Consistency Management for Peer-to-Peer-based Massively Multiuser Virtual Environments

Gregor Schiele*, Richard Süselbeck*, Arno Wacker†, Tonio Triebel*, and Christian Becker*

*University of Mannheim
Mannheim, Germany
{gregor.schiele | richard.sueselbeck | tonio.triebel |
christian.becker}@uni-mannheim.de

†University of Duisburg-Essen
Duisburg, Germany
arno.wacker@uni-due.de

***Abstract*— Massively Multiuser Virtual Environments (MMVEs) require a seamless and consistent execution. To provide this, the MMVE must include a sophisticated consistency management. This management must adapt its behavior to different activities carried out in the MMVE as well as different system loads. As an example, a movement activity must be handled differently to picking up an object. During times of high system load, a lower level of consistency might be acceptable if this enables the MMVE to continue operating with low network delay. Developing such a consistency management is a complex and time consuming effort. To support MMVE developers with this task, we propose the creation of a dedicated consistency management infrastructure. This infrastructure can be configured by developers for their specific MMVE and makes it easier to maintain a consistent MMVE state.**

I. Introduction

A Massively Multiuser Virtual Environment (MMVE) allows thousands of users worldwide to interact with each other in a common environment in real-time. A crucial requirement for such systems is consistency [1]. As an example, if multiple users try to pick up the same object at the same time, the system has to guarantee that only one user actually receives the object. Otherwise, the users may become irritated by unusual effects in the environment's behavior.

At the same time, MMVEs must be highly responsive and progress seamlessly. Users expect the system to react promptly to their input and to progress without sudden state changes or rollbacks. To fulfill these conflicting goals, MMVE developers have to provide a complex consistency management which allows the MMVE to balance dynamically between consistency, responsiveness, and seamlessness. This is even more difficult to achieve in peer-to-peer (P2P) based MMVEs. In such systems, the MMVE is executed cooperatively on the hosts of its users and requires no central server. Therefore, there is no central entity to synchronize the participating entities and guarantee a consistent environment. To help reduce the development effort for such MMVEs, we propose the development of a novel consistency management infrastructure for P2P-based MMVEs. This infrastructure allows the application (i.e. the MMVE using our system) to choose between different consistency models at runtime.

Our work is executed in the context of the peers@play project [2]. The goal of the this project is the development of a comprehensive communication middleware for P2P-based MMVEs, including support for scalable update propagation, security, and consistency. In this paper we present the main design decisions and the architecture of our consistency management infrastructure. We present its main concepts and show how they can be used by developers.

The paper is structured as follows. First, we present our system model and discuss requirements for our consistency management infrastructure. We provide an overview of related work and present our approach. Finally, we offer a short conclusion as well as an outlook on future work.

II. System Model

Our system model consists of a number of users that want to use an MMVE, their devices, a communication network and the MMVE software. Users may be located at any place in the world. The number of users is a priori undetermined and can change dynamically. Each user owns a computing device, which executes the MMVE software and is connected to a common communication network, e.g., the Internet. We call such a device a peer. Each user is represented in the MMVE by a special character, called *avatar*. To interact with the MMVE, the user instructs his avatar to perform different activities, e.g., moving around or picking up an object. In addition, activities can be initiated by so-called non-player characters (NPCs). The state of the MMVE is distributed on the participating peers. If the user executes an activity, the system creates update events describing the resulting state changes. It sends these events to all other peers that hold a copy of this state. We assume the presence of a suitable P2P-based communication middleware to do so. The middleware determines the peers to which an event must be sent and provides all necessary communication services. In the context of the peers@play project, this functionality is provided by the peers@play middleware.

III. Requirements

A consistency management infrastructure in P2P-based MMVEs faces a number of requirements not encountered in traditional client/server-systems. In the following we analyze these requirements in more detail.

1) Flexibility: Users perform a multitude of different activities in an MMVE, e.g. moving their avatar around, picking up objects, and interacting with other users or NPCs. The corresponding consistency requirements vary widely, depending on the activity's type and situation. As an example, the exact location of a user's avatar may not be crucial for nearby users, as long as they do not interact with it. However, once an interaction between users is initiated, position updates must be tightly synchronized. On the other hand, users are often willing to accept different delays for different activities. When a user picks up an object, he might tolerate several seconds delay. However, the same user will most likely not tolerate such a high delay each time he moves his avatar. In summary, both the required consistency and the maximum acceptable delay vary between different activities and situations. The consistency management must be flexible enough to allow the application to balance both these factors with each other.

2) Adaptability: As the MMVE is executed in a peer-to-peer system, the available devices and network resources can vary widely and without warning. If the user decides to log out of the system and turn off his computer, the system has to adapt dynamically to loosing this peer. In addition, user population in a given environment can vary. Thus, the system has to handle different situations concerning resource availability and system load. The consistency management must provide MMVE developers with means to react to high load situations to make sure that the MMVE maintains a satisfactory performance. As an example, the system might decide to lower the consistency requirements for certain activities in case of a high system load to maintain highest possible system responsiveness.

3) Extensibility: Different MMVEs may require different consistency models. To allow the consistency management infrastructure to stay independent of any specific MMVE, the infrastructure must enable MMVE developers to extend it with their own consistency models. As an example, an MMVE could include the possibility to buy objects from other users using real currency. To support such trading activities, the developer wants to include a consistency model that guarantees transactional properties, such as atomicity. In addition, it should be robust against attacks from other peers. On the other hand, a game-themed MMVE requires consistency models that are optimized towards latency.

IV. Related Work

In the past, a number of different approaches for consistency in MMVEs have been designed. These approaches can be divided into two classes, conservative and optimistic. Conservative approaches (e.g. [3],[4]) delay the processing of events until they have confirmed that it is safe to do so. Using this approach, if a peer receives an update event, it delays applying it to its local state until it can guarantee that it will never receive another update event that should have been applied previously. Conservative approaches are able to provide a high level of consistency, e.g. sequential or strong consistency. The ordering of events will always be the same on each peer. In addition, the user can be sure, that an activity will never be undone once it has been executed. However, a consistency model based on a conservative event synchronization approach might lead to high latencies, as faster peers could be required to wait for slower ones. In contrast, optimistic approaches (e.g. [5],[6]) process the events immediately, and try to correct possible inconsistencies through techniques such as rollbacks. They can be used to realize weaker consistency models, e.g. weak consistency. For the user, this means that an activity is executed instantly but that he can never be sure if it will be undone in the future. Hence, consistency models based on an optimistic event synchronization approach are specifically well suited for highly interactive activities that can be reversed without irritating the user too much. None of these isolated approaches are able to provide the necessary flexibility to support all different kinds of activities that are experienced in an MMVE.

For client/server-based MMVEs, several consistency infrastructures exist. Lu et al. [7] propose a consistency control mechanism that allows the application to select between two different levels of consistency, depending on, e.g., system load. The FITGap framework [8] allows states in an MMVE to be associated with different replication models, depending on their consistency requirements. In contrast to these centralized approaches, we provide a decentralized P2P infrastructure. A hybrid architecture was proposed by Pellegrino et al. [9]. It uses a central server to enforce a consistent game state using an optimistic synchronization protocol with rollbacks, while all other information is distributed via a P2P approach. Finally, OpenPING [10] is a middleware for networked games that includes different consistency models and offers dynamic adaptation through reflection. However, the supported consistency models are hard coded into the system and cannot be extended with additional models.

V. Our Approach

Our goal is to provide a consistency management infrastructure for P2P-based MMVEs. Our infrastructure should fulfill the requirements given before, namely: 1. flexibility, 2. adaptability, and 3. extensibility. In this section we discuss the main concepts and design decisions underlying our approach. Subsequently, we describe the architecture of the proposed infrastructure and illustrate how the different subsystems interact with each other using an example activity.

A. Design Rationale

Our system is based on a number of concepts that we discuss in the following. The main idea is to push application knowledge about the current situation of the MMVE and the performed activities into the consistency management to allow the latter to optimize system operation and minimize the latency experienced by users. In addition, the application is enabled to access information about the current situation in the underlying P2P system in order to adapt its requirements to this situation.

1) Selectable Consistency Model: As discussed before, it can be insufficient to rely on a single consistency model. Instead, different situations and activity types should be handled with different consistency models. It may be acceptable in many situations to have loosely synchronized MMVE states between users that do not interact with each other. A user passing by another user's avatar does not need to have a perfectly synchronized view of the other user's current location. However, once both users interact with each other, e.g., by picking up the same object, a higher level of consistency is necessary. To facilitate this, two main approaches are possible. First, the system can provide a single integrated consistency model that provides different behaviors for different situations. Clearly, such a model would be complex and hard to maintain. The second possibility is to provide multiple independent consistency models and select the best model at runtime for each activity. We choose the second approach.

The MMVE can specify for each event which consistency model is required. This allows it to select the best consistency model for a given activity in a certain situation. Thus, if no strong consistency is required, the system can provide higher interactivity. For events for which strong consistency is mandatory, interactivity can be traded for consistency. Choosing the best model for a given situation requires MMVE-specific knowledge. Thus, to make consistency models reusable for different MMVEs, this decision should be separated from the models and pushed into the MMVE implementation. This approach fulfills the first requirement (flexibility).

2) Reflection Application Programming Interface (API): Allowing the application to select a consistency model enables the application developer to choose the best model for each activity. However, as described before, this selection should also depend on the current system load. If the system load is high, the application should be made aware of this and could react, e.g., by choosing a more resource efficient model. To enable this, we propose to offer a reflection API. Using this API, the application can query the consistency management on its current state. More precisely, the consistency management provides information about the currently expected delay for a given consistency model in a specified part of the MMVE. This delay depends, e.g., on the current system load in the corresponding P2P network. Clearly, the API could offer more detailed low level information, e.g., about the bandwidth currently available between specific peers. However, to use this information for selecting a suitable consistency, the application would require knowledge about the consistency model's implementation, e.g. what messages will be sent to whom. The reflection API fulfills the second requirement (adaptability).

We deliberately chose to push the decision how to react to a high system load into the application. Alternatively, this could be done by the consistency management internally. We argue, however, that only the application knows if selecting another consistency model is acceptable for a given activity. Furthermore, the application has to know which model is used to appropriately present the result of an activity to the user. As an example, consider an object that needs be removed from the user's inventory due to a rollback. To warn the user, the application could highlight this object in the graphical interface.

3) Consistency Plugins: So far, we have addressed the first two requirements, flexibility and adaptability. Still missing is to enable MMVE developers to extend the system with MMVE-specific consistency models – fulfilling the third requirement (extensibility). To do so, we propose to extract the implementations of consistency models from the infrastructure and place them into so-called consistency plug-ins. A consistency plug-in encapsulates a protocol implementing a specified consistency model. If an MMVE developer wants to add an additional consistency model, he can do so by implementing a consistency plug-in. He can also select the consistency models that are suitable for his MMVE and include only the plug-ins implementing them in the final product. Generic consistency models can be reused for multiple MMVEs, making development more efficient. Note that in contrast to other plug-in models, our plug-ins are selected at development time. At runtime, no new plug-ins are added. While offering this would allow updating the MMVE more easily, the ability to update code dynamically should be provided for the whole MMVE software and not for plug-ins, only. Therefore, we omit dynamic adding and removal of plug-ins to gain better system performance.

4) Consistency Sessions: All three requirements have now been addressed by the concepts discussed so far. To help the consistency management further optimize the MMVE execution, we provide an additional concept, called *consistency sessions*. Consistency sessions allow the application to group a number of events into a common context. As an example, the application can group events that are causally dependent on each other into a session. Thus it is possible to efficiently perform a partial rollback of these events. In our example of a user picking up an object, assume that the application uses a consistency plug-in with optimistic synchronization. Some time later, the application discovers that it must undo the pick up event. Normally, this would require to rollback the entire state of the MMVE. Alternatively, the MMVE could analyze all causal dependencies between this event and later ones to select the events to undo. If the application opens a new session before picking up the object, it can add all events that depend on the first one into it, and can easily determine the set of events to undo if necessary.

Note that placing an event into a consistency session does not mean that it will be delivered only to other members of the session. Reusing our example of a user fighting an NPC, both entities share a common session for the fight and movement events are tightly synchronized between them. If another user watches the fight, his peer should receive these events, too. However, he will most probably accept a lower level of consistency, if this prevents the fight from being slowed down. Therefore, the MMVE can select the required consistency model for an event per session. In addition, it can select the consistency model that should be used for delivering the event to peers that are not included in a session. In our example, the MMVE would specify the use of strong consistency for movement events when sending them to entities in the session, i.e. the fighting user and the NPC, and no consistency when sending them to other entities, i.e. the observing user.

All events in a session must use the same consistency plug-in. While this restricts the usage of sessions, it makes implementing consistency plug-ins much easier. Otherwise, if a session contains events that use optimistic synchronization, and the MMVE tries to add an event that uses conservative synchronization, the plug-in implementing the conservative synchronization must guarantee that none of the earlier events in the session will ever have to be undone. Otherwise it would have to undo its own event, too, breaking its own semantic. To ensure this, the different consistency plug-ins would have to interact with each other. If an MMVE requires events in the same session to use different consistency models, the MMVE developer could implement a plug-in that integrates these models and makes sure that mixed sessions are executed correctly.

An event can be added into several sessions. This may lead to complex situations that must be handled by the MMVE. As an example, take an event that is added into a session with conservative synchronization and into a session with optimistic synchronization. Later, the system decides to roll back the events in the second session. However, the first session cannot be undone, as conservative synchronization was used. Currently such dependencies are not handled by the consistency management and must be dissolved by the MMVE. While we expect such situations to be rare, we plan to look into this issue more closely in the future.

5) Dynamic Relocation: As another optimization, we propose to let the consistency management initiate the dynamic relocation of objects in the MMVE to other peers. A prominent example for this is a user fighting an NPC in a game-themed MMVE. The activities of the user and the NPC have to be carefully synchronized to guarantee a consistent MMVE state and a satisfactory progression for the user. If the latency between the user's peer and the peer executing the NPC is high, this synchronization may decrease the system responsiveness unacceptably. The consistency management can detect this situation and try to optimize the communication delay. To do so, the management relocates the execution of the NPC to the user's peer. Thus, the network delay is reduced to zero and consistency and responsiveness can be provided efficiently.

As the overhead for relocating objects is usually high, the system should do so only if the resulting performance gain outweighs the initial effort. This depends on several factors, e.g. the current network delay, or how long the interaction between the peers will last. If synchronization between them is only required for a few events, no relocation is performed. On the other hand, if the peers are involved in a long term interaction, a relocation should be initiated. To detect such situations, we rely on the application. It notifies the consistency management about a potentially long running interaction. The management can then check if a relocation results in a sufficiently large performance enhancement and execute it.

B. System Architecture

Our proposed consistency management infrastructure consists of five system components as shown in Figure 1.

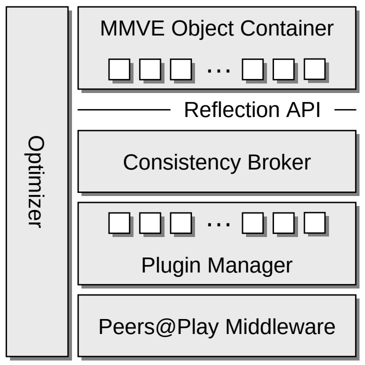

Fig. 1. Architecture of the consistency management infrastructure.

The *MMVE object container* manages MMVE objects. It allows objects to receive and send update messages if their state changes. In addition, the computation of an MMVE object's behavior can be relocated to another peer. To do so, it maintains the object's life cycle, transfers its current state and hands over control to the moved object. To use the object container, application developers inherit new objects from predefined classes.

Located below the object container, the *consistency broker* is the central contact point for the MMVE and its objects. Objects can register for incoming state update events, send update events to other peers, select the consistency models to use and notify the consistency management about possible MMVE object relocations. Furthermore, the broker implements the reflection API. MMVE objects can query the broker for the expected delay for a given consistency model. To support both discrete and continuous queries, the API offers both a synchronous system call as well as an asynchronous callback interface. To offer current delay information, the broker manages a local history of past interactions and exchanges such histories with remote brokers. Therefore a broker is able to provide the application developer with current information, even if the user's avatar has recently moved into another area. As an example, the MMVE might experience high load in a specific area in the MMVE. When a user moves his avatar from another low-load area into this one, his consistency broker exchanges historical data with others in the area and learns of this situation. It can notify the application of this, enabling the latter to initiate an appropriate reaction, e.g. requesting a lower level of synchronization for its activities. Finally, the consistency broker is responsible for forwarding update events between MMVE objects and plug-ins, and creates and maintains consistency sessions.

Consistency plug-ins are maintained by the *Plug-in Manager*. It offers an execution environment for plug-ins, maintains a list of installed plug-ins and mediates events between the consistency broker and the plug-ins. Each plug-in implements a consistency protocol, realizing a defined consistency model. To communicate with remote plug-ins, they interact with the peers@play communication middleware (see Section II), which determines the correct peers to send events to, etc.

Finally, the *Optimizer* adapts the system behavior to provide lower delays for executing interactions. It spans all system layers and combines information from different system components. Currently, it is used to initiate the relocation of MMVE objects. To do so, it accesses the communication middleware to learn about the current system state, e.g. network delay and bandwidth. To learn about long running interactions, it interacts with the consistency broker. If the Optimizer decides to relocate a MMVE object it contacts its local MMVE object container and initiates relocation. In the future, the Optimizer may contain additional optimization algorithms.

C. Example

After discussing the design rationale of our infrastructure and presenting its architecture, we give an example of an activity that is executed within our system. In this example, two users u_1 and u_2 attempt to pick up an object o at roughly the same time. The system needs to ensure that only one of the users receives the object.

First, the application selects as suitable consistency model. It decides that picking up an object is an event for which a strong consistency model is appropriate. The application queries the broker if the system is currently able to provide strong consistency, while maintaining adequate network delays. For the sake of simplicity, we assume that the application is satisfied with the current system state and a strong consistency model can be used. In addition to selecting a consistency model, the application also needs to determine if the event should be part of a consistency session. Since the application decided to use strong consistency, the ability to perform a partial rollback is not needed. Thus, the application decides not to include the event in a session.

Next, the application contacts the consistency broker and instructs it to enforce strong consistency by using the respective plug-in. It also hands the broker a set of context variables that are used by the strong consistency plug-in. In this example, the context contains an area of effect for the pick up event. This area determines all locations from which the object can be picked up. It is used later by the plug-in to determine which peers to synchronize with.

After being called by the broker, the plug-in determines all users and NPCs that are near o and thus would be able to pick it up. Here, this includes only u_2. Note that the plug-in does not know that u_2 has actually just tried to pick up o. It is only aware that she is in a position to do so, and therefore needs to be considered. The plug-in now contacts all affected users and NPCs (again only u_2) and notifies them that u_1 wants to pick up o at this time. In order to this, it uses the peers@play communication middleware. When the corresponding consistency plug-in at u_2's peer receives this request, it checks if it has already taken o or is in the process of doing so. Here, we assume that u_2 actually tried to pick up o a short while after u_1. Therefore u_1 is entitled to receive the object. The plug-in then passes this information back to the broker via the plug-in manager, which informs the application of the success of the event. During this entire process, u_1 has to wait for the confirmation of the consistency management. Only after the confirmation arrives, does the item actually appear in his inventory. The entire process looks almost identical for u_2, with the exception that the consistency management will report that o has already been taken. Thus, u_2 receives a message, that the object is no longer available.

VI. Conclusion and Future Work

In this paper we have proposed a consistency management infrastructure and presented its main concepts. We also gave an overview on the architecture of our infrastructure. Currently we are implementing the discussed concepts and components, as well as integrating a number of different consistency models. So far, our infrastructure offers no support for handling dependencies between different consistency plug-ins. As discussed before, this might be necessary if events are added into multiple consistency sessions. In future work we plan to extend our infrastructure with suitable coordination mechanisms for this.

References

[1] G. Schiele, R. Sueselbeck, A. Wacker, J. Haehner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh International Workshop on Global and Peer-to-Peer Computing*, 2007.

[2] University of Mannheim, Duisburg-Essen and Lebniz University of Hannover, "peers@play homepage," published on the WWW at http://www.peers-at-play.org/, 2008.

[3] N. E. Baughman and B. N. Levine, "Cheat-proof playout for centralized and distributed online games," in *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-01).* Los Alamitos, CA: IEEE Computer Society, Apr. 22–26 2001, pp. 104–113.

[4] J. Steinman, "Scalable parallel and distributed military simulations using the speedes framework," NASA, Tech. Rep., 1995.

[5] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and timewarp: Providing consistency for replicated continuous applications," *IEEE Transactions on Multimedia*, vol. 6, pp. 47–57, 2004.

[6] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," in *NetGames '02: Proceedings of the 1st workshop on Network and system support for games.* New York, NY, USA: ACM, 2002, pp. 67–73.

[7] T.-C. Lu, M.-T. Lin, and C. Lee, "Control mechanisms for large-scale virtual environments," *Journal of Visual Languages and Computing*, vol. 10, pp. 69–85, 1999.

[8] A.-G. Bosser, *Technologies for E-Learning and Digital Entertainment.* Springer Berlin / Heidelberg, 2006, ch. A Framework to Help Designing Innovative Massively Multiplayer Online Games Interactions, pp. 519 – 528.

[9] J. D. Pellegrino and C. Dovrolis, "Bandwidth requirement and state consistency in three multiplayer game architectures," in *Proceedings of the 2nd ACM SIGCOMM workshop on Network and system support for games (NetGames'03)*, New York, NY, USA, 2003, pp. 52–59.

[10] P. Okanda and G. Blair, "OpenPING: a reflective middleware for the construction of adaptive networked game applications," in *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games.* New York, NY, USA: ACM, 2004, pp. 111–115.