Towards an Orchestrated Game Development Approach to Digital Twinning in Autonomous Vehicles

Markus Schatten, Tomislav Peharda, Igor Tomičić
Faculty of Organization and Informatics, Artificial Intelligence Laboratory
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{markus.schatten,tomislav.peharda,igor.tomicic}@foi.unizg.hr

Abstract. *In this paper we provide a game development perspective on Digital twins (DTs) with emphasis on smart and autonomous vehicles. A DT which comprises a virtual representation of some systems' throughout its lifecycle and used real-time input, simulation, reasoning and machine learning to allow for decision making about the actual system is described as an agent or actor inside a simulated (game) environment. We provide a conceptual model for the development of DTs within an orchestration platform for hybrid artificial intelligence services that is being developed for complex game engines.*

Keywords. game engine, interactive fiction, massively multiplayer on-line games, role playing games, multi agent systems

1 Introduction

Digital twins (DTs) is the concept of having a virtual model that describes an observed physical system as accurately as possible. The difference between a DT and a plain simulation is in robustness of a DT in terms of variety of processes that may be analyzed, as well as DTs often working on real-world data from the simulated system. For example, a simulation may be targeting only a single parameterized use-case in order to conduct an analysis, whereas a DT serves as sort of a virtual environment that is capable to run all use-cases that a physical model may engage in (Rassõlkin et al., 2019; Wang et al., 2022). Classically defined, a digital twin is "a virtual replica of a real-world product, system, being, communities, even cities, that are continuously updated with data from its physical counterpart, as well as its environment" (Jiang et al., 2021). Compared to a "classical" model, which is defined as a simplification/abstraction of the real system, DT can be viewed as a complete mirror image model. A DT model could be developed for the purpose of any complex system. Some examples might include wind turbines, autonomous vehicles (AVs), aircraft tracking etc. Several authors view the practice of DT as the backbone of industry (Colombo et al., 2017; Jiang et al., 2021; Jones et al., 2020; Melesse et al., 2020). Although the term was coined about 20 years ago, it draw significant attention more recently, due to digital infrastructure being more embedded throughout the industry, cities, communities and everyday activities (Batty, 2018).

In this paper we will present benefits and caveats of using an orchestrated game development platform for the implementation of DTs of AVs, including the elimination of duplicated and "boilerplate" code as well as the reuse potential of the original AV system code due to the introduced modularity and proces distribution. However, such benefits might come with the price of additional computing resources.

2 Digital Twins in Autonomous Vehicles

Recently, the number of research papers on the subject of using DTs for AVs has been rising constantly. For example, in (Rassõlkin et al., 2019) it has been investigated how the DT concept can be beneficial on handling propulsion in an electrical vehicle. The idea is that a physical vehicle, which is equipped with plenty of various sensors as well as measurement systems, shares the vehicle propulsion information with the service system which is connected to the DT environment. On demand, the service system conducts simulations inside the DT environment for which it gets a feedback. The service system then uses the feedback to control the vehicle. On a similar note, (Wang et al., 2022) propose the usage of the DT concept, however, with the focus on traffic modelling. They suggest that proper traffic simulations are crucial to AVs safety and performance as different traffic dynamics will require the vehicle to apply adequate driving style depending on it. The DT concept may as well be utilized for better economics (Bhatti et al., 2021) of vehicle aspects in terms of gas consumption, driver assistance systems, battery management systems, vehicle power electronics, etc. Vehicle safety and security is another domain that requires high-level of attention (Almeaibed et al.,

2021). An example that is being brought up in the research is related to insuring consistent behaviour of a vehicle in case of a cyber attack that may harm the vehicles data integrity and consistency. Obviously, such a scenario could even lead to a traffic accident. In that sense, DTs are proposed to simulate data collection, data processing, and analytics.

It is evident that in the domain of autonomous vehicles solely, there is a wide range of aspects that require proper monitoring and forecasting, which could be supported by the usage DTs. This implies that the development of DTs is a challenging task thus requiring proper planning and system architecture. artificial intelligence (AI) and especially machine learning (ML) are greatly advocated for in this domain as they help predicting and proposing future actions based on the current circumstances. Obviously, usage of such methods adds another layer of complexity. As the components may change, it is also significant to make the DTs components extendable and modular in order to easily adjust them to new requirements. For example, with a presence of more AVs every day, that may indicate a different development paradigm shall be applied (Almeaibed et al., 2021; Bhatti et al., 2021). In the following we will try to address these problems related to the need of complexity reduction as well modularization of AV and DT architectures.

3 Awkward Π-nguin Orchestration Infrastructure

In (Schatten, Okreša Đurić, et al., 2020) we have introduced a high-level conceptual model of our microservice orchestration platform shown on figure 1. We have partially implemented[1] this platform in form of APi which is a declarative agent-based programming language based on Π calculus (Milner, 1999) (a process calculus) that allows us to model communication flows between microservices.

APi is agent based (i.e. microservices are represented as autonomous agents) and in fact holonic (Rodriguez et al., 2011) (i.e. agents can be organized hierarchically as holons). APi also features a declarative engine implemented in Python and especially Smart Python Agent Development Environment (SPADE) (Palanca et al., 2020) with parts implemented in ANTLR (syntax parser) (Rajan, 2022) and BASH shell scripts (for inter-process communication). It is programming language agnostic (i.e. microservices can be implemented in any programming language that can be executed under UNIX-friendly environments) and communication protocol agnostic (i.e. services can communicate using stdin/stdout process communication, files, HTTP, TCP, UDP or WebSockets whilst other protocols can be added through a partially implemented plug-in system) which is accomplished by creating wrappers around each microservice to be used in an ensemble. These wrappers then behave as agents inside a holonic multi agent system (HMAS) and can be used as building blocks for the creation of complex architectures. In a way, APi is a high-level network implementation of the UNIX inter-process communication system based on input/output redirection in which special programs (called filters) can be used to create complex chains of processing whereas each process reads the standard output of a previous process as its input and "filters" it producing new output to be used in a consecutive process. The same basic idea applies to APi but with the addition that these processes (in our case microservices or agents in the end effect) can be distributed among various servers and communicate across a network regardless of their implementation.

The syntax of the APi programming language is influenced by Python and Π-calculus. The following excerpt (listing 1) shows some of the main features of the language.

Listing 1: Example APi syntax

```
environment :
 input1 => { 'val1': ?x, 'val2': ?y }
 output1 <= { 'action': ?act }

channel c :
 { 'data'->?x } -> { 'payload' -> ?x }

agent a ( x ) :
 input1 -> self
 self -> ?x

agent b :
 c -> self
 self -> output1

start a( c ) & b
```

In this excerpt a holon is defined which communicates with the environment through channels `input1` and `output1`. A local channel (`c`) is defined which transforms accepts messages of the form `{ 'data'->?x }` (JSON format with logic variables) defined on the left-hand side, and emits messages of the format `{ 'payload' -> ?x }` defined on the right-hand side. Additionally two agents (microservice wrappers) are defined (`a` and `b`). Agents' definitions correspond to corresponding service-descriptors which describe how to start the service, how it communicates, what type of service it is etc., and can take arguments. In each agent definition there are one or more channel mapping definitions, for example agent `a` will direct messages from `input1` to its own input (keyword `self`) and redirect its outputs to channel held in the variable `?x` (an instantiation argument). Both agents are started consecutively (when agent `a` finishes successfully agent `b` will be started). The de-

[1] Awkward Π-nguin (APi) is open source and available at https://github.com/AILab-FOI/APi

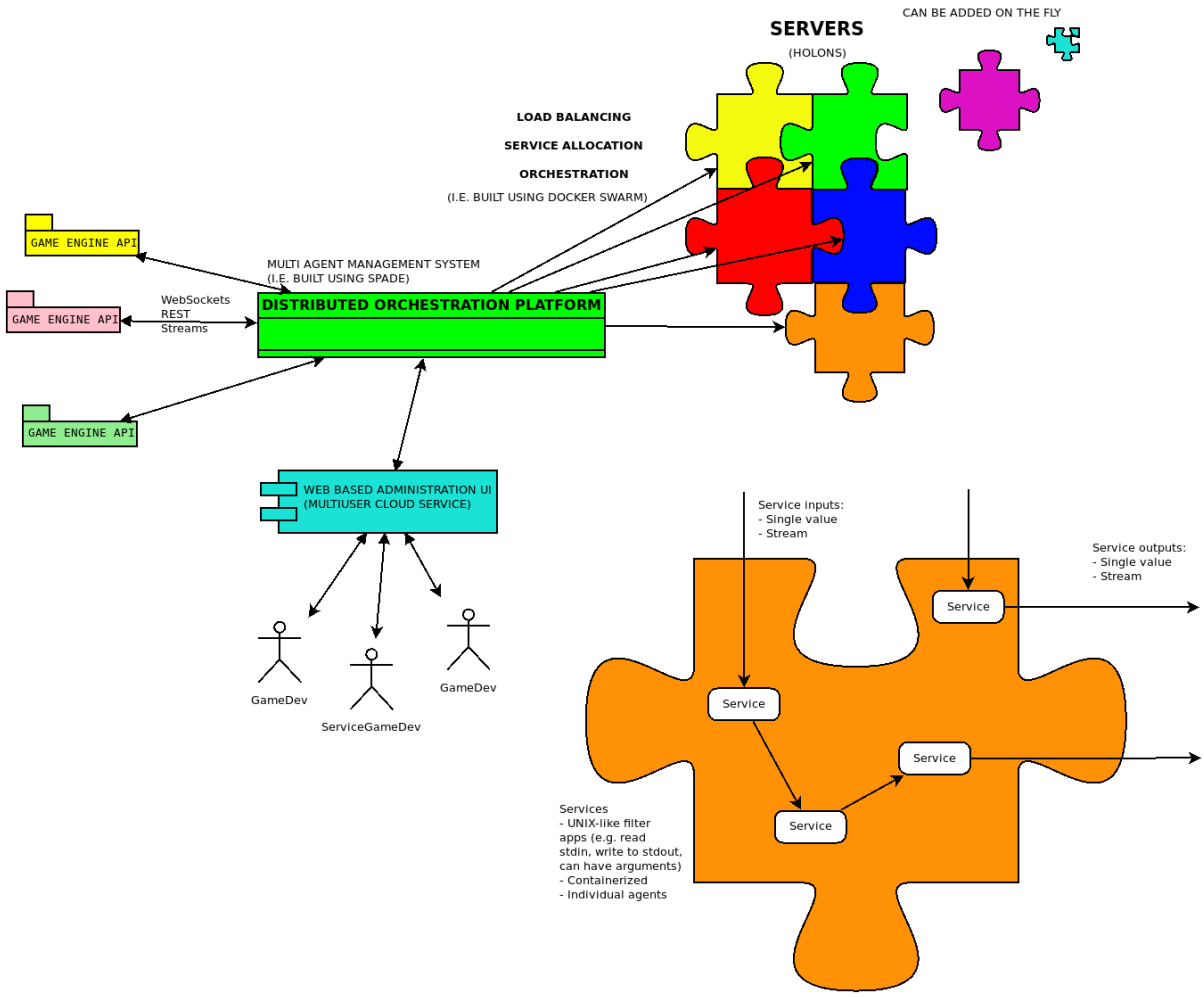

Figure 1: High-level Conceptual Model (Schatten, Okreša Đurić, et al., 2020)

fined holon can then be imported into other holons to create complex orchestration architectures. For example if the above holon had been defined in a file called `holon1.api` we could import and use its output as shown in listing 2

Listing 2: Example APi syntax

```
from holon1 import output1 as h1

agent a ( ... ) :
     h1 -> self
```

The platform is intended for the orchestration of especially AI and computer game related microservices as outlined in (Schatten, Okreša Đurić, et al., 2019) and (Schatten, Tomičić, et al., 2020). In that regard, it allows us to connect and assemble complex data streaming applications which is a prerequisite for the implementation of AV systems and consequently DTs.

4 Digital Twins as Game Actors

A common software development pattern in computer game design is the actor model (Lindley, 2002; Silva et al., 2000) which corresponds to the multiagent systems (MASs) model or more precisely to intelligent virtual environments (IVEs) (Okreša Đurić et al., 2019; Rincon et al., 2014). In such environments actors (agents) interact with their environment which consists of other (potentially intelligent and or autonomous) agents as well as dynamic or inanimate, static objects.

It has been argued that IVEs provide us with the necessary toolset to develop, test and monitor AVs architectures (Ferreira et al., 2002; Schatten, Okreša Đurić, et al., 2019; Zhang et al., 2008). Herein we argue additionally that a serious game development environment with an orchestration platform as described in section 3 can provide additional benefits. Firstly, game development environments usually feature an integrated game engine that provide basic components necessary for the development of IVEs including but not limited to 3D physics, various sensors, a simulation engine, AI methods, actor templates, cameras, logging facilities etc. Additionally, it provides us with instant visual monitoring of the environment. Secondly, an orchestration platform allows us to implement complex data streaming ensembles that might consist of various ML cloud services that are usually to resource consuming

to be executed on one computer during a simulation and/or real-time monitoring.

In order to implement a DT of an AV real-time monitoring of the vehicle in sense of sensor inputs, actuator outputs as well as internal decision making mechanisms is crucial. With an orchestration platform in which multiple cloud-based servers can be employed to handle various complex tasks like pattern recognition, automated planning or reasoning; such a workload can be distributed and handled. In the following we will show a simple proof-of-concept implementation of a DT of an AV using the APi language.

An implementation of a AV system usually consists of a number of sensors (which might include LiDAR, radar, cameras etc.), a number of decision making services (e.g. ML models, complex AI algorithms including but not limited to search and planning techniques), a number of communication services (communicating with relevant on-line services like traffic congestion and other information, routing services etc.), as well as a number of actuators (e.g. steering, car safety monitoring like tire pressure, fuel or battery life, infotainment systems etc.). All these subsystems can be wrapped into mutually communicating agents inside a holon as described in section 3.

For sake of simplicity, let us assume that the AV is a holon (i.e. a multi-agent system of various component) that communicates by reading its sensors and appropriate external services and writing commands to appropriate actuators. An implementation of such a holon in APi would look similar to the code in listing 3

Listing 3: AV holon implementation - `AVHolon.api`

```
environment :
     sensor_1 => ...
     ...
     sensor_n => ...
     external_service_1 => ...
     ...
     external_service_k => ...
     actuator_1 <= ...
     ...
     actuator_m <= ...

agent AV :
     sensor_1 -> self
     ...
     sensor_n -> self
     external_service_1 -> self
     ...
     external_service_k -> self
     self -> actuator_1
     ...
     self -> actuator_m
```

It is important to note that by defining the AV system in this way we abstract away all internal mechanisms of communication, meaning that we now have standard communication interfaces (channels) for all included components and that these interfaces can be reused, i.e. we can redirect them at will to other agents or holons. This possibility of redirection is a special feature of the declarative engine of APi - it allows us to listen to or write to any channel in the given scope (i.e. the current holon which in this case acts as a namespace). This feature is most important for the implementation of a DT since it allows us to "tap into" communication flows and process real time data.

In order to have benefits from a DT we should additionally add a corrective input, i.e. an additional communication channel to allow for intervention if the AV system does not function as intended. The environment would then look as shown in listing 4.

Listing 4: Adding a corrective to the AV holon

```
environment :
     corrective => ...
     sensor_1 => ...
     ...
...
```

Having the prerequisites defined, the implementation of a DT using APi is now straightforward. We firstly have to import all communication channels and redirect them to a new agent which will represent the DT. In the end we start the DT in parallel with the AV holon defined above as shown in listing 5.[2]

Listing 5: Digital twin implementation

```
from holonAV import sensor_1 as s1
...
from holonAV import sensor_n as sn

from holonAV import external_service_1 as
    ↪ e1
...
from holonAV import external_service_k as
    ↪ ek

from holonAV import actuator_1 as a1
...
from holonAV import actuator_m as am

from holonAV import corrective as cr

agent AVtwin :
     s1 -> self
     ...
     sn -> self
     e1 -> self
     ...
     ek -> self
     a1 -> self
     ...
     am -> self
     self -> cr
```

[2]The | character between two agents denotes parallel execution.

```
start holonAV | AVtwin
```

In this way the DT can monitor all inputs and all outputs of the original AV system and use this data to update its IVE, simulate and analyze its decision making process as well as send corrective information if needed.

5 Discussion

The main benefit of an orchestrated implementation as shown in section 4 is the modular approach that minimizes code duplication especially in "boilerplate" communication code. In a non-orchestrated implementation each service instance from the original AV system which communicates in any way (which is in fact most instances) would have to be either rewritten or at least extended to forward communicated data to the DT. Also, for the implementation of the DT each component simulation of the original would have to be implemented anew in an IVE, game engine or other simulation environment using native development tools and languages. By using agent wrappers around these components most (not hardware specific) code could be reused and work in a more or less unchanged way as it would in the original setting.

Still, there are caveats of course. By wrapping the original AV system components as agents additional overhead code is introduced which might use additional computing and resources in order to function and communicate with the orchestration platform. Such additional resources might not be available or it might be not desirable to use them in real-time and mission critical settings. Additionally, a problem that isn't solved using the given approach is the need to implement the update process of the IVE by using real-time data from the original AV system. These update components need to be implemented as additional services which will update the simulated environment based on sensory data and integrated with the other (possible reused) services. On the other hand, the integration process is fairly trivial, due to the modularity of APi, i.e. they can be introduces without intervention into the code of the original services, which might not be the case in a non-orchestrated setting.

6 Conclusion

In this paper we have argued the benefits and caveats of using an orchestrated game development platform for the implementation of DTs of AVs. By using the APi platform that we are actively developing we have shown how to implement a basic DT ensemble. The most important advantages of using an orchestrated game development platform, besides the various building blocks provided by game development software (especially game engines), include the elimination of duplicated and "boilerplate" code as well as the reuse potential of the original AV system code due to the introduced modularity and proces distribution. These benefits come with an expense of introducing the need of additional computing resources on the AV system side.

Our future research is aimed towards the implementation of a proof-of-concept digital twinning environment for AVs in which we shall address some of the issues outlined above.

Acknowledgement

This work has been fully supported by the Croatian Science Foundation under the project number IP-2019-04-5824.

References

Almeaibed, S., Al-Rubaye, S., Tsourdos, A., & Avdelidis, N. P. (2021). Digital twin analysis to promote safety and security in autonomous vehicles. *IEEE Communications Standards Magazine*, *5*(1), 40–46.

Batty, M. (2018). Digital twins.

Bhatti, G., Mohan, H., & Singh, R. R. (2021). Towards the future of smart electric vehicles: Digital twin technology. *Renewable and Sustainable Energy Reviews*, *141*, 110801.

Colombo, A. W., Karnouskos, S., Kaynak, O., Shi, Y., & Yin, S. (2017). Industrial cyberphysical systems: A backbone of the fourth industrial revolution. *IEEE Industrial Electronics Magazine*, *11*(1), 6–16.

Ferreira, F. P., Gelatti, G., & Musse, S. R. (2002). Intelligent virtual environment and camera control in behavioural simulation. *Proceedings. XV Brazilian Symposium on Computer Graphics and Image Processing*, 365–372.

Jiang, Y., Yin, S., Li, K., Luo, H., & Kaynak, O. (2021). Industrial applications of digital twins. *Philosophical Transactions of the Royal Society A*, *379*(2207), 20200360.

Jones, D., Snider, C., Nassehi, A., Yon, J., & Hicks, B. (2020). Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, *29*, 36–52.

Lindley, C. A. (2002). The gameplay gestalt, narrative, and interactive storytelling. *CGDC Conf.*

Melesse, T. Y., Di Pasquale, V., & Riemma, S. (2020). Digital twin models in industrial operations: A systematic literature review. *Procedia Manufacturing*, *42*, 267–272.

Milner, R. (1999). *Communicating and mobile systems: The pi calculus*. Cambridge university press.

Okreša Đurić, B., Rincon, J., Carrascosa, C., Schatten, M., & Julian, V. (2019). Mambo5: A new ontology approach for modelling and managing intelligent virtual environments based on multi-agent systems. *Journal of Ambient Intelligence and Humanized Computing*, *10*(9), 3629–3641.

Palanca, J., Terrasa, A., Julian, V., & Carrascosa, C. (2020). Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, *8*, 182537–182549.

Rajan, H. (2022). Antlr: A brief review.

Rassõlkin, A., Vaimann, T., Kallaste, A., & Kuts, V. (2019). Digital twin for propulsion drive of autonomous electric vehicle. *2019 IEEE 60th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, 1–4.

Rincon, J., Garcia, E., Julian, V., & Carrascosa, C. (2014). Developing adaptive agents situated in intelligent virtual environments. *International conference on hybrid artificial intelligence systems*, 98–109.

Rodriguez, S., Hilaire, V., Gaud, N., Galland, S., & Koukam, A. (2011). Holonic multi-agent systems. In *Self-organising software* (pp. 251–279). Springer.

Schatten, M., Okreša Đurić, B., & Tomičić, I. (2019). Towards simulation of ambient intelligence in autonomous vehicles using car racing games. *Central European Conference on Information and Intelligent Systems*, 3–6.

Schatten, M., Okreša Đurić, B., & Tomičić, I. (2020). Orchestration platforms for hybrid artificial intelligence in computer games-a conceptual model. *2020 Central European Conference on Information and Intelligent Systems*, 3–8.

Schatten, M., Tomičić, I., & Okreša Đurić, B. (2020). Towards application programming interfaces for cloud services orchestration platforms in computer games. *2020 Central European Conference on Information and Intelligent Systems*, 9–14.

Silva, D. R., Siebra, C. A., Valadares, J. L., Almeida, A. L., Frery, A. C., da Rocha Falcão, J., & Ramalho, G. L. (2000). A synthetic actor model for long-term computer games. *Virtual Reality*, *5*(2), 107–116.

Wang, S.-H., Tu, C.-H., & Juang, J.-C. (2022). Automatic traffic modelling for creating digital twins to facilitate autonomous vehicle development. *Connection Science*, *34*(1), 1018–1037.

Zhang, T., Liu, X., Mei, T., Tang, G., Li, B., & Wang, X. (2008). A novel platform for simulation and evaluation of intelligent behavior of driverless vehicle. *2008 IEEE International Conference on Vehicular Electronics and Safety*, 237–240.