Towards a common platform simulator for European armored combat vehicles using a modular software architecture

***Abstract*—In this paper we present a novel approach for building a common software platform for simulating armored combat vehicles. We use Unreal Engine 4 as our simulation software. The presented approach is an attempt towards integrating different combat vehicle modules into a simulated environment. The presented simulator architecture can be used in various training combat scenarios, such as reconnaissance, coordinated firing on targets, different cooperation scenarios, etc.**

Keywords—training simulator, game engines, armored vehicles, serious games

I. Introduction

Training simulators are a vital part of modern warfare. They are an effective means for preparing young cadets for the realities of the battlefield. The virtual experience those simulators provide are comparable to those of the real world. Training simulators are a part of a larger family, called serious games. Those games are intended and focused primarily on education and skill building rather than entertainment [1]. The military are the first institutions to adopt the strategy for preparing their soldiers using simulated experiences [2], [3]. There are good reasons for using simulators for military training: they save human life and reduce training costs. Teaching young soldiers on mocked-up models of military machines – airplanes, battle tanks, armored vehicles, etc., is safer than the alternative - to let them deal with the real equipment on their first day. As we well know, flying simulators are a mandatory step in all cadet training programs. That virtual experience will later on make the future pilot more confident and less stressed when entering and piloting the real airplane. In addition, there are several scenarios that are impossible or very dangerous to be performed in a real airplane or helicopters, such as engine failures, being shot at, autorotation emergency landings, landing on a frozen surface, etc.

The other advantage of using simulators is that their production and maintenance costs are far less than those of their real-world counterparts. For instance, the production, operational and maintenance costs of a fighter jet are far greater [4] than those for a simulator [5]. That statement holds true for armored ground vehicles – such as main battle tanks (MBT) and light-armored vehicles (LAV). There are currently efforts to research and create a common European vehicle combat platform (Figure 1), as part of the PESCO (Permanent Structured Cooperation) project [6] of the European Council. The PESCO project will "*assure the combat vehicles would be based on a common platform and would support fast deployment maneuver, reconnaissance, combat support, logistics support, command and control, and medical support*". There are additional initiatives in that direction, particularly the European Defence Industrial Development Programme (EDIDP) by the European Defense Agency (EDA). The common platform will strengthen the Common Security and Defence Policy (CSPD) ensuring, at the same time, the interoperability among European armies. The same statement holds true for the simulators of this common vehicle platform – a vital and important part of this endeavor.

Figure 1: A concept main ground combat system. [7]

In this paper we propose an approach for building the software counter-part of the common vehicle platform. We will present and discuss an approach that uses novel distributed software architecture for building serious games [8]. That architecture allows us to use separate modules to represent different functional units. We discuss the essential in our opinion modules that need to be developed and integrated in such a simulation platform. We also choose and motivate the use of game engines as an effective means for such development.

II. Previous work

There are numerous examples of using simulators for combat training and study scenarios. One of the first combat training serious games for the US military is America's Army [9] and was used for recruiting soldiers for the US army. Another such example is the simulator TC3Sim [10]. In it, soldiers are trained how to provide emergency aid on the battlefield while under fire. This simulator trains physical skills, as well as psychological response in the cadets. The game is developed with the help of the game engine Unity 3D and is available to play via a web browser.

Other work in this field focuses on feasibility studies of possible real-world scenarios and the evaluation of new machinery. For instance, McHugh et al. studied the feasibility of slowing aerial descent of the M1 by utilizing the energy impulse, generated by its main cannon [11]. This is an important study, since the conditions described in the paper are hard to set-up in the real world. Other scientists focus on studying the effects and projectile paths of shells. Such an example is the work of Magier and Merda [12] who explore how the projectile velocities of battle tank and mortar shells change with regards to air drag. For their study, the researchers use numerical simulations with a custom-built analytics software in order to create a computational fluid dynamics mathematical model. Their findings help test and evaluate the projectile characteristics of newly developed mortar rounds. This study saves production resources and time for building real-world models and performing the experiments and measurements. Other authors have shown how to easily build an M1A2 main battle tank simulator while keeping the development costs low [13]. Their research helps evaluate external projectile ballistics and generate various scenarios for troop's exercises.

But why is the use game engines so widely spread among scientists and what are the benefits for the development and simulation process? First, game engines provide an easy to use framework that already has built-in, simulation-ready subsystems, such as physics engine, sound engine, artificial intelligence, 3D model interaction and visualization. One such example of using physics model and Unreal Engine 4 is the work of [14], who simulate the movement of underwater cables, attached to remotely operated vehicles. The benefits in this case are the out-of-the-box physics simulation model of the cable that would otherwise take a considerable amount of time to be implemented and visualized. Another example for using game engines is the game America's Army [9], which also employs Unreal Engine as its physics and rendering environment. On the other hand, SanTrain [15] and TC3Sim [10] both use Unity 3D – another popular game engine these days. There are, however, still scientists who prefer to build custom simulation solutions for their research.

III. Simulator requirements and scenarios

Our approach includes the following phases. First, we describe an overview of the whole software platform and analyze the specific system requirements. After that, we choose a software architecture that will best fit those requirements. Then, we describe the selected modules that will be created – weapon systems, drivetrain / hull, turret station, sensors, etc. Then we describe each module individually and how it would be integrated into the whole system simulator.

Let's start by explaining what a common base vehicle platform is. When there is a joint cross-country military operation, each country provides its own military equipment. In the case of armored ground forces, Germany will participate with the Leopard 2 – its main battle tank, while France will deploy its Leclerc. Other countries, such as those from Eastern Europe, will deploy old Russian tanks, mainly T-72 models (Czech Republic, Bulgaria, Hungary, Poland) or T-55 (Romania). Other European countries have only light-armored vehicles (Estonia - Infantry fighting vehicles, IFVs, Lithuania – Armored fighting vehicles, AFVs). This diversity of combat vehicles creates several problems, mainly due to logistics and maintenance. Instead of supporting only one type of platform, the military command has to spread its resources to support all those types of platforms and vehicles, each moving with its own propellant. The same statement goes true for the training and combat readiness of troops. It is far more cost effective to build and support one common type of vehicle simulator and load it with specific task modules than a large variety of software training systems. An additional advantage is the interoperability – cadets that trained on a common vehicle platform simulator in one country can easily train to operate other vehicles from the joint task force. For such a training system to be built the requirements need to be defined. One of those is using the same technology. It makes sense to use the same software and one way to achieve it is by using game engines. Game engines, such as Unity 3D or Unreal Engine 4 have the added benefit that can be compiled for different operating systems, without re-writing the source code. If one army has the policy to use Windows OS, and another – to use a Linux distribution (Debian or Cent OS, for instance), the simulation software can be compiled for all those platforms, without added development costs.

Since the common European vehicle platform is still in development, we cannot directly describe the necessary requirements. However, a company - General Dynamics has a working vehicle platform called ASCOD (Figure 2) that is developed specifically for the European market [16]. The modular design architecture offers adaptability and scalability and is remarkably cost-efficient for maintenance. It is a one platform for all combat roles, which is the intention of the common European vehicle platform, as defined by PESCO. That is why we will use the ASCOD platform to analyze the requirements towards designing a common vehicle platform simulator for training.

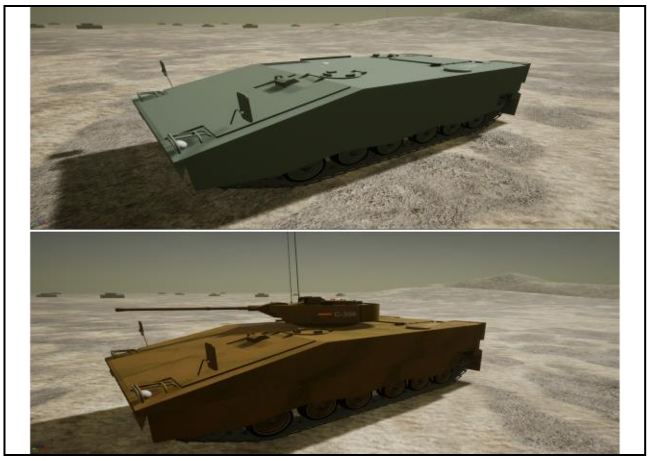

Figure 2: The ASCOD vehicle platform, loaded into a scene in Unreal Engine 4. With the press of a button, the platform changes configuration, transforming its role from a troop transport into a battle tank.

Reading through the PESCO project description and studying the already developed ASCOD, we can identify the following roles, required to be performed by the common vehicle platform:

- IFV – infantry vehicles
- RECCE – reconnaissance, military scouting
- APC – armored personnel carrier
- Repair and recovery
- Bridge layer scenarios
- Mortar operations
- Artillery

In order to fulfill all those roles and scenarios, the common platform should support the mounting of some modules. The modules themselves can be developed as 3D models with embedded functionalities, as suggested by the R.A.G.E. [17] initiative, which can facilitate further the interoperability between member countries.

Analyzing the above requirements, we can define the following main system modules: base platform or hull; weapon systems; accessories; game manager. Another requirement is to easily change vehicle role. That requirement can be integrated into the management module. An example of this functionality is illustrated in Figure 2.

A. *Base platform*

We cannot have a common vehicle platform without a standard base, also known as a hull. This is the part of the vehicle that holds the soldiers, the ammunition, where the driver is located and to which the drivetrain and other accessories are attached. There are two types of most widely used drivetrains – tracked and wheeled. The corresponding module needs to be designed in such a way, as to except both drivetrain configurations. In this paper we consider the tracked drivetrain however, a similar analysis can be performed on the wheeled one.

In addition, the driving and physical simulation components of those platforms need to be an integral part of the 3D asset. In Figure 2 we can see an example of such a component and its integration onto the game engine (Unreal 4).

B. *Weapon systems*

The offensive capabilities of a combat vehicle are essential for its effectiveness in battle. The same holds true for simulating those weapon systems. The match between the real-world and the virtual experience and way of operation should be so similar that even in some scenarios are required to be indistinguishable. [5]. The reason for that is experiences in a simulated virtual environment can be as vivid and effective as the real ones. However, such systems should have a response time of 6 to 20 milliseconds [18]. That makes designing such virtual systems a challenge by itself.

It is important to note that weapon systems are not required in all scenarios. For instance, in a search and rescue scenario the movability and vehicle speed take priority. For those scenarios that do require the mounting of a weapon system, there are several options available [6], [16]:

- RWS – remote weapon systems
- Small caliber RCWS: remote controlled weapons station
- Direct fire – large caliber artillery
- A rotating turret station
- Several multi-caliber machine guns
- Loading and firing the appropriate ammunitions

The above mentioned weapon systems can be represented as sub-modules, each packed with its own functionality. Again, it should be possible to easily change and swap weapon systems per training scenario.

C. *Accessories*

Building a common vehicle platform simulator will require several accessories or add-ons to be included into the simulator design. Those can be: defensive vehicle capabilities, such as smoke grenade launchers; thermal vision, night vision, radar, dozer blade, AVLB (armored vehicle-launched bridge), modular armor, etc. Those add-ons can be installed, removed, turned off and on depending on the combat scenario and current configuration role that is being simulated. For instance, in a bridge layer scenario, the AVLB will be turned on. Or, if the scenario requires a land mines clearance, then a dozer blade add-on will be turned on. Another possible use of the sub-modules is loading different parts of the modular armor and observing how the added weight reflects the change in maneuverability and stress on the suspension. Experimenting with different loadouts and configurations in-game can reduce actual configuration times in the hangar. In addition, the optimal configuration strategy per training scenario can easily be tested in the simulator.

D. *Game manager*

It is a common practice that training simulators include a game manager. The game manager module is responsible for defining and loading different training scenarios, starting, recording and re-playing training sessions. Usually, there is one person in charge of the game manager that directs the simulation, sets goals and evaluates performance. That person is the tutor, also known as instructor. Typical for such simulators, training sessions should be recorded and replayed for performance analysis, hence the need of a training session sub-module. In addition, the tutor should be able to initiate a new simulation or live replay from a certain point of already played scenario. That level of re-play is achieved by actively recording every input action and decision of every player at every game update. Examples of such are role exist in several commercially available military simulators [19], [20].

IV. A MODULAR SOFTWARE ARCHITECTURE FOR SERIOUS GAMES

Analyzing the requirements, we can summarize, that the proposed software simulator should consist of several modules, such as the ones presented in the previous section. That fact points us to use a modular architecture to build our simulator. The architecture that we consider for our prototype is called DiAS [8]. A quick overview is presented in Figure 3.

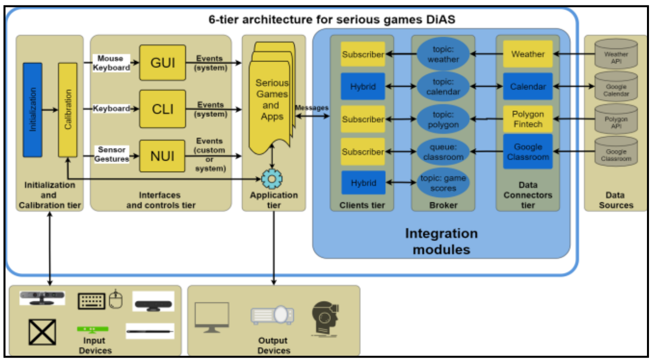

Figure 3: DiAS – a distributed modular software architecture for creating serious games and simulators.

The architecture is independent of its input controls, which allows us to use various input devices – such as commercial-off-the-shelf (COTS) H.O.T.A.S. (Figure 4), or some custom controller. The latter can be a pure hardware controller, such as the one described in Figure 4 (left). Or it can be one that uses a natural user interface [21] for recognizing natural human gestures and translating them into movement and fire controls.

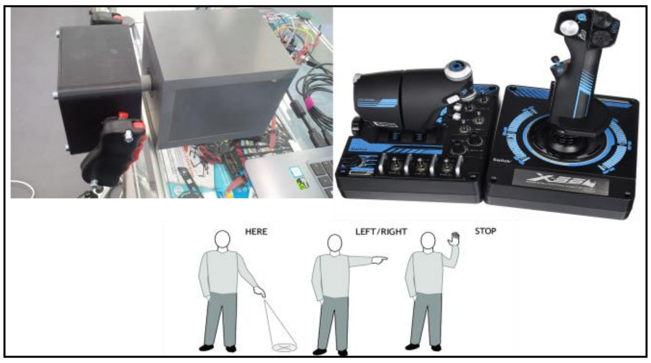

Figure 4: Different controller options: Hands on throttle and stick (H.O.T.A.S.) controller (top right); Custom controller, created for the M1A2 Abrams tank simulator (top left); Gesture NUI controller (bottom)

That diversity is possible since DiAS supports various input interfaces. On the other hand, that architecture allows convenient replacement of output devices. A desktop game can quickly be transformed into a VR game. An additional benefit of using DiAS is that the architecture allows loading of resources via a computer network. Furthermore, the architectural approach is compatible with modern game engines. That will facilitate the effortless cross-compilation of the simulator code to various operating systems (Linux, Windows, macOS) without further overhead. The DiAS is a modular and distributed architecture, which means that different software components can be situated in different locations. For instance, the gunner battle station can run on one computer, the commander's station – on another. Communication is achieved in real time via a local area network. There are simulators which employ a similar concept, mainly VBS4 [19].

DiAS also supports dynamic loading of resources that makes it suitable for loading various modules on demand. The modules can be organized in different packages. That approach decouples the core simulation and input logic from the level (training scenarios) and module logic. Once the simulator is compiled and deployed, new modules can be added and old modules can be updated without the need to re-compile the simulator core. A proposed system design is presented in Figure 5.

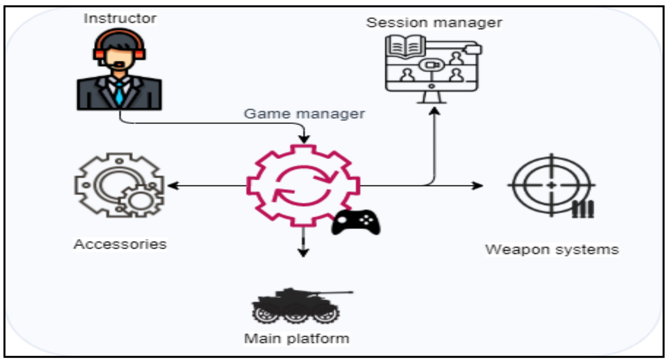

Figure 5: Common vehicle platform simulator design proposal.

As we can see, the proposed system design puts the game manager module in charge of the simulation process. The game manager, being directly controlled by an instructor, is responsible for loading different vehicle modules – the main hull, weapon systems, and accessories. In addition, at the start of each training scenario, it creates a new instance of a session manager - a sub-module, that records and monitors players' inputs and can instantly replay the simulation from any given point of the training exercise. As we mentioned in section III that is achieved by constantly writing the game world state, the players' input commands and the current timestamp to a database. For optimization purposes, the write operation does not need to be performed at each game update. Our empirical trials show that writing the game state once per second is sufficient.

In our proposed architecture, each module may contain various sub-modules which can be loaded dynamically into the simulation during runtime. That reconfiguration of modules allows for rapid prototyping and changing of the training conditions on the fly. For instance, the instructor in charge of the game manager will have the power to swap a battle tank turret with a RCWS to train remote-controlled operations. That action is dictated by the fact that in recent years more effort is spent to promote the use of self-propelled and autonomous vehicles [6].

V. CONCLUSION

In this paper we have presented a possible approach to designing and implementing a common software architecture for European ground combat vehicles simulators. The proposed approach could be useful for future studies in this area, since EDA is moving steadily and surely in this direction. We have shown that using game engines is a major trend when it comes to building military training simulators. The required scenarios and operational roles for the common European vehicle platform can be summarized and achieved by employing a distributed and modular approach, such as DiAS. Last but not the least, we have discussed possible ways of communication between the different software modules and their interoperability on various operating systems and hardware.

ACKNOWLEDGMENT

This research is partially funded by the "Symbiosis between mathematics and computer science" initiative by the Faculty of Mathematics and Informatics of the Plovdiv University. The authors would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

REFERENCES

[1] M. Zyda, A. Mayberry, J. Mccree, and M. Davis, "From viz-sim to vr to games: how we built a hit game-based simulation," Organizational Simulation Rouse/ Organizational, pp. 553–590, 2005.

[2] J. F. Dunnigan, "The complete wargames handbook: how to play, design, and find them," Subsequent edition, December 1, 1992, ISBN: 978-0688103682

[3] P. P. Perla, "The art of wargaming: a guide for professionals and hobbyists," US Naval Institute Press, March 16, 1990, ISBN: 978-0870210501

[4] https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104505/f-16-fighting-falcon, retrieved 01.02.2021.

[5] "Airline pilots fly anywhere in the world – without leaving the ground". Popular Mechanics. Hearst Magazines, p. 87, September 1954.

[6] L. Brozic, "PESCO – More security for Europe, contemporary military challenges," pp. 9-11, 2018, 10.33179/BSV.99.SVI.11.CMC.20.3.00.

[7] Rheinmetall group annual report, https://ir.rheinmetall.com/download/companies/rheinmetall/Annual%20Reports/DE0007030009-JA-2018-EQ-E-00.pdf , retreived on 09.02.2021.

[8] S. Stavrev, T. Terzieva, and A. Golev, “Concepts for distributed input independent architecture for serious games,” CBU International Conference Proceedings: Innovations in Science and Education, Prague, Czech Republic, September 2018.

[9] America's Army. Available at: http://www.americasarmy.com, retrieved: 01.02.2021.

[10] T.S. Hussain and S.L. Coleman, “Design and development of training games,” Cambridge: Cambridge University Press, pp.1-5, 2014.

[11] M. McHugh, A. West, J. Blake, and R. Hall, “Using high velocity rounds to slow aerial descent,” Journal of Physics Special Topics, October 2011.

[12] M. Magier and T. Merda, “Comparison analysis of drag coefficients for supersonic mortar projectiles”, Problemy Techniki Uzbrojenia, vol. 140(4), pp 21-28, 2017.

[13] S. Stavrev and D. Ginchev, “A low-cost battle tank simulator using Unreal Engine 4 and open-hardware microcontrollers,” IEEE Xplore, Proc. XXIX International Scientific Conference Electronics - ET2020, September 16 - 18, 2020.

[14] O. Ganoni, R. Mukundan, and R. Green, “Visually realistic graphical simulation of underwater cable,” 26th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2018), Plzen, Czech Republic, 28 May - 1 June, 2018, ISSN: 1213-6972

[15] A. Dobrovsky, U.M. Borghoff, and M. Hofmann, “Applying and augmenting deep reinforcement learning in serious games through interaction,” Periodica Polytechnica Electrical Engineering and Computer Science, 61(2), p.198, 2017.

[16] https://www.gdels.com/ascod.php, retrieved: 01.02.2021.

[17] W. Der Vegt, W. Westera, E. Nyamsuren, A. Georgiev, and I.M. Ortiz, “RAGE architecture for reusable serious gaming technology components,” International Journal of Computer Games Technology, vol 2016, Article ID 5680526, 10 pages, 2016.

[18] S. Kudrle, M. Proulx, P. Carrieres, M. Lopez, “Fingerprinting for solving a/v synchronization issues within broadcast environments,” SMPTE Motion Imaging Journal. Vol 120 (5), pp. 36–46, July 2011.

[19] VBS4, Available at: https://vbs4.com/, retrieved: 01.02.2021.

[20] https://www.kongsberg.com/digital/products/maritime-simulation, retrieved: 01.02.2021.

[21] S. Stavrev, “Natural user interface for education in virtual environments,” REPLAY, Polish Journal of Game Studies 03, pp.67-80, 2016.