M^{2}etis: An adaptable Publish/Subscribe System for MMVEs based on Event Semantics

Thomas Fischer, Johannes Held, Richard Lenz

{thomas.fischer, richard.lenz}@cs.fau.de,
mail@johannesheld.net

Abstract: Massive Multiuser Virtual Environments (MMVEs) have grown to a huge industry in recent years. They allow for simultaneous activity of thousands of players in a virtual world. Distributed event-based systems are a promising approach to reach both performing and scalable architectures. The potential of these architectures can only be fully exploited if event semantics are used to optimize event handling. In this paper we introduce the ***M****assive* ***M****ultiuser* ***E****vent* ***I****nfra****S****tructure* (M^{2}etis), an adaptable and optimizable publish/subscribe system, which is capable of channel-wise multidimensional optimization. We describe the arising challenges and propose a possible solution to the design of such a publish/subscribe system.

1 Introduction

Internet scale distributed applications have risen to industrial significance over the last years. One of the most growing markets in this area are virtual environments in general and *Massive Multiuser Virtual Environment*s (MMVEs) in particular. This promising market states some unique challenges for designers, software architects and especially researchers.

MMVEs define a distributed virtual world shared by thousands of participants, each represented by an avatar, who compete and cooperate in one shared persistent world. This world may be for entertainment as in *Massive Multiplayer Online Game*s (MMOGs) or e.g. a large scale simulation. The design of a dynamically evolving virtual world requires armies of artists and the software backing a world of such enormous dimensions must satisfy several hard to achieve requirements, e.g. consistency, availability, persistence and interactivity [FDI$^+$10]. But scalability is the main challenge of MMVE architectures: Interaction quality must be kept high, even if the number of users and participating sites increases drastically. To address this challenge highly adaptable and optimizable distributed architectures are required, which are capable of dealing with increasing numbers of events of various types.

The M^{2}etis project aims for a holistic approach of utilizing a comprehensive set of different *semantic* properties of events that may arise in an MMVE to reduce the communication effort between clients. Our goal is to develop a generally applicable infrastructure for event propagation which is capable of treating each event type in a specifically adapted way, according to its semantic properties. This enables much larger potential for optimization than the uniform treatment of events in existing architectures. In order to realize such an architecture, our approach follows a channel based publish/subscribe paradigm, as many existing distributed virtual environments communicate using this paradigm.

The underlying model for event classification is based on a multidimensional classification schema for event types, which has been described in [FDI$^+$10]. This initial classification schema has been derived by analyzing the characteristics and the potential for optimization of typical event types of MMVEs. In this paper we exemplify this classification schema, and we introduce an architecture for an adaptable publish/subscribe system which supports a channel based optimization of event dissemination by exploiting the semantic properties of the corresponding event types.

The paper is organized as follows: First in section 2 we give a short introduction into the domain of MMVEs and explain the basic problem, the M^{2}etis architecture adresses. Afterwards in section 3 we describe our notion of event semantics followed by their exploitation in the M^{2}etis architecture described in 4. In section 5 we describe the prototypic implementation as a proof of concept and conclude afterwards in section 7.

2 Massive Multiuser Virtual Environments

Current industry strength MMVE architectures favour a client/server architectural style. To cope with more and more players, tremendous effort is invested by the usage of grid approaches like in Linden Lab's *Second Life* or the deployment of large hierarchical clusters like for the operation of CCP's *Eve Online*. In general, an MMVE consists of n interconnected nodes all being part of the same virtual world. This world is described by a global shared state, which is the sum of the states of all entities existing in the MMVE. Each node manages an as consistent as possible replica of this global state and runs a game engine. This game engine is responsible for the manipulation and the generation of a view on the world state.

Assuming the usage of a broadcast mechanism to keep the (distributed or centralized) world state consistent and that in an ideal case, all events are securely delivered and processed in the same order nearly at the same time on all clients, requires an effort of $O(n^2)$. It is obvious that this dissemination strategy does not scale well. Multiplayer games like *Quake 3 Arena* have shown that without any optimizations, the limit is reached by about 64 clients, depending on the required update rate of the game.

Enabling scalability beyond that limit states a challenge. Many recent event-dissemination architectures for MMVEs [HCC06, BDL+08, BPS06, ZKD08, Fer08] optimize the event dissemination based on one basic idea: Exploitation of event semantics to reduce message exchange between clients.

3 Event Semantics

Events used in MMVEs feature many characteristics like for example a spatial context in which the event is valid that may be exploited to develop specialized dissemination algorithms. But first, when talking about event semantics, we have to distinguish between event types and events, which means for example "item pickup" is an event type and "player x picks up flower y at position z" is an corresponding event.

Pickup events for example have a spatial context, persistent effects and some form of synchronization to ensure only one player can pick up a certain flower. Spatial context means those events are only relevant for a limited subspace of the virtual world, therefore special multicast algorithms, which exploit spatial neighbourhood may be used to distribute those events. Persistent effect means that the picked up flower is added to an inventory of the character which has to survive session boundaries. Moreover there are other dimensions like validity, delivery and security, which all enable another dimension of optimization possibilities.

Existing approaches concentrate their efforts on event types which occur with high frequency, mainly position updates. They tailor their architectures towards those events, without considering other semantic aspects of event types required by MMVEs. We strive for a more generalized approach which incorporates all relevant event types and provide the best possible dissemination for each special type. The semantics of each event type, which is exchanged between clients of a virtual world, may be analysed and semantically described. Therefore we introduced a multidimensional classification in [FL10], where we identified orthogonal dimensions to model independent semantic properties for event types. We defined disjoint characteristic classes for each dimension. To classify an event type, it has to be assigned to one class along each dimension. Based on this scheme, the class of an event type is defined as the sum of the characteristics along each dimension. The power of such a multidimensional class space enables optimization of each event type along each dimension with a different strategy in order to gain a better overall optimization footprint of the system, than if a fixed strategy is used for the whole system. We exhaustively discussed an initial set of dimensions with their corresponding classes in [FDI+10].

The flower pickup example shows the variety of optimization potential each event type has and indicates the adequateness of our multidimensional approach. The concrete classifications in an application may vary in detail, for example depending on the intended tolerance of the consistency in the virtual world. The consistency is mainly controlled by the synchronization semantics, but also influenced by physical

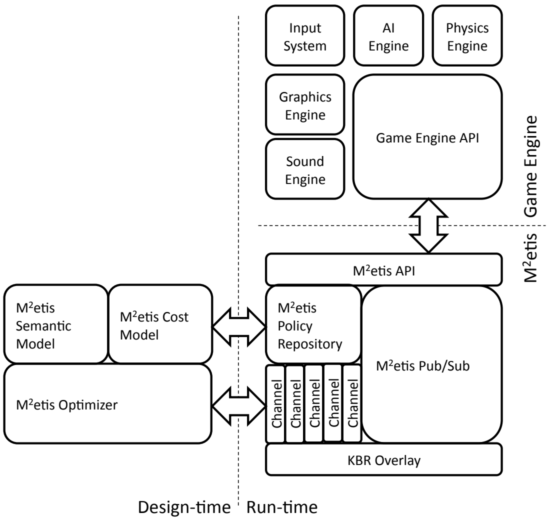

Figure 1: M^{2}etis components

constraints like network latency. This makes it even more important to provide different strategies optimized to the special requirements of each event type. Based on this multidimensional notion of event semantics a publish/subscribe system may be designed which is adaptable and optimizable to a variety of different processing requirements as we discuss in the next section.

4 The M^{2}etis Architecture

To our knowledge, all existing optimization approaches address only one or two aspects of event semantics and propose an optimized event-dissemination architecture for that specialized aspect. (cf. [HCC06, BDL$^+$08, BPS06, ZKD08, Fer08]) Moreover, typically all event types are treated in the same way, despite their different semantical properties. The M^{2}etis project incorporates those existing approaches in a uniform framework which considers each event type alone according to its semantic properties and the resulting optimization capabilities.

The realization of such an architecture supporting adaptable optimizations states many challenges. Singhal and Zyda [SZ99] identify the main technical challenge for networked virtual environments as the management of their dynamic shared state. We use a channel based publish/subscribe approach to exchange state change information, as it is versatile and fulfils all requirements for as identified by Singhal and Zyda. M^{2}etis provides a networking middleware for MMVEs which integrates seamlessly into the reference architecture for game engines (cf. Fig. 1). One of the core benefits of this approach is the separation of the game engine and the message dissemination architecture, which reduces the complexity in the game engine significantly. It is designed around the central multidimensional semantic event classification, which makes the framework adaptable to multiple scenarios. Each application has to provide a semantic classification of its event types according to our schema. Based on the definition of the type itself and its semantic properties the M^{2}etis system decides which optimization strategy is chosen to disseminate each event type across the participating nodes.

Fig. 1 shows the components of our architecture. The core of a node is its Pub/Sub component. It manages one channel for each event type. The different nodes communicate via a *key-based routing* (KBR) overlay network [DZD$^+$03]. For each dimension a channel may be optimized along, the policy repository holds different optimization strategies. As all required channels are known at design-time, a channel is constructed at compile-time by the usage of applicable strategies from the policy repository. The decision which strategy fits best is made based on a cost model and the according semantic requirements. As this decision is made at compile-time expensive optimization calculations do not hamper run-time performance.

We aim for two inputs to deduce the optimized publish/subscribe channels: A semantic model of the event types and a description of the parameters the cost model should optimize upon. Therefore we identified two major challenges. On the one hand the integration of numerous optimization approaches with their characteristic costs and on the other hand the design of the cost model and the optimizer itself. These are beyond the scope of this paper as we first have to show a solution for the architectural challenge of an adaptable publish/subscribe component fulfilling our requirements, which proofs the applicability of the M^{2}etis approach.

To do this we have to introduce a corresponding processing model of our publish subscribe component describing the processing behaviour at run-time, in order to enable the multidimensional optimization of each single channel.

4.1 Processing Model

The processing model describes how the publish/subscribe system abstracts from the KBR network, taking into consideration the different dimensions to optimize the event dissemination.

Each channel offers the common publish/subscribe actions: *subscribe, unsubscribe* and *publish.* Each action for each channel is connected with a corresponding type of message sent via the network. The network is accessed via a KBR-API providing three methods to inform about new nodes joining the network, a message being forwarded and/or delivered to the user's node [DZD$^+$03].

The semantical dimensions introduced in [FDI$^+$10] are covered by seven policies. These policies define the interfaces for the implementation of different strategies for each dimension and their implication on the processing of the publish/subscribe messages. We support the following policies:

Routing is the logic of event dissemination and creates a multicast-tree.

Filter permits to attach a filter predicate to subscribtions and ensures that these predicates are merged upwards in the multicast-tree to filter messages as early as possible.

Delivery defines the logic of the message delivery, e.g. acknowledgements.

Order defines the synchronization strategy for a channel.

Persistence provides persistence for messages delivered to the target nodes.

Security may be used to encrypt a channel.

Validity discards invalid messages and therefore decrease the amount of messages.

The mapping between publish/subscribe and KBR actions is straight forward as table 1 shows. All policies are involved in the delivery of publish messages. This ensures that these messages arrive at the destination host and are not changed while routing. Subscribe and unsubscribe messages require actions in forward and deliver for routing and filtering purposes, as the decision of their strategies may affect the structure of the multicast-tree.

Message-	KBR-	Policy for every channel						
type	method	Routing	Filter	Delivery	Order	Persistence	Security	Validity
publish	deliver	+	+	+	+	+	+	+
subscribe	deliver	+	+				+	
	forward	+	+				+	
unsubscribe	deliver	+	+				+	
	forward	+	+				+	

Table 1: Matrix for policy application in the processing model

Based on the given mapping, the challenge is to find a generic application order for the policies in the processing model as shown in figure 2.

For the distribution of a message the routing policy has to provide a list of target nodes, to whom the message must be send. Every multicast-tree has one or more corresponding root nodes. Subscribe or unsubscribe messages are always sent to that nodes, while publish messages from a non-root node must always be sent to the root nodes to trigger the publish process. Such messages are flagged *to root*. On the root nodes, this flag is set to *from root*. Nodes sending messages as *from root* return the list of subscribers as target nodes which are filtered according to their predicate given at subscription. Finally the message is encrypted and handed over to the network.

Decryption is the first action if a message is processed in deliver. After that subscribe and unsubscribe messages are handled separately. These messages are passed to the strategies for routing and filter to alter the multicast tree. Publish messages on the other hand have to pass the validity check. The next step checks whether the message must be spread i.e. distributed to other nodes. That is the case if the

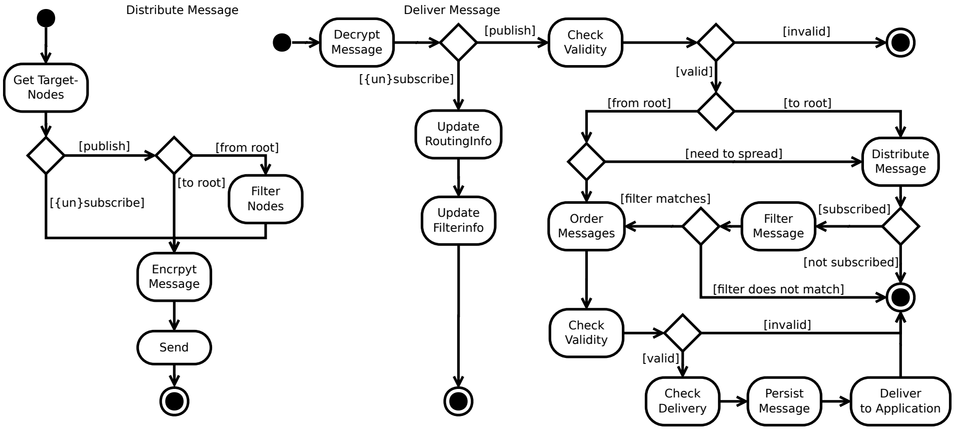

Figure 2: Sending and delivery of a message

actual node is a root node or an intermediate node with distribution responsibility. If yes, the message is distributed using the send-process explained above. Now, the subscription and the predicate of the actual node are checked to ensure a proper delivery if that node is subscribed, too. If the actual node is a simple subscriber and the message is flagged *from root* the predicate is not tested, as it is already checked while sending the message at the publishers side. The two paths merge at the ordering step. As ordering may hold messages back until they are ready the validity has to be checked again. The deliver dimension is called next offering to send acknowledgements back to the sender. Persisting the message is the last action before the message is finally delivered up to the application.
The handling in the forward-action from the KBR is similar to deliver as the subscribe or unsubscribe information is extracted and passed to the strategies and they may alter the message.

As failures are not detected activeley in such networks, it is important to automatically renew the subscription. If nodes in the logical multicast-tree fail, the messages are automatically routed via other hosts, rebuilding the tree. However, it is possible that some messages are lost during the recovery. Optimization strategies have to compensate this if required.

4.2 Optimization Strategies

For every policy M^{2}etis provides one ore more strategies implementing an optimized behaviour. For example distributed *multicast* algorithms for routing. The framework permits inclusion of user-defined strategies following the required behaviour of the policy. Different channels use a different strategy for every policy, creating a variety of optimization options. Each implemented strategy must provide additional cost information that is used in the optimizing step and may add information onto each message header to enable customized treatment.

Using the example of Scribe [CDKR02] and VON [HCC06] some details of the processing model showed in figure 2 are explained. Scribe creates a multicast-tree trying to minimize the amount of messages while VON is neighbourhood centric algorithm based on voronoi diagrams.

Using Scribe for routing, *Get Targetnodes* returns either the calculated root node for the channel or the list of subscribed nodes to which a publication must be forwarded. On the other hand a routing strategy like VON returns the in-game neighbours, obtained through application-level knowledge, as each node subscribes at his neighbours in the virtual world.

Scribe processes unsubscribe and subscribe messages in forward. Each node on the routing path adds the sender to its list of subscribers and changes the message to create the multicast-tree. The periodic re-subscription is triggered externally for subscribed nodes as the channel will call its own subscribe method again or internally for intermediate routing hops as the algorithm will resend its subscriptions if other nodes down in the multicast-tree will refresh their subscription. It is necessary that the filter strategy is always involved to ensure the correct merge of predicates upwards the logical multicast-tree.

VON on the other hand does not need automatic periodic subscriptions, because each node will unsubscribe and subscribe frequently. Using the example of position updates it is obvious that each node and its neighbours move often and need to alter their subscriptions each frame in the game.

This exemplary discussion of two different strategies indicates the generic nature of our processing model in terms of a multidimensional optimization for publish/subscribe channels.

5 Prototypic Implementation of the Publish/Subscribe Component

Our prototypic implementation shows the applicability of this multidimensional optimization approach to a channel based publish/subscribe system. We chose C++ as language of choice and Chimera [AA06], a KBR-overlay as basis. In order to minimize the overhead introduced by the flexibility of such a framework e.g. on message size or stack depth caused by function calls, we use *template metaprogramming* (TMP) and *policy based*-design [Ale01] to create the optimized channels at compile-time. The different optimization strategies for each dimension are implemented as policies encapsulating their behaviour. Each channel is therefore a template class with all dimensions as template parameters, which are instantiated with strategies for each parameter.

With TMP it is possible to derive custom-tailored message headers, depending on the chosen strategies. This ensures small message sizes with a high payload ratio. Each channel itself is therefore in charge to orchestrate strategies for the different policies with regard to our processing model, derived by the semantic description for all optimization dimensions. All described design decisions ensure that the system has a small footprint at runtime and that it can be used without further knowledge of the system internals, the channels or the used strategies.

6 Further Work

Based on the introduced publish/subscribe system further work must be done to enable self optimization in M^{2}etis as planned. A cost model has to be developed in order to decide which optimization strategy is optimal for different event semantics. Currently we are adapting different existing optimization algorithms for all dimensions to our processing model. Based on this variety of algorithms, measurements will be taken to derive a suitable cost model for optimization, with a simulator based on OverSim [BHK07] currently under development. Another part of our ongoing research is the construction of a *domain-specific language* (DSL) designed for the semantic description of event types and the overall system with their optimization targets.

7 Conclusion

In this paper we introduced the M^{2}etis architecture, an adaptable publish/subscribe system for event dissemination. The approach aims for the exploitation of event semantics to optimize event channels individually. We described the design of the core publish/subscribe component of M^{2}etis along with its processing model, enabling for a channel-wise multidimensional optimization. The novelty of this approach is the consideration of a wide range of semantic aspects for optimization of the message dissemination. The proposed processing model supports in the current version as many as seven dimensions for optimization, but there is still much work required for the development of the optimization model, as this work is still in a preliminary stage. However, the prototype implementation showed that the general approach is feasible. Evaluations are planned to quantify the optimization effects for typical application scenarios.

References

[AA06] Matthew S. Allen and Rame Alebouyeh. Chimera: A Library for Structured Peer-to-peer Application Development. Technical report, University of California, Santa Barbara, 2006.

[Ale01] Andrei Alexandrescu. *Modern C++ Design Generic Programming and Desgin Patterns Applied*. Addison Wesley, 2001.

[BDL$^+$08] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. *SIGCOMM Comput. Commun. Rev. (CCR)*, 38(4):389–400, 2008.

[BHK07] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07), Anchorage, AK, USA*, pages 79–84, May 2007.

[BPS06] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. In *3rd Symposium on Networked Systems Design & Implementation (NSDI)*, pages 155–168, Berkeley, CA, USA, 2006. USENIX Association.

[CDKR02] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[DZD$^+$03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 33–44–44. Springer Berlin / Heidelberg, 2003.

[FDI$^+$10] Thomas Fischer, Michael Daum, Florian Irmert, Neumann Christoph P., and Richard Lenz. Exploitation of Event-Semantics for Distributed Publish/Subscribe Systems in Massively Multiuser Virtual Environments. In *Proceedings of the 2010 international Symposium on Database Engineering & Applications (IDEAS '10)*, Montreal, QC, Canada, 2010.

[Fer08] Stefano Ferretti. A Synchronization Protocol For Supporting Peer-to-Peer Multiplayer Online Games in Overlay Networks. In *2nd International Conference on Distributed Event-Based Systems (DEBS)*, pages 83–94, New York, NY, USA, 2008. ACM.

[FL10] Thomas Fischer and Richard Lenz. Event semantics in event dissemination architectures for massive multiuser virtual environments. In *DEBS '10: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 93–94, New York, NY, USA, 2010. ACM.

[HCC06] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *IEEE Network*, 20(4):22–31, July 2006.

[SZ99] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley Professional, New York, NY, USA, 1999.

[ZKD08] Kaiwen Zhang, Bettina Kemme, and Alexandre Denault. Persistence in Massively Multiplayer Online Games. In *7th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*, pages 53–58, New York, NY, USA, 2008. ACM.