4.414	5.664	6.832	8.097	9.230	10.511	11.690
Event 9	0	1.236	3.152	4.511	5.985	7.112	8.306	9.546	10.800	12.254
Event 10	0	1.430	3.948	6.978	8.846	10.161	11.414	12.895	14.241	15.387
Event 11	0	2.726	5.914	8.189	9.844	11.611	13.055	14.434	15.720	17.098
Event 12	0	1.270	2.622	3.770	5.081	6.268	8.639	10.669	11.947	13.104
Event 13	0	2.351	4.615	7.440	10.479	12.075	13.260	14.675	16.055	17.613
Event 14	0	2.459	5.061	7.423	10.067	12.541	14.011	15.102	16.195	17.475
Event 15	0	1.251	3.621	5.164	6.869	8.410	9.717	10.906	12.145	13.287
Event 16	0	1.253	2.562	3.844	5.029	6.478	7.680	8.888	10.111	11.325
Event 17	0	2.332	4.737	6.616	7.719	9.018	10.131	11.324	12.638	13.916
Event 18	0	1.124	2.513	4.022	5.534	6.665	7.909	9.022	10.192	11.330
Event 19	0	1.412	2.715	3.878	4.984	6.348	7.518	8.704	9.881	11.125
Event 20	0	3.309	5.722	7.735	9.886	11.527	12.672	13.902	15.091	16.296
Event 21	0	2.612	5.138	7.450	9.915	11.622	12.686	13.934	15.120	16.501
Event 22	0	2.972	5.636	7.155	8.499	9.893	10.943	12.208	13.533	15.003
Event 23	0	1.484	2.926	4.286	6.071	7.412	9.974	12.764	15.288	16.978
Event 24	0	1.212	2.659	4.042	5.406	6.609	7.857	9.147	10.367	11.554
Event 25	0	2.207	4.373	6.603	8.648	10.329	11.754	13.894	15.967	19.026
Event 26	0	1.971	3.677	5.738	8.239	9.596	10.708	11.997	13.076	14.327
Event 27	0	1.328	2.810	3.956	5.426	6.833	8.172	9.517	10.855	12.023
Event 28	0	1.456	3.029	4.359	5.526	6.756	7.994	9.278	10.842	12.381
Event 29	0	1.259	2.732	3.900	5.838	8.418	11.170	12.595	13.863	15.066
Event 30	0	1.456	2.964	4.303	5.910	8.226	9.886	11.007	12.418	13.638
Average	0	**1.921**	**3.849**	**5.566**	**7.223**	**8.911**	**10.305**	**11.670**	**13.003**	**14.334**

Table 6. Relative time chart of each time variable compared to t_{m1s} for C_{vms}

	M1S	M1E	M2S	M2E	VM1S	VM1E	VM2S	VM2E	VM3S	VM3E
Event 1	0	3.259	7.523	10.246	4.545	13.014	5.718	13.022	11.664	16.290
Event 2	0	1.280	5.389	6.542	2.527	9.725	4.002	9.738	8.303	12.119
Event 3	0	1.188	5.990	7.149	2.661	9.639	4.771	9.640	8.416	11.200
Event 4	0	1.426	6.009	7.893	2.827	12.696	4.023	12.716	10.245	15.743
Event 5	0	1.635	7.355	8.566	3.290	11.055	5.164	11.062	9.814	12.910
Event 6	0	2.612	7.079	8.357	4.509	10.904	5.765	10.911	9.613	12.614
Event 7	0	1.218	5.193	6.263	2.509	8.599	3.811	8.609	7.500	10.396
Event 8	0	1.778	6.673	7.956	3.523	10.856	5.359	10.862	9.571	12.563
Event 9	0	1.241	8.181	9.286	3.349	11.718	6.520	11.718	10.443	14.280
Event 10	0	1.088	5.569	7.452	2.380	9.893	3.563	9.902	8.632	11.828
Event 11	0	1.463	7.507	8.712	3.579	10.881	6.107	10.881	9.818	13.196
Event 12	0	1.165	5.530	6.588	2.698	8.906	4.072	8.912	7.793	10.806
Event 13	0	2.485	10.190	12.198	5.375	14.549	7.724	14.554	13.330	16.346
Event 14	0	1.338	5.134	6.392	2.481	8.738	3.862	8.757	7.526	10.529
Event 15	0	1.257	5.264	6.491	2.597	9.066	4.089	9.074	7.802	10.673
Event 16	0	1.141	5.215	6.290	2.531	8.606	3.853	8.611	7.502	9.817
Event 17	0	1.332	5.048	6.213	2.619	8.731	3.887	8.738	7.396	10.447
Event 18	0	1.419	5.716	7.178	2.741	9.742	4.193	9.742	8.416	11.018
Event 19	0	1.345	5.790	7.736	2.567	10.305	4.597	10.312	9.124	11.859
Event 20	0	1.468	5.087	6.334	2.645	8.713	3.935	8.719	7.426	10.701
Event 21	0	1.448	5.872	7.281	2.828	10.009	4.471	10.015	8.658	11.960
Event 22	0	1.509	7.831	9.275	2.711	11.937	5.228	11.939	10.610	13.643
Event 23	0	1.241	4.699	5.904	2.355	8.190	3.526	8.197	7.000	9.757
Event 24	0	2.697	7.740	8.948	5.012	11.674	6.486	11.680	10.188	13.740
Event 25	0	1.133	5.031	6.235	2.442	8.793	3.689	8.800	7.514	10.671
Event 26	0	1.304	7.032	8.337	3.395	10.681	5.504	10.687	9.504	12.423
Event 27	0	1.562	6.433	7.580	2.852	10.099	5.234	10.169	8.806	11.903
Event 28	0	2.294	8.656	9.854	4.770	12.213	7.301	12.219	11.107	13.432
Event 29	0	1.976	7.452	9.747	4.374	12.577	5.805	12.583	11.386	14.308
Event 30	0	1.606	7.216	8.500	4.176	10.986	5.906	10.992	9.721	12.694
Average	0	**1.597**	**6.447**	**7.850**	**3.229**	**10.450**	**4.939**	**10.459**	**9.161**	**12.329**

C_{none}은 동기화 기능이 없는 구조로서 앞서 이미 데이터를 모았으니 추가적으로 C_{ms}, C_{me}, C_{vms}를 구현해서 데이터를 기록했다.

Table 4, Table 5, Table 6은 각각 C_{ms}, C_{me}, C_{vms}를 구현해서 t_{m1s}를 기준으로 각 기록 위치를 지난 시간을 계산한 차트고 Fig. 10, Fig. 11, Fig. 12도 비슷하게 첫 5개의 정보처리가 t_{m1s}를 기준으로 각 기록 위치를 지난 시간을 표현한 그래프다. 이 데이터를 가지고 동기화의 효과를 계산하기 위해 몇 가지 공식을 다음과 같이 정의한다.

$$T_{ms} = \max(t_{m1s}, t_{m2s}) - \min(t_{m1s}, t_{m2s}) \quad (1)$$

먼저 (1)은 프로그램이 모델 1이 정보 갱신을 받은 시각과 모델 2가 정보 갱신을 받은 시각 중 먼저 지나간 시각에서 나중에 지나간 시각을 빼서 모델 간 정보 갱신 요청을 받은 시각의 최대 차이를 구하는 것이다.

$$T_{me} = \max(t_{m1e}, t_{m2e}) - \min(t_{m1e}, t_{m2e}) \quad (2)$$

마찬가지로 (2)는 프로그램이 모델 1이 정보 갱신을 마친 시각과 모델 2가 정보 갱신을 마친 시각 중 먼저 지나간 시각에서 나중에 지나간 시각을 빼서 모델 간 정보 갱신을 마친 시각의 최대 차이를 구하는 것이다. (1), (2)를 계산하는 이유는 이 두 값을 가지고 모델 쪽에서 프로그램이 동기화가 얼마나 됐는지 측정하기 위함이다.

$$T_{vms} = \max(t_{vm1s}, t_{vm2s}, t_{vm3s}) - \min(t_{vm1s}, t_{vm2s}, t_{vm3s}) \quad (3)$$

다음으로 (3)은 프로그램이 뷰모델 1, 뷰모델 2, 뷰모델 3 중 가장 먼저 정보 갱신 알림을 받은 시각과 가장 늦게 정보 갱신 알림을 받은 시각의 차이를 구하는 공식이다.

$$T_{vme} = \max(t_{vm1e}, t_{vm2e}, t_{vm3e}) - \min(t_{vm1e}, t_{vm2e}, t_{vm3e}) \quad (4)$$

마지막으로 (4)는 프로그램이 뷰모델 1, 뷰모델 2, 뷰모델 3 중 가장 먼저 정보 갱신을 마친 시각과 가장 늦게 정보 갱신을 마친 시각의 차이를 구하는 공식이다. (3), (4)를 계산하는 이유는 이 두 값을 가지고 뷰모델 쪽에서 프로그램이 동기화가 얼마나 됐는지 측정하기 위함이다.

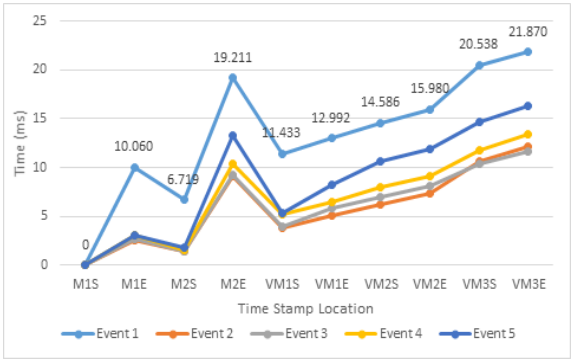

Fig. 9. Relative time graph of each time variable compared to t_{m1s} for C_{ms}(shows first 5 events out of 30)

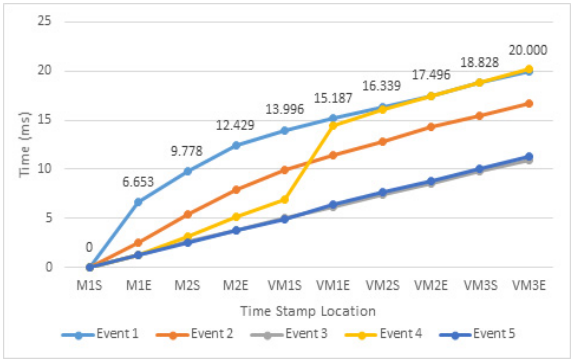

Fig. 10. Relative time graph of each time variable compared to t_{m1s} for C_{me}(shows first 5 events out of 30)

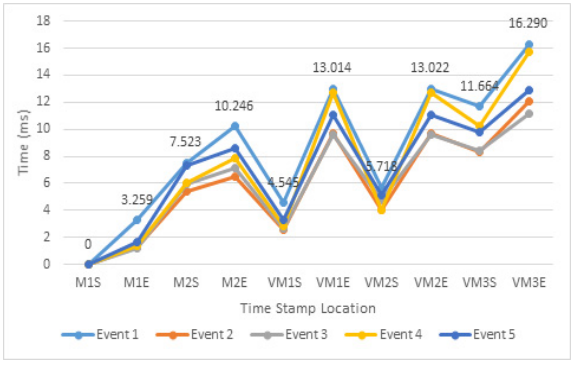

Fig. 11. Relative time graph of each time variable compared to t_{m1s} for C_{vms}(shows first 5 events out of 30)

Table 7. Average time between components of target software for each configuration

상황	T_{ms} (1)	T_{me} (2)	T_{vms} (3)	T_{vme} (4)
C_{none}	10.042 ms	9.566 ms	9.090 ms	8.718 ms
C_{ms}	2.155 ms	7.670 ms	7.118 ms	6.744 ms
C_{me}	3.849 ms	3.645 ms	5.780 ms	5.423 ms
C_{vms}	6.447 ms	6.253 ms	5.932 ms	1.879 ms

Table 7은 앞서 모은 데이터를 가지고 (1) ~ (4) 공식대로 시간을 계산한 수치들의 평균을 나열한 표다. 먼저 C_{none}을 보면 다른 상황들에 비해 (1) ~ (4) 모두 비슷한 수치가 나왔다. 이는 앞서 Fig. 5에서 보인 것처럼 프로그램 흐름이 모델과 뷰모델을 순차적으로 거쳐 가는데 모델 1에서 데이터 처리하는 데 걸리는 시간이 모델 2에서 데이터 처리하는 데 걸리는 시간과 비슷해서 T_{ms}와 T_{me}가 크게 차이 나지 않고, 마찬가지로 뷰모델 1에서 데이터 처리하는 데 걸리는 시간이 뷰모델 2 혹은 뷰모델 3에서 데이터 처리하는 데 걸리는 시간과 비슷해서 T_{vms}와 T_{vme}가 크게 차이 나지 않는다.

추가로 뷰모델에 동기화 기능을 구현하지 않은 C_{none}, C_{ms}, C_{vms}를 보면 모두 T_{vme}가 T_{vms}보다 낮은 것을 볼 수가 있다. 이는 뷰모델 1에서 데이터 처리하는 데 걸리는 시간이 뷰모델 3에서 데이터 처리하는 시간보다 더 길다고 볼 수 있는데 실제로 뷰모델 내부 처리 과정을 살펴보면 일관성이 있다고 볼 수 있다. 뷰모델 1에서는 데이터 갱신 과정에서 11개의 속성을 갱신시키지만, 뷰모델 3에서는 데이터 갱신 과정에서 하나의 속성만 갱신시켜서 데이터를 처리하는 데 시간이 더 걸리는 걸로 파악된다.

다음으로 C_{ms}를 보면 C_{none}과 비교해서 T_{ms}가 78.5 % 낮아진 것을 볼 수가 있다. 이는 모든 모델이 정보 처리 요청을 받을 때까지 모델의 데이터 갱신을 못 하게 동기화 기능을 넣은 것이 의도대로 작동한 것으로 보인다. 동기화 기능이 없던 C_{none}에서는 모델 1에서 데이터 갱신 알림을 받는 뷰모델 1과 뷰모델 2가 모델 2가 정보 처리 요청을 받기 전에 데이터 갱신 과정을 거치게 되는데 C_{ms}에서는 모델 1이 정보 처리 요청을 받으면 코드 내 await 키워드에서 프로그램 흐름을 멈추고 모델 2에 정보 처리 요청을 보내게 된다. 모델 2도 정보처리 요청을 받은 위치에 도달하면 모

델 1에 Task가 완료됐다는 알림을 보내서 모델 1에서 프로그램 흐름이 재개되게 되므로 뷰모델 쪽으로 순서가 넘어가기 전에 모델의 처리가 모두 시작되고 그로 인해 T_{ms} 수치가 낮게 나온 것이다.

하지만 모델 쪽 처리의 시작 시각을 동기화했으니 T_{me} 수치도 낮게 나올 거로 추측 했지만 예상과는 다르게 T_{me} 수치는 C_{none}과 비교해서 19.8 % 낮은 값으로 C_{ms}의 감소보다는 훨씬 적게 효과를 봤고, T_{vms}와 T_{vme}도 각각 21.7 %, 22.6 % 감소했다. 원인은 Fig. 9를 보면 알 수 있는데, 동기화 기능을 설정한 위치까지는 동기화 효과를 봤지만, 그 이후로는 여전히 프로그램 흐름이 순차적으로 진행돼서 T_{me} 뿐만 아니라 T_{vms}와 T_{vme}에서도 큰 효과를 보지 못한 걸로 파악이 된다. 모델 2가 정보처리 요청을 받으면 모델 1의 await이 바로 풀리고 프로그램 흐름이 모델 1쪽으로 넘어가서 그다음 순서인 t_{m1e}, t_{vm1s} ~ t_{vm2e}, t_{2me} 그리고 마지막으로 t_{vm3s}와 t_{vm3e}를 통과한 것으로 보인다. 이에 따라 T_{me}, T_{vms}, T_{vme}는 여전히 높은 수치를 보인 것이고, 일관적으로 비슷한 감소율을 보인 것은 C_{none}과 비교했을 때 프로그램이 t_{vm2e}와 t_{vm3s} 사이에 t_{m2s}, t_{m2e} 두 위치를 거치는 것이 아닌 t_{m2e} 만 거치기 때문에 그만큼 일관적으로 시간 차이가 감소한 것으로 보인다.

다음으로 C_{me}를 보면 C_{none}과 비교해서 T_{ms}가 61.7 % 낮아졌고, T_{me}가 61.9 % 낮아진 것을 볼 수가 있다. C_{ms}때와 비슷하게 프로그램 흐름이 모델에서 뷰모델로 넘어가기 전에 동기화 기능이 작동해서 C_{none}과 비교했을 때 T_{ms}와 T_{me}는 많이 감소하고 T_{vms}와 T_{vme}는 비교적 적게 감소했다. T_{vme}와 Tvme는 각각 36.4 %, 37.8 % 감소했는데 이는 프로그램이 t_{vm2e}와 t_{vm3s} 사이에 t_{m2s}, t_{m2e} 두 위치를 거치지 않고 바로 순차적으로 진행되기 때문에 그만큼 일관적으로 시간 차이가 감소한 것으로 보인다.

마지막으로 C_{vms}를 보면 C_{none}과 비교해서 T_{ms}, T_{me}, T_{vms}가 각각 35.8 %, 34.6 %, 34.7 % 낮아졌고, T_{vme}가 많이 감소해서 78.4 % 낮아진 것을 볼 수가 있다. C_{ms}, C_{me}와 비교해서 T_{ms}, T_{me}가 오히려 더 적게 감소한 것을 볼 수가 있는데 Fig. 11을 보면 원인을 알 수 있다.

C_{ms}, C_{me}와 비슷하게 프로그램의 흐름이 t_{m1s}, t_{m1e}부터 순차적으로 통과하지만, 다음에 t_{m2s}로 가는 것이 아니라 t_{vm1s}, 그리고 t_{vm2s}를 통과하는 것을 볼 수 있다. 이는 동기화 기능이 모델에서 정보 처리 요청을 받는 시간이나 갱신 완료 시간을 서로 동기화하는 것이 아니라 뷰모델 쪽에서 데이터 갱신 알림을 받을 때를 동기화해서 그런 것이다. 그래서 모델 1에서는 await 키워드를 만날 일 없이 바로 뷰모델 1, 그리고 뷰모델 2에 데이터 갱신 알림을 보내는 것이고, 각 뷰모델에서는 데이터 갱신 알림을 받자마자 await 키워드에서 멈추고 다음 뷰모델로 프로그램 흐름을 보내는 것이다. 뷰모델 2가 await 키워드에 도달하면 그다음 순서는 모델 2로 가서 데이터를 갱신하고 뷰모델 3으로 알림을 보내고, 뷰모델 3도 await 키워드에 도달하면 다시 뷰모델 1부터 순차적으로 await 단계를 넘어가 데이터 갱신 과정을 거치게 되는 것이다. 따라서 T_{ms}, T_{me}, T_{vms}는 모델 쪽의 동기화가 없었기 때문에 C_{ms}, C_{me} 보다 감소량이 적었던 것이고 반대로 T_{vme}는 크게 효과를 본 것이다. 다르게 표현하면 모델 쪽 처리 과정을 모두 마쳤기 때문에 뷰모델 데이터 갱신을 하는 동안 다른 작업을 할 필요가 없어서 T_{vme} 값이 많이 감소할 수 있던 것이다.

만약 동기화 기능을 모델과 뷰모델 두 군데에 넣으면 (C_{me}와 C_{vms}를 동시에 적용하면) T_{ms}, T_{me}, T_{vms}, T_{vme} 모두 감소시킬 수 있겠지만 본 논문의 목적상 그럴 필요는 없어 보인다. 최종적으로 각 화면 영역이 갱신되는 시간 차이를 줄이는 게 목표이기 때문에 중간에 어떤 순서로 프로그램이 실행되어도 현재 관점에서는 문제가 없다. 오히려 너무 많은 위치에 동기화 기능을 구현 해놓으면 프로그램 흐름이 자주 옮겨 다녀서 특정 정보처리 요청에 대해 최종적으로 모든 화면 영역이 갱신되는 시간까지 더 오래 걸릴 수 있다. 그러므로 본 논문에서 측정한 T_{vme} 값이 화면 영역 간 갱신 시간 차이를 측정하는 가장 중요한 값이라고 볼 수 있고, 이 값을 78.4 % 감소시켜서 모든 소프트웨어 구조 중 가장 크게 낮춘 C_{vms}가 제일 적합하다는 것을 확인하였다.

Table 8. Percentage reduction in T_{vme} for each configuration compared to C_{none}

상황	T_{vme}	% Reduction
C_{none}	8.718 ms	0 % (baseline)
C_{ms}	6.744 ms	22.6 %
C_{me}	5.423 ms	37.8 %
C_{vms}	1.879 ms	78.4 %

하지만 본 논문의 한계점은 앞서 언급한 것처럼 타깃 소프트웨어의 구조상 뷰 쪽에서 동기화 기능을 위한 코드를 작성하기 어려워서 C_{vme}는 측정하지 않은 것이다. 만약 뷰 쪽에도 시간을 측정할 수 있다면 화면 영역 간 갱신 시간 차이를 줄이는 데는 C_{vme}가 더 효과적일 수 있고, 나아가서 비동기 함수와 await 키워드 자체를 뷰 쪽에서 구현이 가능하면 더욱 큰 효과를 볼 수도 있을 것으로 생각된다.

6. 결 론

본 논문에서는 많고 다양한 정보를 처리해야 되는 무기체계 UI 소프트웨어의 시각적 동기화 문제를 해결하기 위해 비동기 함수를 사용한 동기화 기능을 구현해서 화면 영역 간 갱신되는 시간 차이를 최대 78.4 % 까지 줄였다. 이 방법은 모듈화된 MVVM 소프트웨어의 시각적 동기화 문제를 해결하는 새로운 기법으로서, 흐름 동기화 기법처럼 정보 처리 별 인터페이스 갱신을 통해 갱신 시각 차이를 최소화하면서, 동시에 관찰자 동기화 기법의 알림 구독 시스템을 사용해서 코드 유지보수성도 좋게 한다. 본 논문에서는 MVVM 형태로 된 타깃 소프트웨어의 구조상 뷰 쪽에서의 동기화까지 구현하진 못했지만 추후 연구에서는 뷰의 프로그램 흐름도 제어해서 더 효과적인 동기화 효과를 구현하고자 한다.

References

[1] Y. An, H. Kim, C. Lee, H. Na, “Development of Modular Simulation Program for Guided Weapon,” KIMST Annual Conference Proceedings, pp. 1529-1530, 2022.

[2] D. Wi, Y. You, Y. Kim, “Software Modularization Architecture for multiple types of missiles control,” KIMST Annual Conference Proceedings, pp. 1261-1262, 2020.

[3] J. Lee, U. Park, Y. Park, et. al., “Modularizing Software using Direct and Indirect Class Coupling Metrics,” Journal of KISS: Software and Applications, pp. 327-339, 2014.

[4] D. Park, Y. Seo, “A GUI-based Approach to Software Modularization,” Journal of the Korea Society of Computer and Information, Vol. 23, No. 4, pp. 97-106, 2018.

[5] T. W. Simpson, K. Barron, L. Rothrock, et. al., “Impact of response delay and training on user performance with text-based and graphical user interfaces for engineering design,” Research in Engineering Design, Vol. 18, pp. 49-65, 2007.

[6] L. Ming, A. Katrin, V. Luisa, et. al., “Influence of temporal delay and display update rate in an augmented reality application scenario,” Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia, pp. 278-286, 2015.

[7] A. Syromiatnikov and D. Weyns, “A Journey Through the Land of Model-View-Design Patterns,” 2014 IEEE/IFIP Conference on Software Architecture, pp. 21-30, 2014.

[8] J. Kouraklis, “MVVM in Delphi,” Apress Berkely, CA, pp. 1-12, 2016.

[9] C. Anderson, “The Model-View-ViewModel(MVVM) Design Pattern,” Apress Berkeley, CA, pp. 461-499, 2012.

[10] J. Kim, N. Shin, Y. Lee, “Resolving the Visual Synchronization Issue of Modular UI Software Made with MVVM Pattern,” KIMST Annual Conference Proceedings, pp. 617-618, 2023.

[11] B. Marco, F Sebastian and A. Sahin, “Event-based Synchronization of Model-Based Multimodal User Interfaces,” Proceedings of Second International Workshop on Model Driven Development of Advanced User Interfaces, 2006.