Context-Aware 3D rendering for User-Centric Pervasive Collaborative computing environments

Nijdam, Niels Alexander

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique — Professeur José Rolim

FACULTÉ DES SCIENCES
ÉCONOMIQUES ET SOCIALES

Département de systèmes d'information — Professeur Nadia Magnenat-Thalmann

Context-Aware 3D Rendering for User-Centric Pervasive Collaborative Computing Environments

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Niels Alexander NIJDAM

de

Deurne (Pays-Bas)

Thèse N°4631

GENÈVE

Atelier d'impression Repromail

2014

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Doctorat ès sciences
Mention informatique

Thèse de *Monsieur Niels Alexander NIJDAM*

intitulée :

" Context-Aware 3D Rendering for User-Centric Pervasive Collaborative Computing Environments "

La Faculté des sciences, sur le préavis de Madame N. MAGNENAT THALMANN, professeure honoraire et directrice de thèse (Faculté des sciences économiques et sociales, MIRALab), de Messieurs J. ROLIM, professeur ordinaire (Département d'informatique), F.-E. WOLTER, professeur (Welfenlab, Leibniz University of Hannover, Germany) et M. PREDA, docteur (Institut MINES Telecom, Paris, France), autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 7 janvier 2014

Thèse - 4631 -

Le Doyen, Jean-Marc TRISCONE

N.B.- La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

Acknowledgements

I would like to thank my supervisor Prof. Nadia Magnenat-Thalmann for granting me this unique opportunity at MIRALab. Her guidance, insights and support throughout this PhD were vital during these years, for which I'm very grateful. It allowed me to further my knowledge in new directions and hone my skills in computer-science. I also want to thank the members of the jury for their time and effort in reviewing my manuscript; Prof. Jose Rolim (University of Geneva, Switzerland), Prof. Franz-Erich Wolter (Leibniz University of Hannover, Germany) and Prof. Marius Preda (Artemis Telecom SudParis, France).

A huge thanks to all my colleagues at MIRALab, past and present, for their collaboration and generally having a great time. In particular many thanks to Mingyu Lim and Seunghyun Han for their guidance and support they provided for my topic and collaboration. A "space filling" thanks to Bart Kevelham for some really great collaboration. Thanks to all to make this a memorable and enjoyable stay: Maher Ben Moussa, Yvain Tisserand, Ugo Bonanni, Mustafa Kasap, Yeonha An, Nedjma Cadi, Marlène Arévalo-Poizat, Zerrin Yumak, Alessandro Foni, Sylvain Chagué, Etienne Lyard, Ghislaine Wharton. For the collaboration and discussions with the former medical group, Jérôme Schmid, Caecilia Charbonnier, Lazhari Assassi and Jinman Kim; And current, Matthias Becker, Andra Chincisan and Hon Fai Choi.

The presented work was supported by several projects. From the European research 6th Framework Programme InterMedia and Leapfrog. From the 7th Framework Programme the Servive and last the European Marie Curie RTN project 3D Anatomical Human. These projects enabled me to broaden my topic and furthered the applicability of my work. I would like to thank all the people for their collaboration and their hospitality during exchanges. In particular I would like to thank, Xavier Righetti and Riccardo Rapuzzi.

To my father, with his unconditional support, I thank him from the bottom of my heart. My brother who has always been an example to me. My daughter my greatest joy in life. But most of all, the one person who made this PhD truly possible, my beloved wife Anastasia.

Niels A. Nijdam

Abstract

In the last decade technology has improved rapidly, in terms of computing power and mobility, and has changed the way how we interact with media in general. Mobile devices are capable of performing decent 3D renderings and with the increase in wireless communications these devices are centralized portals to a wealth of shared data always available on demand. Albeit the progress, most social media is still reduced to exchange of textual messages, video/audio *buffered* streaming and documents that need to be downloaded first. Especially when we focus on 3D simulations we see that this barely exists let alone in a collaborative manner. For mobile devices especially the games are showing the current 3D capabilities, but are often greatly compressed and all kinds of tricks are utilized in order to provide a balance between frame rate and image quality. This is logical since a mobile device is still limited and in terms of computational power greatly lacking in comparison to a stationary machine. With a focus on complex 3D virtual environments and collaborative aspects it is impractical not only to manually copy 3D content from one device to another whenever a user decides to switch from device or to share the environment with others, but also to render the complex 3D data locally on resource-limited devices such as mobile phones and tablets. The problem becomes more apparent when running a 3D virtual environment that is driven by a complex simulation as often this is tightly coupled, e.g. deformation of a 3D model. In addition the simulation might have several dependencies such as compiler output for a specific platform and hardware, e.g. when using some computing language such as OpenCL or CUDA. Depending on the simulation and rendering technology used, not only mobile devices but also regular workstations can be overwhelmed and become unusable (often leading to higher costs in hardware).

In order to overcome the limitations for such devices we are looking at remote solutions, specifically for 3D virtual environments, involving one or more simulation driven 3D entities and in addition provide support for collaborative aspects. We strive to enabling user-centric pervasive computing environments where users can utilize nearby heterogeneous devices any-time and anywhere. Providing cloud-like services to which one or more users can connect and interact directly with the 3D environment, without the burden of any direct dependencies of the provided service.

We propose a context-aware adaptive rendering architecture which visualizes 3D content with customized user interfaces, dynamically adapting to current device contexts such as processing power, memory size, display size, and network condition at runtime, while preserving the interactive performance of the 3D content. To increase the responsiveness of remote 3D render-

ing, we use a mechanism which temporally adjusts the quality of visualization, adapting to the current device context. By adapting the quality of visualization in terms of image quality, the overall responsiveness and frame-rate are maintained no matter the resource status. In order to overcome inevitable physical limitations of display capabilities and input controls on client devices, we provide a user interface adaptation mechanism, utilizing an interface markup language, which dynamically binds operations provided by the 3D application and user interfaces with predefined device and application profiles.

A generalized data sharing mechanism based on publish/subscribe methodology is used to handle data between service-service and service-user allowing for peer to peer or server-client structured communications. Providing easy binding to different kinds of data models that need to be synchronized, using a multi-threaded approach for local and remote updates to the data model. An extra layer between the sharing of the data and the local data is applied to provide conversions and update on demand capabilities (e.g. a data model in Graphics memory needs to be pre processed to adhere to the constraints of the rendering pipeline).

The framework is divided into several layers and relies on the Presentation Semantics Split Application Model, providing distinct layers with each clear defined functionalities. The functionalities from each layer are isolated and encapsulated into a uniform model for implementation, resulting into a single minimalistic kernel that can execute at runtime the requested functionalities, which are called nodes. These nodes then can be concatenated in parent/child relationships, or be executed as stand-alone processes, providing easy deployment and scalability.

The context-aware adaptive rendering framework is used in several use-case scenarios based on different domains, User Centric Media, Telemedicine and E-Commerce. User Centric Media focuses on the adaptation rendering and support for heterogeneous devices. Telemedicine has a focus on collaboration and access to a diverse set of data, 3D as well as non 3D data such as 2D extracts from volumetric MRI data. E-Commerce explores the possibilities for augmented reality on mobile devices as a service and overall deployment of 3D rendering services with user management.

Résumé

Durant la dernière décennie, la technologie s'est améliorée rapidement en termes de puissance de calcul et de mobilité. Cela a changé la façon dont nous interagissons avec les médias en général. Les appareils mobiles sont capables d'effectuer des rendus 3D décent et avec l'augmentation des communications sans fil, ces dispositifs sont des portails centralisés où une multitude de données partagées sont toujours disponibles sur demande. Malgré le progrès, la plupart des médias sociaux sont encore réduits à l'échange de messages texte, vidéo ou audio sous forme de flux à mémoire tampon, les documents doivent eux être téléchargés avant l'utilisation. En ce qui concerne les simulations 3D, nous voyons que cela existe très peu et encore moins de manière collaborative. Pour les appareils mobiles, les jeux montrent les capacités 3D actuelles mais ils sont souvent fortement compressées et toutes sortes d'artifices sont utilisés afin d'assurer un équilibre entre le taux de rafraichissement et la qualité d'image. Cela est logique car un appareil mobile est encore très limité en termes de puissance de calcul comparé à un poste de travail. En mettant l'accent sur les environnements virtuels 3D complexes et les aspects collaboratifs, il est impossible non seulement de copier manuellement le contenu 3D d'un appareil à un autre lorsque l'utilisateur décide de changer de dispositif, de partager l'environnement avec d'autres personnes, mais aussi d'effectuer le rendu de données 3D complexes localement sur les machines à ressources limitées tels que les téléphones mobiles ou les tablettes. Le problème devient plus évident lorsque l'exécution d'un environnement virtuel en 3D est générée par une simulation complexe, par exemple lors la déformation d'un modèle 3D. De plus, la simulation peut avoir plusieurs dépendances telles que la production de compilateur pour une plate-forme ou du matériel spécifique, par exemple lors de l'utilisation d'un langage informatique comme OpenCL ou CUDA. Selon la simulation et la technologie de rendu utilisé, non seulement les appareils mobiles, mais aussi les postes de travail fixes peuvent être dépassés et devenir inutilisable (conduisant souvent à des coÃ»ts plus élevés dans le matériel).

Afin de surmonter les limites de ces dispositifs, nous étudions des solutions à distance, en particulier pour les environnements virtuels 3D, impliquant une ou plusieurs simulations 3D et pouvant fournir un support pour les aspects collaboratifs. Nous nous efforçons de permettre l'utilisation d'environnements informatiques étendues, orientés utilisateurs où les utilisateurs peuvent utiliser des dispositifs hétérogènes en tout temps et n'importe où. Nous voulons fournir des services dans le nuage pour lesquelles un ou plusieurs utilisateurs peuvent se connecter et interagir directement avec l'environnement 3D, sans les problèmes dues aux dépendances directes du service fourni.

Nous proposons un système de rendu adaptatif sensible au contexte, permettant de visualiser le contenu 3D avec des interfaces utilisateurs personnalisées, adaptées dynamiquement aux contextes de l'appareil tels que la puissance de calcul, la quantité de mémoire, la taille de l'écran ou encore l'état du réseau lors de l'exécution, tout cela en préservant la performance interactive du contenu 3D. Pour accroître la réactivité à distance du rendu 3D, nous utilisons un mécanisme qui ajuste en temps réel la qualité de la visualisation, en l'adaptant au contexte de l'appareil. En adaptant la qualité de la visualisation en termes de qualité d'image, la réactivité globale et la vitesse de rafraichissement sont maintenus, peu importe le statut de la ressource. Afin de surmonter les limites physiques inévitables de capacité d'affichage et les contrôles d'entrée sur les machines clientes, nous fournissons un mécanisme d'adaptation d'interface utilisateur, basé sur un langage de balisage de l'interface qui lie dynamiquement les opérations prévues par l'application avec les interfaces 3D et l'utilisateur, comprenant des profils de périphériques et d'applications prédéfinies. Un mécanisme de partage de données généralisé fondé sur la méthode publication/abonnement est utilisé pour traiter les données entre le service-service et le service-utilisateur permettant une communication structurées de type pair à pair ou client-serveur. Nous voulons fournir facilement une liaison entre différents types de modèles de données qui doivent être synchronisés, en utilisant une approche parallélisée pour mettre à jour un modèle de données en locale et à distance. Une couche supplémentaire entre les données partagées et les données locales est appliquée pour fournir des conversions et mettre à jour les capacités à la demande (par exemple, un modèle de données dans la mémoire graphique doit être prétraité afin d'adhérer aux contraintes du pipeline de rendu).

Le système est divisé en plusieurs couches et s'appuie sur la *Presentation Semantics Split Application* fournissant des couches distinctes contenants chaque fonctionnalité d'une manière clairement définie. Les fonctionnalités de chaque couche sont isolés et encapsulés dans un modèle uniforme pour l'implémentation, le résultat est un noyau minimaliste, appelé nœud, qui permet d'exécuter les fonctionnalités demandées. Ces nœuds peuvent être concaténés dans une relation parent / enfant ou être exécutés comme des processus autonomes, fournissant un déploiement et une évolutivité facile.

Le système de rendu adaptatif lié au contexte est utilisé dans plusieurs scénarios ou cas d'utilisation rattachés à des domaines, le *User Centric Media*, la télémédecine et e-commerce. Le *User Centric Media* se concentre sur le rendu adaptatif et le support de périphériques hétérogènes. La télémédecine met l'accent sur la collaboration et l'accès à un ensemble varié de données, des données 3D mais aussi des données 2D comme par exemple des données 2D d'IRM volumétrique. Le e-commerce explore les possibilités de la réalité augmentée sur les appareils mobiles en tant que services ainsi que le déploiement global de services de rendu 3D tout en incluant la gestion des utilisateurs.

List of Publications

Most of the material presented in this thesis previously appeared in the following publications.

Peer-reviewed Journals

- M. Lim, B. Kevelham, N. Nijdam and N. Magnenat-Thalmann, "Rapid Development of Distributed Applications Using High-level Communication Support", *Journal of Network and Computer Applications, Elsevier*, 34, Issue1, pp. 172-182, January 2011.
- S. Han, N. Nijdam, J. Schmid, J. Kim, and N. Magnenat-Thalmann, "Collaborative telemedicine for interactive multiuser segmentation of volumetric medical images", *The Visual Computer Journal, Proc. CGI 2010*, vol. 26, no. 6-8, pp. 639–648, 2010.

Peer-reviewed Conferences

- N. Nijdam, Y. Tisserand, and N. Magnenat-Thalmann, "Refurbish a single user 3D application into a multi-user distributed service - A case study", *VRST 2013 - The 19th symposium on virtual reality software and technology*, to be published.
- N. Nijdam, B. Kevelham, S. Han, and N. Magnenat-Thalmann, "An application framework for adaptive distributed simulation and 3D rendering services", *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry - VRCAI 12*, p. 103, 2012.
- S. Han, N. Nijdam, J. Schmid, J. Kim, and N. Magnenat-Thalmann, "Collaborative telemedicine for interactive multiuser segmentation of volumetric medical images", *The Visual Computer Journal, Springer*, vol. 26, no. 6, pp. 639-648, June 2010.
- N. Nijdam, S. Han, B. Kevelham and N. Magnenat-Thalmann, "A context-aware adaptive rendering system for user-centric pervasive computing environments", *MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, pp. 790-795, April 2010.
- J. Schmid, N. Nijdam, A. Han, J. Kim, and N. Magnenat-Thalmann, "Interactive Segmentation of Volumetric Medical Images for Collaborative Telemedicine", *Modelling the Physiological Human, Proc. 3DPH*, vol. LNCS 5903. Springer, 2009, pp. 13-24.
- N. Magnenat-Thalmann, N. Nijdam, S. Han, and D. Protopsaltou, "InterMedia: Towards Truly User-Centric Convergence of Multimedia", *the 1st International ICST Conference on User Centric Media, Springer*, pp. 3-10, November 2009.
- M. Lim, N. Nijdam, N. Magnenat-Thalmann. "A general collaborative platform for mobile multi-user applications", *13th IEEE International Conference on Emerging Technologies and Fac-*

tory Automation (ETFA2008), IEEE, CD Proceedings (ISBN: 1-4244-1506-3), pp. 1346-1353, September 2008.

Other

- F. Chehimi, N. Nijdam, D. Schmidt, E. Rukzio and N. Magnenat-Thalmann, "An Interactive Table Supporting Mobile Phone Interaction and 3D Content", *UbiComp 2009*, Orlando, FL, USA, 30/09/09 - 3/10/09
- B. Kevelham, N. Nijdam, G. Papagiannakis, M. Lim, N. Magnenat-Thalmann, "Remote Augmented Reality for Virtual Fashion Models", *Workshop on Hyper-media 3D Internet, Geneva, October 14 2008*

Contents

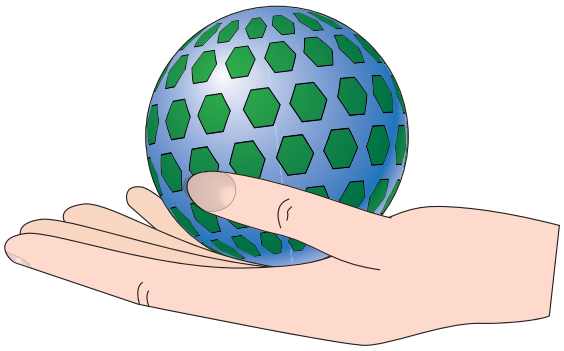
List of Figures

List of Tables

List of Codings

Chapter 1

Introduction



1.1 Motivation

The Internet, a vast and seemingly endless cyberspace connecting millions of people, has become part of our daily life. Over the last decade, the introduction of new consumer hardware has shaped how we use the internet. With more and more content becoming available every day, the internet has an every increasing presence. Through a multitude of different devices, such as e-readers, phones, tablets, desktop systems etc., we are able to exchange data with each other across the world, independent of location, time and the device used. Applications are extended with networking capabilities, either deployed locally on the client device or, as is the trend now, fully hosted on the Internet (servers) as a service (cloud services). Our focus is to bring rich and interactive 3D content to a broad range of end-devices together with the capability of real-time collaboration. The presentation of 3D content varies from application to application since it has to adhere to the functionality and intended usage, this could be a fully static scene with no dynamic aspects in terms of 3D rendering (static 3D models, fixed light models, no interaction) with just a few polygons or a fully dynamic morphing 3D environment with a multitude of 3D entities within it (deformable 3D models, dynamic light approximations, complex interactivity within the 3D scene). We are interested in the latter scenario. While small computing devices are capable of rendering in high detail, supporting shader programs and offering progressively more calculation power, they still have their limitations and will likely not become more powerful than a stationary computer. In traditional way, developing for a mobile platform often comes with a huge set of optimizations to perform, such as lower texture resolution, using higher compression (not even speaking on limitation of supported compression techniques due to different hardware), limited polygon count and shader complexity has to be reduced greatly. Often for light approximations, statically lit scenes with pre-baked *light-maps* are used instead of fully dynamic high dynamic range (HDR) light models, and shadow rendering has to resort to single *shadow mapping* with no depth complexity (e.g. blurring and smoothing according to intensity and angle) or other low computational approximations. The development for a multitude of devices leads to several *code and artistic pipelines*, that have to be maintained and any update in the base data has to be checked thoroughly. We therefore propose an adaptive remote 3D rendering solution that can overcome these limitations, with additionally the support of real-time collaboration. This would lead to a singular point of access, independent of the device, while retaining all normal visuals and interactions. Our focus is less on the streaming and compression of the data itself, but more on the management and handling of the data, managing the synchronization between clients and provide novel methods of adaptation schemes that not only take into account the single attachment of a client, but incorporates the possibility of multi-user and multi-device. Within our focus we cover three major research domains which will be explained and motivated in the following sections. The eldest research domain is *Computer-Supported Cooperative Work (CSCW)*, with its prime focus on collaboration, secondly the domain of *Distributed Virtual Environment (DVE)* which orients around services in a network specifically designed for Virtual Environments (VEs) and last the most recent domain is *adaptive rendering* targeted at providing a more optimized output for visualization. Albeit we handle all three domains, as they are

tightly intertwined with one another, the presented work will show more focus on the adaptive rendering, which we present as *Context Aware Adaptive 3D Rendering (CAAR)*, due to supporting heterogeneous devices the context-awareness and multi-user deployment have a great impact within this particular domain.

Following in this chapter is a short look into the three domain, the objectives of the conducted research and an overview of the several chapters is presented in section 1.4.

1.1.1 In the field of Computer-Supported Cooperative Work

We can easily say that CSCW exists since the first computer network was established (one might argue the term being used[1]) and covers a broad variety of technologies (covering the full Open Systems Interconnection (OSI) model)and research areas. Since then it has come a long way and many practices are now common in many applications. Especially in the area of mobile and ubiquitous devices, all kinds of media are being used and shared everywhere through CSCW based software[2]. The main principle is to have a user-centric multimedia converge, which puts the (mobile) user in the centre. However applications on mobile devices often lack the more complex functionalities as the ones can be seen on desktop systems. Such mobile applications need to be optimized and adapted, and are often dumbed down version of their desktop counterparts, lacking functionality. It is a true challenge to provide the same functionality that is dependent on a collaborative framework, due to previous mentioned short-comings, and with most mobile-phones and tablets having a capacitive touch screen (meaning the input is controlled with a finger instead of a stylus) the user's input is a challenge. Providing on-screen keyboards takes away more display-space, and given that finger based input is far less accurate than a conventional mouse pointer, the interface needs to be adapted to these situations. Yet these devices bring great potential and can be used in a lot of new areas. Where previously mobile devices and networks were barely capable of live streaming data, we have now come to a point where it is feasible to have live-interactive streaming data. Technology wise, especially on the network infrastructure, it will still take years to get a good coverage, but in closed environments (short range, with limited connection capabilities) we see remote rendering being used in products (Nintendo Wii U, Sony Playstation Vita etc.).

The focus in our work lies on the transmission and modification of 3D content and bringing this to a multi-user environment, an example is given in figure 1.1. The targeted environment can be described as a system with three primary features, where "Context-Aware Adaptive 3D Rendering" is the process of providing 3D content to a end-user device in such a manner that it delivers the best possible format for the given device and network connection. The second part has to be taken into two forms, where the first has the focus on device heterogeneity and the second on the collaborative aspects for virtual environments. To elaborate this a bit better the first part can be noted as "User-Centric Pervasive Computing Environments" and aims at the ability to provide service scalability and heterogeneous device support access the system from all kinds of devices and still retain the same functionalities and visuals. Whereas the second part "Collaborative Computing Environments" points out that the system is capable of handling

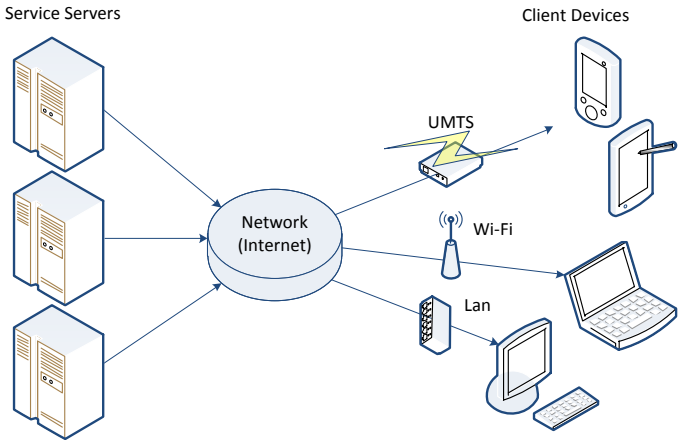


Figure 1.1: The environment for a collaborative environment, with service servers provide a multi-user environment capable of handling a diversity of devices.

multiple users and have them interact in the same user-space which brings forth other topics such as interest management for virtual environments and interaction/event handling. A system with these kinds of capabilities is applicable in many fields, examples are

- Collaborative 3D design work (remote assistance and modelling)
- Educational (remote e-learning, local group-work, virtual classrooms)
- Multi-user gaming environment (Serious games and streamed games in general)
- Medical (Telemedicine, intra-network systems with mobile devices, virtual training)
- On-site analysis (construction sites, city/building plans for repairs, travelling)
- E-Commerce (virtual shops and product presentations/customizations)

1.1.2 3D Content Going Mobile

In the last decade wireless networks provide faster, more stable & reliable and often persistent connections to mobile devices and considerable efforts have been made for nomadic access of multimedia in mobile computing[3]. However more than often this nomadic access limits the users in that it forces them to use their mobile devices as a singular point of access to their multimedia. It is also impractical to manually transmit 3D content from one device to another whenever the user wishes to utilize different devices. Not only does it distract the user in forcing to copy the 3D content but also some resource-limited devices, such as a mobile phone or a Personal Digital Assistant (PDA), may not be capable to render complex 3D data directly on the device itself. This also still leaves out possible simulations that modify the 3D content, which may require high computational power, or are platform specific and as for mobile devices may drain the battery.

To overcome the resource limitations of mobile devices, a data-centric approach[4], with which a client device uses semantically equivalent 3D data depending on different device capabilities,

has been used. However, with this approach there is no guarantee that the consistency of modified 3D content is maintained while a user is switching between different devices. Adaptive remote rendering approaches [5], [6] have been proposed to seamlessly access 3D data with heterogeneous devices. They adaptively generate size of images and frame rate based on the pre-configured client rendering capabilities, display size and network conditions on a dedicated server and stream to the client devices. This, however, decreases the interactive performance of 3D content, not only because it takes a longer response time to get feedbacks if 3D content is fully rendered remotely and streamed back to a client but also because there is no support for adapting the dynamic context changes such as network bandwidth at runtime. Hybrid rendering approaches [7], [8] maximize the utilization of resources on the server-side by delegating the rendering of highly complex 3D scenes to a dedicated server. They can support the client's interaction with 3D content even in the case of temporal network failure. However, with this approach the users are not able to exploit diverse devices and have it adapting to the current context at runtime. The client devices therefore still need to transfer large amounts of 3D data to the server on-demand. Furthermore these approaches do not consider client device capabilities, such as display size and input controls, to represent the user interface. It is impractical to provide complex full-featured controls on resource-limited devices such as Netbook, Ultra Mobile Personal Computer (UMPC), PDA and smart phone devices due to small displays, limited computational power and/or limited connectivity.

With the current trend of storing data in cloud based data centres and more and more ubiquitous devices and interoperability between device types, it is the next logical step towards a truly and fully integrated service platform that combines all of this. It is therefore our vision that a system can provide in a collaborative working environment in a distributed virtual environment, with distributed simulations for rendering 3D content and seamless streamline this into a multi-user environment, whilst being able to adapt to the end-user hardware. Topics regarding this vision include:

- Network Topologies
- Scalability
- Collaborative Virtual Environments
- Interest Management and Event filtering
- Real Time Simulations
- Adaptive 3D Rendering
- Concurrency Control
- Streaming 3D-Content

With an architecture specifically designed for being deployed with a grid/module (cloud) based approach, being itself modular and easily scalable, can offer on-demand services, in our case specifically aimed at 3D virtual environments, which are inserted as plug-ins, can automatically

load-balance themselves accordingly in a set of dedicated servers (which may be located around the world) and support mobile, ubiquitous and stationary devices can greatly improve the way we have access and interact with 3D content nowadays[1]. Utilizing service protocols so it can dynamically fine-tune towards the end-user, in terms of application, device and network capabilities. Providing the capability of dynamically switching devices without losing it's current session and or interactions with the group in a collaborative session. Where we define a service not only as a rendering service but could be a simulation which modifies 3D content, it's output can be shared by multiple users, either independently or in the same user-space.

1.2 Contributions

We propose a context-aware adaptive rendering architecture which visualizes 3D content and an user interface on top of a CVE, while dynamically adapting to the current context such as device availability in the current location and their capabilities, i.e. processing power, memory size, display size, and network condition, while preserving interactive performance of 3D contents. To maintain an optimal responsiveness of remote 3D visualization, we use a scheme-based mechanism, including a device profile, to adjusts quality of visualization adapting to the current device contexts. By adapting the quality of visualization in terms of image quality, the overall responsiveness and frame-rate is maintained no matter the resource status. To overcome the inevitable physical limitations of display and input controls on client devices, an interface adaptation mechanism, based on an interface mark-up language, which dynamically binds operations provided by the 3D application and user interfaces with predefined device and application profiles. A generalized data sharing mechanism based on publish/subscribe methodology is used to handle data between service-service and service-user allowing for peer to peer or server-client structured communications, providing easy binding to different kinds of data models that need to be synchronized, using a multi-threaded approach for local and remote updates to the data model. An extra layer between the sharing of the data and the local data is applied to provide conversions and update on demand capabilities (e.g. a data model in Graphics memory needs to be pre processed to adhere to the constraints of the rendering pipeline). The framework is divided into several layers and relies on the Presentation Semantics Split Application Model (PSSAM), providing distinct layers with each clear defined functionalities. The functionalities from each layer are isolated and encapsulated into a uniform model for implementation, resulting into a single minimalistic kernel that can execute at runtime the requested functionalities, which are called nodes. These nodes then can be concatenated in parent/child relationships, or be executed as stand-alone processes, providing easy deployment and scalability.

With this architecture we cover the three prime technical domains, shown in figure 1.2 presented in a triangular shape complimenting each other. In practise, as will be shown later, the domains are tightly intertwined and it is not always clearly to distinguish in which domain a certain functionality lies. The CSCW handles all multi-user collaborative aspects and influences the DVE in terms of multi-user instances that need to be created in the virtual world (such as a

[1]Project Intermedia section D.1

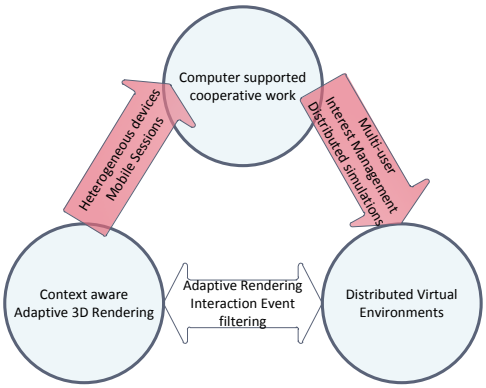


Figure 1.2: The target platform can be divided into three domains, where the lower two domains are extending the upper domain.

virtual camera extending to a fully controllable visualized virtual avatar), providing information about the interest of each user in order to limit the output of the DVE to the most essential data for an individual or a group of users and last handling the interaction with other services simulation services. The DVE provides the simulation and rendering distribution instead of single deployment approaches for scalability and the linking between several instances of the simulation, user representation in the virtual world and filtering of higher level interactions. On top of this is the CAAR providing the rendering layer capable of supporting low-end to high end devices with fine-tuned performance rendering schemes.

1.3 Research Context and Objectives

The main goals are:

- Development of the framework architecture for a platform with support for multiple-users, services and rendering modules.
- Development of a service oriented modular base component using the subscriber/publisher model.
- Development of the client architecture for heterogeneous devices.
- Development of algorithms for handling the interest-management.
- Development of algorithms for handling the adaptive rendering and context switching.

For creating the framework, first its groundwork has to be established (design and implementation). The architecture of the core and base components should be set up in a generic manner, in order to be very versatile towards the services that are build on top of it to provide flexibility and scalability. Yet it should be kept comprehensive in its functionalities to avoid overhead and overly complexity.

The overall architecture elaborates on the support of simulation, rendering and interacting remotely with an 3D virtual environment, while incorporating CVE basic principles, such as sup-

porting multi-user and collaboration. Simulation is the logic of the program that provides the raw data. Rendering is the logic of taking the data and visualize it to its end device-display. The interaction is the influence of the user on the simulation.

Nowadays we have large virtual network, however most of them are only accessible by a limited type of devices. For example a Massive Multi-player game is mostly only accessible by Personal Computer (PC), or by a game-console. More and more software providing similar virtual environments for hand-held devices providing the same functionality. However these are all separated. The deployment of the framework supports a scalable approach, providing the means to distribute any service within a network.

The framework must meet the certain requirements

- Provide service scalability, with the increase in users the service must be able to coop with the increase by being able to balance out the users among its available resources. As the number of users in a DVE increases, a large number of interactions are likely to impose a heavy burden on network and computational resources. One of the key aspects to consider for lessening the burden is scalability, that is, a system should keep interactive performance of users from significantly degrading even with an increase in the number of users.
- Provide efficient event-handling schemes
- Provide heterogeneous device support, by being able to switch between different schemes of rendering types and dynamic interfaces.
- Provide Runtime adaptation depending on the connection quality, the user events and the content being rendered.

In order to realize the architecture several aspects are researched and are explored in several domains of deployment as shown in figure 1.3. We use User Centric Media, Telemedicine and E-Commerce as our target domains. By examining the differences and requirements and design an architecture that can support all of these fields.

All aspects combined form together the “Herd” framework. Which will consist out of a collection of modules coupled together forming the base of the framework, and a set of Application Program Interface (API) and client side specific developed software modules. The main system is developed in C++, targeted at a Windows environment, whereas the client modules target the Java, Android and Windows Phone systems. The framework is described in-depth in section 4.2

In order to cover the domains as shown in figure 1.3, several application scenarios are introduced. The architecture is formed into the “Herd framework” and each application scenario covers several aspects of the goals stated in section 1.3. The scenarios also show a growing aspect of deployment, in that, it extends its technical field with each follow-up.

- The Architecture. The global architecture that enables the envisioned scenarios.
- Simple Remote Rendering for Low-end Devices. The initial approach on the remote 3D rendering and it’s usability.

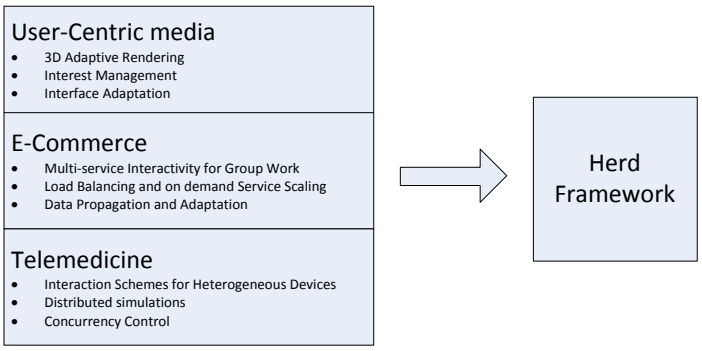


Figure 1.3: From several domains certain aspects of the target framework are realized and combined into one, the Herd framework.

- Remote Rendering with Augmented Reality. Experimenting with Augmented Reality as an extension to remote rendering and simulation.
- Adaptive Rendering with Dynamic Device Switching. Combining technologies, such as mobile IP and Zigbee wireless communication modules for triangular based indoor localization, into a single approach towards an adaptive remote rendering system capable of switching between devices.
- Service Distribution and Render Context Switching. Setting up and maintaining multiple 3D rendering contexts and simulation services for remote adaptive rendering. Using load-balancing schemes and user session management for service control and distribution.
- The Collaborative Magnetic Resonance Imaging (MRI) Segmentation. Providing two types of services, the segmentation and the 3D visualization. Accessible from mobile and stationary devices. Supporting multiple locking mechanisms for experimental purposes.
- Collaborative Services with shared data for simulation and virtual environments. A full scale deployment, prototyping all aspects of the framework. Offering independent user applications with shared data between services and end-user applications.

1.4 Manuscript Organization

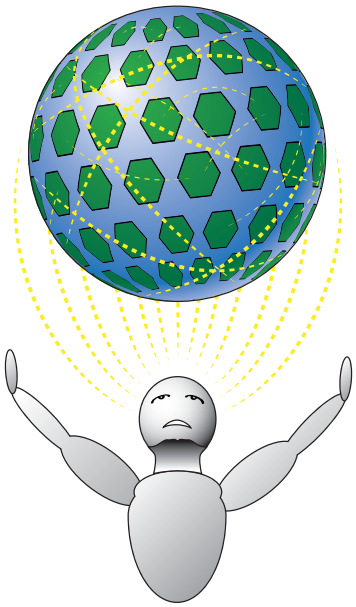
- Chapter 2 Related Work
  The chapter is divided into several sections starting with the introduction into CVE and highlights its aspects. Followed by two sections onto which our framework mainly focuses, Interest Management and 3D Adaptive Rendering. These two aspects will be explained in more detail and related work in the field is presented. Then the whole concept of CVE is related to the three deployment domains as mentioned in section 1.2.
- Chapter 3 Methods and Design
  In this chapter we re-examine what has been said on our contributions and deepen the research goals. As we further examine the individual parts of our proposed architecture methodological level and are described in detail, alongside with similar closely related

research.

- Chapter 4 Technical Design and Implementation
  Where chapter 3 handles the architecture in theoretical manner, this chapter deals with its technical design and implementation. Placing the theory into practice is more than often not as straight forward as we would wish it to be, we therefore highlight the choices on technical design issues and implementation strategies. We make a separation also between the design and implementation, as the design could be platform independent while its implementation might be bound to the use of a certain programming language/compiler, platform and hardware.
- Chapter 5 Experiments and Results
  Along the way of designing and implementing the proposed architecture several experiments were conducted. Early stage proof of concepts were used to elaborate certain aspects of the architecture as it evolved into its final form. Each experiment is described in detail, which entails the deployment domain(User-Centric Media, E-commerce, Telemedicine), the technical aspects, the set-up, execution and results.
- Chapter 6 Conclusion
  The final words concluding our findings on the presented research, the proposed architecture, the established implementation and conducted validation during several stages of its design and implementation.

Chapter 2

Related Work



2.1 Motivation

Albeit the main focus lies on researching new ways on how to bring 3D content to heterogeneous devices, we also look at the parts around it, which overlap with several deployment domains. At the core there is the CVE. The term "Collaborative Virtual Environment",in short CVE, has its roots in the CSCW, which in turn has a long history of research of more than half a century[9] . Albeit the term CSCW back then might have been different, e.g. group support or office automation, in essence it took off in the early 1960. We take a short look at where this area of research came from and which systems have been developed along the way. Collaborative systems exists in many varieties with each their own specific target of deployment[1], [10]–[13]. An overview is given in figure 2.1, where each ring represents a work level. Even if the terms in the figure are outdated it still can be projected to the current day application environment. Where *Organization* is related massive multi-user, *Project* and *small-group* may have become more overlapped or even merged, with globally faster Internet connections and overall standardized means for collaboration and data sharing. Still, a distinction can be made if a target environment is intended for Local Area Network (LAN), e.g. within a company using specialized software, or Wide Area Network (WAN), e.g. public accessible services with collaborative support. This is still reflected on the right side within each ring. Whereas on the left side the main type of development within the given area is depicted. Notably the most important development, with an increase of users, is the human-factors and computer human interaction. This also exist at the smallest level, however the complexity increase with multi-user systems become very challenging. Even the simple working procedures can become quite complicated due to several factors, such as providing consistency among the client-systems, reliability of transmission, interaction responsiveness and interaction dominance (two users at the same time perform interaction which conflict with eachother). The target systems are also still applicable to some extend, e.g. mainframes still exist today but are outperformed by computer clusters (super computers), minicomputers (minis) are no longer existing mainly due to increased performance in desktop computers. The structural deployment stays the same, with an increase in supported user clients the targeted software and hardware gain in complexity and overall higher requirements for development and maintenance.

A distinction can be made in how certain multi-user applications are deployed. The small matrix in figure 2.2 gives an overview with examples of what kind of collaboration is taking place, where horizontal is time and vertical is place. If at the same place and time, the collaboration takes place in a meeting environment, where everybody is together and works together. Then combinations of different but predictable or unpredictable times of collaboration against same, different and predictable or unpredictable places. The category into which a CVE is placed depends on the target application of the CVE and often overlaps (supports) multiple possibilities of time and place. Current popular CVE systems are the Massive Multi-player Online Game (MMOG), in which a persistent virtual world is accessible to users from all over the world connected to the Internet. Using a client application to connect to a server system on which the persistent world is being kept. There is no direct need for collaboration and collaboration can

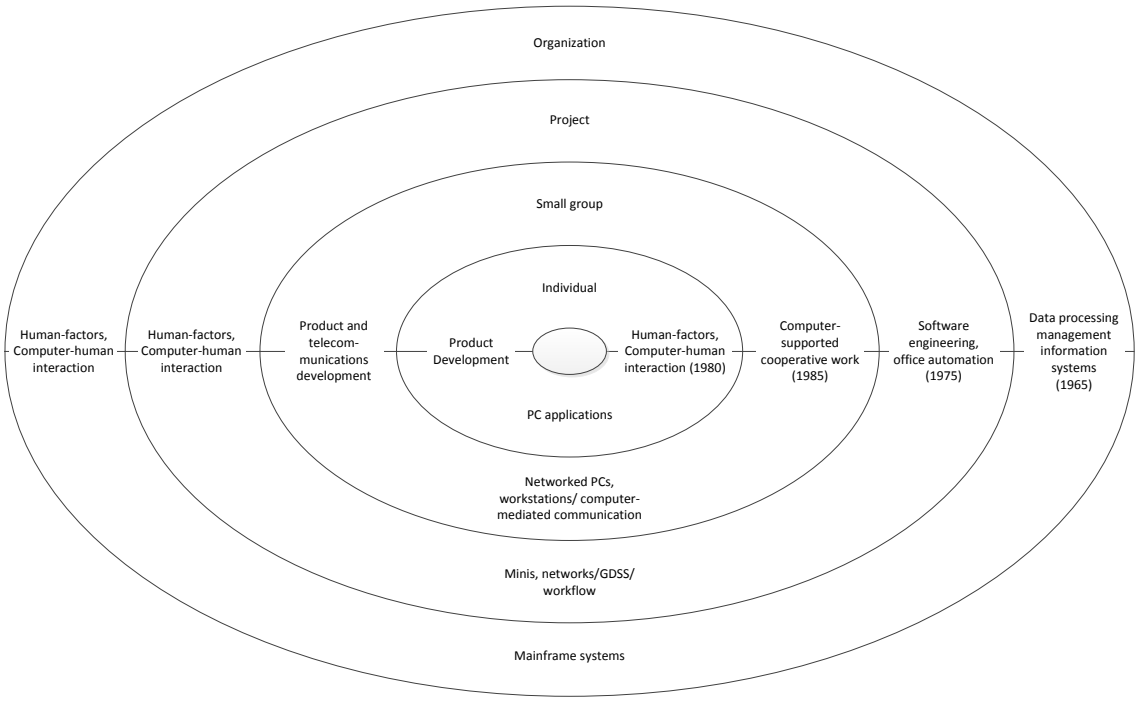


Figure 2.1: Each ring abstracts a level of scale on cooperative work and group-ware.Top the level, below targeted systems, left development and right the research areas [1]

take place in various forms, e.g. simple text chat, audio conversation or meet up with friends by having the virtual avatar (the users representation in the virtual world) all together in one place and perform game or non-game related collaboration. Obviously the main target of a MMOG is to have game-related collaboration, however since it often supports various forms of communication, other kinds of collaboration, outside of the game perspective, may take place. Here it becomes also a bit more complicated since in the real world the time and place may differ, but also in the virtual world the time and place may differ. Albeit, there is no Massive Multi-player Online (MMO) to date in which virtual time-zones exist and therefore is always the same throughout the virtual world. This differs greatly from more specialized CVE, which for example only target the support of virtual meetings, in which it is the aim to have the users representation more expressive and believable, since these are stronger related to user-avatar interaction. Within such an environment there is no virtual time or place and from figure 2.2 this is placed at same time-different and unpredictable place. It seems more constrained, yet this is only because of shifting the focus on what kind of collaboration has to be performed. With a virtual meeting the users need to be tracked, the "speaker" needs to be highlighted, there might be a form of document/media sharing or a common white-board interaction. Eventually these types of virtual environments will merge together as progression in interaction, graphical representation and new collaborative schemes are researched and developed. An example is a Virtual shopping street, where multiple users can roam the streets and enter virtual shops. Within the shops users can discuss products and see 3D representations and annotations about it. There is a flow from Massive multi user environment to an isolated virtual meeting environment. Many MMOG to date already use similar approaches[14], but this is mainly due to game content constraints and

by providing the users in the isolated environment an optimized/stable interactive performance.

	Same	Different but predictable	Different and unpredictable
Same	Meeting facilitation	Work Shifts	Team rooms
Different but predictable	Tele/video Desktop conferencing	Electronic mail	Collaborative writing
Different and unpredictable	Interactive Multicast seminars	Computer Bulletin boards	Workflow

Figure 2.2: "3-by-3 map of group-ware options categorizes representative applications according to place and time."[1]

In this introduction we have shed some light on the origin of collaborative virtual environments and how they are placed within the place and time matrix. A more in depth view is provided in the next section 2.2 "Collaborative Virtual Environments", where the building block of an CVE system are being highlighted and which systems provide full CVE experiences. In section 2.3 and 2.4 two core components of a CVE system are highlighted. Section 2.3 delves into the aspects of "Interest Management" and section 2.4, the main field of advancement in the last few years, and primary focus of this thesis, "Heterogeneous Environments and Adaptive 3D Rendering". The last section "Deployment Domains" 2.5 gives a small overview of the domains mentioned in figure 1.3.

2.2 Collaborative Virtual Environments

Over the past decades many CVE systems have been developed (Appendix B figure B.1) and found their origins in the military simulation [15][16], educational research and entertainment environment. Notably early CVEs are SIMNET[15], Rendezvous[17] and Habitat[18].

> *"Habitat is a multi-participant online virtual environment, a cyberspace."* [19]

An extended list of CVE systems and certain extensions of the CVE systems is given in appendix B "Frameworks". Note that the given list is far from complete as many additions have been made in neighbouring fields such as rendering, compression, telemedicine, human computer interaction, etc, all contributing to the CVE research area. This is because a CVE in itself is a support tool, providing the means to "work together"(figure 2.3) and therefore depends on all kinds of research areas.

The base components for a CVE were established quite early on and have stayed in essence the same over time. This is presented in figure 2.4 as separate "manager" modules. For each user/-client connection a session needs to be established and maintained, therefore a session manager is needed. The whole virtual environment can be seen as users accessing simultaneously a database and only take the information out of it which is of "interest" for them through a Data Manager and visualized to their local client device. The *Interest manager* which incorporates strategies to

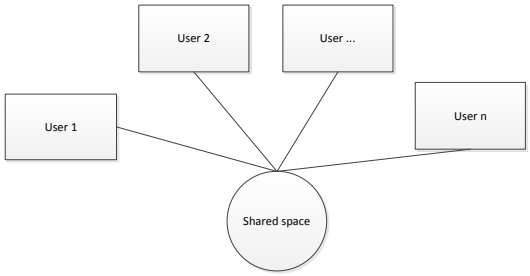


Figure 2.3: Abstract view what a CVE offers.

filter, aggregate and optimize the data generally determines which data is important for the user to have. Since we're dealing with multiple possible interactions and mutations to the data, which is shared among the clients, a Concurrency Manager is needed. At a lower level, events routed between the sender and recipients need to be handled, therefore an Event Manager is needed and at the lowest level there is the Communication Manager for handling all incoming and outgoing traffic (handling different types of events with certain requirements E.g. Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) streams, buffering, packet control etc.).

Figure 2.4: Common CVE Management modules

Where the first CVE used broadcasting but was abandoned fairly quickly due to congestion and later approaches switched to utilize the Multicast principle [20]–[24], which is fine for university and basically any LAN environment, however for normal consumers who are connected to the Internet, *Multicast* is not being routed by most of their Internet Service Providers (ISPs)[25]. Techniques to circumvent this are the use of Multicast protocols, for example Distance Vector Multicast Routing Protocol (DVMRP), Multicast Extensions to OSPF (Open Shortest Path First) (MOSPF), Protocol Independent Multicast - Dense Mode (PIM-DM) and Protocol Independent Multicast - Sparse Mode (PIM-SM). However for these to work the routers between the server and clients still need to be configured to support the protocols or tunnelling. There are also the backbones Abilene, MBone (albeit both are retired)and the current Internet2 Network backbone (United States only), however these are also primarily targeted at universities and are less relevant to heterogeneous environments and consumer connected devices. The way an CVE operates depends heavily on the type of network, as where our focus is the consumer connection over Internet and therefore Multicast becomes more difficult to apply, but not impossible as commercial applications such as "Onlive" [1] and "Gaikai" [2] have proven. However even with the launch of Internet protocol version 6 (IPv6) in 2011 and 2012, [3] (major websites and providers switch permanently to IPv6, while still being backwards compatible with Internet protocol ver-

[1]Onlive web-link `http://www.onlive.com/`
[2]Gaikai web-link `http://www.gaikai.com/`
[3]IPv6 web-link `http://www.worldipv6day.org/` and `http://ipv6.com/`

sion 4 (IPv4), but newer websites may only be available by IPv6), which has Multicast embedded as a standard, the chances are slim that a proper Multicast scheme can be used without striking major deals with ISPs [26] and still have to fall back on the use of tunnels[27]. In figure 2.5 three basic types of routing schemes are given, which have been used by most CVEs. Broadcasting, one to all, is still used for internal LAN environments and mostly for service discovery/announcements, Multicast, one to many, uses the UDP and is favoured for use on video and audio streaming and the last one Unicast, one to one, is the most prominent used method for exchange of data currently on the global Internet infrastructure.

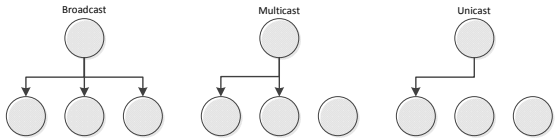


Figure 2.5: Basic communication types, Broadcast[28], Multicast[29] and Unicast[30].

Starting from 1997 newly developed CVE systems started to provide multiple combinations communication architectures to overcome the limitation of Multicast on Internet. Mainly by using Peer to Peer approaches[31]–[35], which are solely Unicast connection between the clients and several strategies have been researched in creating hybrid systems[36]. Figure 2.6 shows common examples of some basic communication architectures. The server-network construction if often used in combination with multicasting. Clients connected to a server may either reside within a LAN or a Virtual Private Network (VPN), where the connection between the servers provides connection to other grouped clients. The database between the server is fully replicated, as this provides the fastest interaction response for the clients within one group. This reduces the amount of data being send over the network and provides less delay for clients, since they are connected to a nearby server. The client-server set-up is possibly the most basic set-up, but is still one of the most used set-ups for Massive Multi-player Games as it provides full control over the data. Whereas Peer-to-peer utilized a shared distributed database among its peers, which provides updates to only those users who might need it and is often used with an Application Layer Multicast Protocol on top[25], [37]. Note that the deployment of these basic communication architectures also depend on the scale, the infrastructure and type of CVE.

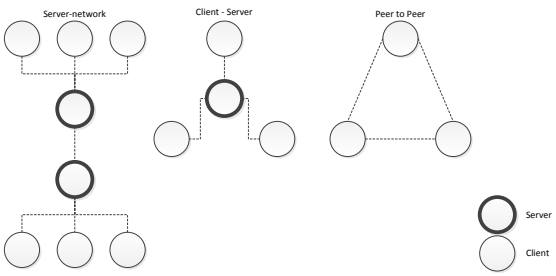


Figure 2.6: Basic network communication architectures.

The main reasons why we are concerned with communication types and architectures are the limitations of network infrastructures and end devices [38]. The Communication Manager can

be seen as the titan Atlas holding up the "virtual" world. Every packet send over the network takes its time to be "put" on the network, to be received by its recipient and processed again. On a single device with a single user everything can be handled easily sequential. Having two users on one device (with each their own input devices) already increases the complexity and the need for managing the input, however on one device the input received would still be instantaneously and can be seen as sequentially. Put a network between the input devices lag and jitter are introduced. Lag is the time it takes for a packet to arrive, often measured by the Round Trip Time (RTT). Whereas jitter is the variation in transmission time due to different routing across the network, packet corruption/loss and resend or network congestion. Network congestion occurs when the network cannot handle the amount of data being send, data would be lost and might need to be resend (e.g. TCP congestion control).

2.3 Interest Management

In every form of information exchange we could state that there is always a form of "Interest Management". From a simple conversation "face to face" we direct our information to the recipient, within networks it works in a similar way. As mentioned before using basic communication types, however these also have their limitations and in order to overcome these communication architectures are used to control these basic communication types. However in complex multi-user networks these systems architectures are limited (often in term of computational power, network bandwidth, network speed) and are unable to keep a consistent fully replicated state of the application, this is especially the case in Large Scale virtual environment (LSVE). The solution to this problem is to "simply" not have a fully replicated state and only keep the "important" sub-states up to date. To determine what is important is dependent on what the client needs and what it is allowed to have. Simply said it is fully dependent on the application. For VE systems this would be the direct environment of where the user resides within the Virtual World (VW). For example, assuming the client has no information at all, except his position in the VW, it will require all the data that are within his sensory range. This can be the geometry of the world (mesh data), textures, sounds (environmental, interaction), dynamic objects, Non-Playable Character (NPC) and other users represented by avatars (animated 3D models, consisting out of mesh data, textures, event sound etc.), all this data does not necessarily has to come from a "resources"-server, but could be included in the client dataset, yet still it needs the information on what, when and where to enable. This is handled by the Interest Manager and is normally placed just above the Event Manager (EM) (2.4). Many systems exist to date that incorporate different strategies on handling information [21], [39]–[47].

> *"The Interest Manager (IM) seeks to reduce the volume of data exchange by exploiting the fact that clients in the system generally have a limited area of the virtual environment with which they interact at any given point in time."* [48]

The most obvious aspect of an VE is that it revolves around one or more virtual worlds and most systems take advantage of that, separating the virtual worlds and dividing a virtual word

into smaller sections [39], [40]. These are called many names, e.g. “Rooms” finding it’s origins from Multi User Dungeon (MUD) applications, Cells [49] and “Regions” are well known from NPSNET [21], SPLINE[23], SCORE[50], ATLAS[34]. This falls under the notion “Region-Based filtering”. Regions that are connected by portals which is based on an older term “door” as used by MUDs.

- Region based Divide the world up into compartments
- Spatial based Sensory of the entity Aura and nimbus
- Class based Parameter based
- Hybrids Most systems employ a mixture of the previously mentioned IM methods

2.4 Heterogeneous Environments and Adaptive 3D Rendering

Several adaptive rendering mechanisms [4]–[8], [51] have been proposed to overcome resource heterogeneity. Krebs et al. [4] proposed a data-centric approach where users are using the same or semantically equivalent 3D data with heterogeneous devices. It is composed of three tiers. The presentation tier contains the controller and view parts of the Model View Controller (MVC) paradigm [52]. The domain tier contains application semantics as well as data. The manifold tier glues the presentation and domain tier. Device heterogeneity is handled by pre-described profiles in eXtensible Mark-up Language (XML)/eXtensible Style-sheet Language (XSL). The XSL document maps elements in the XML document to nodes in the result tree so that the renderer knows how to render them. Parsing the common XML file and the local XSL file generates the view at a particular user’s machine. However, this approach is not able to guarantee the consistency of modified 3D data. It furthermore takes a long time to transfer the 3D data when a user moves across diverse devices. Preda et al. [6] proposed a formal model of adaptive rendering for multi-user 3D games. They defined a set of transformations for adaptation to heterogeneous client devices, rendering, coding, simplification, and modelling and possible process chains for visual adaptation. However, they did not provide any mechanism to dynamically adapt the current context at runtime and to increase interactive performance for users. To increase interactive performance of 3D simulation and rendering, a hybrid rendering approach in which a common subset of 3D models are rendered on both the client and server sides, could be utilized. Engel et al. [7] proposed a system which maximized utilization of resources on server-side by delegating rendering of highly complex 3D scene to a dedicated server. It, however, still limited users to exploit diverse devices adapting to the current contexts because a client needed to transfer large amount of 3D data to server on-demand. Weaver et al. [8] proposed a perceptually adaptive rendering system for immersive virtual reality which reduced the computational burden by rendering detail only where it is needed. Eccentricity from the user’s point of gaze is used to determine when to render detail in an immersive virtual environment, and when it can be omitted in order to display higher quality environments without reducing interactivity.

2.5 Deployment Domains

2.5.1 User-centric Media

Nowadays even the smallest of devices have the computational capabilities and connections to provide access to most of our media (music, video, documents etc) [3]. This provides interactive environments where media can be accessed, visualized, interacted and shared with [2]. With the uprising of social networking sites, such as Facebook, Twitter, LinkedIn, Flickr etc, a whole new way of distributing and interacting with social media content has been developed[53]. According to Tian et al. [54] this has led to a new area of research Social Multimedia Computing. Research that has led to this new area also made considerable efforts in the field of audio-video systems and applications convergence, especially in "smart home environments" [55] as well for "Mobile computing" [56]–[58]

2.5.2 Telemedicine

Telemedicine is a rapidly developing application system in the medical domain that harnesses the advances in telecommunication and multimedia technologies [59] and has led to many healthcare benefits, e.g., telesurgery applications that use image-guided robotics and real-time consultation (teleconferencing) for distance-based surgery [60]; and telemonitoring systems for elderly prevention and care by using wireless sensory devices to communicate the status/condition of patients with physicians [61]. Moving ahead from simple text, image and video communications, modern telemedicine applications are supporting complex systems such as surgery (telesurgery), collaborative diagnosis (tele-diagnosis), and online rehabilitation (tele-rehab). Furthermore, typically these telemedicine systems are used for hospitals that do not have the facility/-expertise, and locations could be remote or in under-developed countries, thus having restricted networking capabilities. In particular, teleradiology - a telemedicine system that involves the electronic transmission of digital radiographic images (e.g., MRI and X-Ray) from one geographical location to another, has revolutionized the healthcare practices by enabling efficient distribution and sharing of medical images. Teleradiology has evolved from teleconferencing and shared 2D images running on imaging workstations with LAN [62], to current state-of-the-art systems that are capable of streaming volumetric medical images in real-time via a wireless network [63]. Moreover, modern systems now support thin client/cloud computing [64] that processes (e.g., volume rendering) and stores all the medical image data in a client-server relationship, thus not requiring local imaging workstations to access and view the images. Despite the remarkable growth in the medical image processing and analysis capabilities, little progress has been made in the mechanism that enables real-time interactive collaboration among multiple users in teleradiology domain. In such applications, collaboration tools include the usual text and video based interaction, image navigation, and image editing, which we refer to hereon as 'collaborative editing'. Basic image editing involves appending information to the image without changing the image's state, i.e., textual annotations and measurements using a ruler or simple

Table 2.1: Classification of Social Media [67]

		Social presence/Media Richness		
		Low	Medium	High
Self-presentation / self-disclosure	High	Blogs	Social networking sites(e.g. Facebook)	Virtual social worlds (e.g. Second Life)
	Low	Collaborative projects(e.g. Wikipedia)	Content communities (e.g. Youtube)	Virtual game worlds (.e.g. World of Warcraft)

brightness/contrast state changes of an image and lookup table of the image stack using a simple strict locking concurrency control mechanism [63], [65]. Unfortunately, these studies degrade real-time interactive performance of users because only a single user, who has ownership of the shared object, can edit the object and the others wait until the lock is released by the current owner. They also barely provide a scalable interaction mechanism among a group of users nor a scalable data (information) sharing architecture among distributed multiple sites. It might not only cause degradation of system performance if the number of users increases but also limit to utilize diverse medical data distributed in the local sites. Our previous study [66] investigated the concept of using collaborative mechanism for real-time interactive segmentation among multiple users. The goal was to demonstrate that such capability can find large array of useful applications in teleradiology, especially for education and clinical usage.

2.5.3 E-Commerce

E-commerce has been booming in the last decade, from on-line services for buying digital wares, ordering physical products, booking flights, hotels are common. Full social frameworks are provided around these services and offer customer-customer feedback (ratings, forums, blogs). Eventually even e-mail will be replaced, or at least completely assimilated, by technologies such as *Twitter* and *Facebook*. Albeit similar software already existed for a long time, it became greatly accepted by the public (globally) and many companies adapt their marketing and internal strategies to take advantage and integrate these *social media* technologies into their core-business [67]. An interesting classification table concerning Social Media is given by Kaplan and Haenlein, table 2.1.

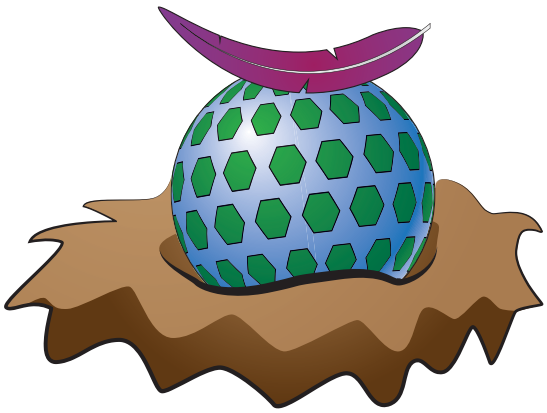
2.6 Summary and Conclusion

We have explored the core technologies and research areas that revolve around the research of 3D Adaptive Rendering, including the deployment domains *User centric media*, *Telemedicine* and *E-Commerce* with a strong focus on Mobile computing. As research on Mobile computing has mainly focused on device-centric approaches which makes a mobile device as a point of multimedia convergence to support nomadic access to multimedia, which is still limited to the usage

of a single mobile device with the limited resources. We aim to overcome these limitations, in the next chapter we address how to move beyond the home and device-centric convergence towards truly user centric convergence of multimedia. Our vision is to make a user as a multimedia central which means that "a user is a point at which services and interaction with them (devices and interfaces) converge across spatial boundaries". To realize the user-centric convergence, we defined three key challenging issues: dynamic distributed networking, mobile and wearable interfaces, and multimedia adaptation and handling. Dynamic distributed networking layer mainly focuses on a transparent access to diverse networks for seamless multimedia session continuity which enables a user to switch among different devices and networks with minimal manual intervention from the user. Mobile and wearable interfaces layer provides dynamic composition of wearable devices and various mobile interfaces to access multimedia contents exploiting diverse devices nearby to users, which make users free from using specific devices to access multimedia contents. Multimedia adaptation and handling layer support multimedia contents to be presented to different devices for personal manipulation which requires adaptation of multimedia to device or personal context along with seamless presentation of the multimedia for different devices.

Chapter 3

Methods and Design



3.1 Motivation

This chapter delves deeper into what we proposed (section 1.2), a context-aware adaptive rendering architecture. First we overview the architecture, lay out the aspects that need to be handled and provide insights in its components on a methodological level. The architecture itself is the backbone to the several application scenarios (use-cases) that were developed and experimented on. Albeit the initial designs remained the same, the architecture morphed along with the defined application scenarios which where conducted chronologically in order to accommodate the new requirements. As it tries to incorporate all the requirements of each scenario in a generic manner, not only to fulfil the requirements as a re-usable design for further application scenario, but also to gain a broader spectrum of applicability outside of the specified scenarios.

In the first section we discuss the architecture and lay out the three core aspects, adaptive rendering, collaborative virtual environments and scalability with heterogeneous device support. Before handling the *application scenarios* an introduction is given into an real-time virtual fashion model simulation library (VTO) as it functions as a basis for several of the application scenarios (section 3.3). Each application scenario is first introduced in section 3.4, where we justify the purpose and choice of each scenario and explain the order of how the scenarios came to be.

The first scenario in section 3.5 is the basic utilization of the architecture, where in section 3.6 the application is extended with Augmented Reality (AR) and using the VTO library. In section 3.7 we present a collaborative work, where the framework is used with different third party modules to provide a seamless hand-over of the running session to another device without interruption and how mobile devices can be used for data transfer to a localized collaborative device (touch table). The application scenario thereafter in section 3.8 takes the VTO library and provides it as a scalable service. Offering strategies for deploying similar services in a cloud-like environment. We then present how the framework can be used in a different domain, in this case Telemedicine, and focus more on collaborative aspects in section 3.9. The last application scenario, section 3.10 contains several envisioned services, showing off the various aspects of the architecture. Using the VTO for the simulation, but providing a collaborative pattern designer, from which automatically meshes are extracted and given to the simulation. An overview of the key aspects of each scenario is given in table 3.1.

Since the main focus is on bringing media into a user-centric context every application scenario is logically placed within that domain. Most of the scenarios are applicable for E-Commerce usages, either by deploying at small scale, LAN environments as native applications, or in at grand-scale, e.g. WAN environment as web-based services. Most notably are the last two application scenarios as they show the full capabilities of the framework and its usage.

Table 3.1: Application scenarios

Remote Rendering for Low-end Devices	Remote Rendering with Augmented Reality	Adaptive Rendering with Dynamic Device Switching	Service Distribution and Render Context Switching	The Collaborative MRI Segmentation	Collaborative Services with shared Data Models
Deployment of the framework. Multiple devices can access the same stream simultaneously. Input handling and adaptation are utilized.	A webcam on the mobile device is used. The AR tracking is either done on mobile or server. Rendering is performed on the server and composition either server or mobile device	Two sub scenarios, both on integration with 3rd party modules. 1 Enabling seamless switching of device. Rendering from server and streaming to a single session. 2 Seamless data transfer from device to device.	Focus on service maintenance and deployment. Using load balancing and automatic redirecting of a client connecting to a service. In addition the service is provided in a web-browser environment.	Multiple services for collaborative segmentation of MRI data. Offloading high data usage from client. Operating on same data, with device independency. Single point of data store which influences a autonomous simulation.	Offering coupling for complex data model for synchronization. Complex data models are data models where the use of the data is dependent on other data in the data model. Several services are used.
User-Centric	User-Centric	User-Centric	User-Centric	User-Centric	User-Centric
	E-Commerce	E-Commerce	E-Commerce	TeleMedicine	E-Commerce

3.2 Concepts and Architecture

Albeit the main target is to bring 3D content to any kind of device, the intended architecture should be capable to be used in a variety of deployment areas where 3D content may not necessarily be the main focus. This involves the deployment of services in a network, keeping several data connections with services, interactivity with services and propagation of changes in the service data states among users who share the same interest. As a result the architecture relies strongly on the concept of publish/subscribe model[68] and the PSSAM[44]. We reiterate the research context and objectives into three core aspects of the framework as follows:

- Adaptive rendering: To overcome resource limitations of mobile devices, we propose a context-aware adaptive rendering system which visualizes 3D content and a user interface, while dynamically adapting to the current context such as device availability in the current location and their capabilities, i.e., processing power, memory size, display size, and network condition, while preserving interactive performance of 3D contents.

- Collaborative multi-user environments: The ability to not only share the same view but also interact with each-other and with the provided service. In conjunction with the adaptive rendering the challenge is to preserve the interactive performance using adaptation schemes and maintain a synchronized collaborative data state of the simulations, client applications and services.

- Service scalability and heterogeneous device support: Whereas Adaptive rendering focuses on optimizing for a single device, the service scalability focuses on overall deployment scalability and optimizations. Handling adaptation for services and end-user application and devices. The challenge of being able to place services anywhere, create modular components that together form a service and easily can be maintained and support a seamless range of devices for visual output and interaction.

It should be noted that the presented architecture designs are independent from the technical designs (which are presented in chapter 4). As the architecture design aims to retain a certain level of abstractness and therefore platform independence, thus without being limited by implementation and deployment issues. The technical designs on the other-hand adhere more to the

feasible requirements of development and deployment environment. E.g. utilizing common programming patterns, restrictions in threading and multi-process setups, network communication, and buffering. With the technical designs the presented architecture design is reconstructed, yet it does not resemble this in all cases (depending on the requirements per application scenario certain aspects are treated differently).

3.2.1 Publish and Subscribe

The architecture leans strongly on the concept of the publish/subscribe model we therefore elaborate more on this, and justify the usage throughout our work. Therefore, at the base of the each presented aspect we apply this concept as follows:

- Scalability
  By using a modular approach where services or partial services can link together using a publish/subscribe approach can easily be deployed in different environments. A single service can reside on a computer, in order to have it announced it can either use a broadcast receive/response mechanism or a centralized service approach. Meaning that a client connects to the main service and asks for a service.

- Interactivity
  Interactivity is about binding input to functionality, which can be done using the same approach, whereas the input is the publisher and the functionality the subscriber. Yet the way it is announced can differ, in this case the binding of the input with the functionality happens from a publisher makes him known to the subscriber. Another aspect of interactivity is the abstraction of functionality through the use of 2D interface elements, using singular elements for the generation of events that are translated into, the selection of functionalities to be executed with the given parameters.

- Context Aware
  Context aware in this context is expressed in the device and network capabilities. In order to switch strategies upon changes in capabilities (e.g. change of device) different adaptation paths may be taken, this involves the use of different kind of services (e.g. remote rendering, or direct streaming of 3D data). Applying publish subscribe provides a uniform model for changing functionality in a dynamic setup of services that.

- Remote 3D Content Streaming
  The streaming of 3D content may involve, as mentioned in the previous point, several services, such as Level of Detail (LOD) reduction, Colour optimizations and general compression (e.g. video compression) algorithms. These may all be separated services that can be concatenated at runtime depending on the context awareness scheme. Aside from the separated components, there is also the tightly integrated optimization of the 3D content, internally in the engine and program logic. Often this is best known as the LOD (as and example, polygon meshes can be reduces in polygon count depending on distance), and is used in any major graphics engine, however this is fully dependent on the running

program and its restrictions on the LOD capabilities (e.g. we want no loss in quality in a medical simulation, however in an crowd simulation it is preferred over loss in fluidity).

The usual behaviour of the publisher/subscriber method can be best illustrated by the programming design pattern *Observer pattern* as shown in figure 3.1. The observer defines itself a as a *template* (interface) upon which concrete implementations are build, which are than able to be added to an *observable* object. Whenever the observable object changes its state it may notify all of his observer objects.

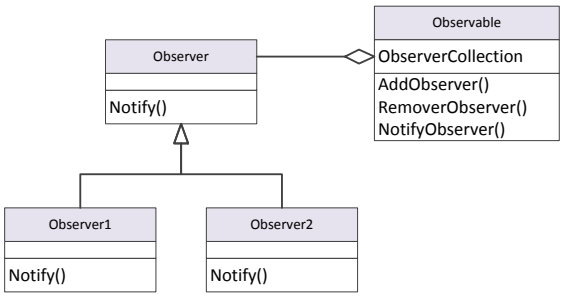


Figure 3.1: UML diagram of the Observer Pattern[69].

This concept is used throughout the whole architecture as it is the uniform method for keeping consistency among one-to-many relationships without the need of having tight coupling of functionality and as indicated in the points above it offers flexibility in all areas. May it be from a clean MVC point of view, scalability and multi platform deployment or functionality adaptation.

3.2.2 Adaptive Rendering System

In order to support polymorphic visualization of 3D content on heterogeneous devices, the adaptive rendering system exploits the PSSAM [44], in which an interactive 3D application can be typically divided into four layers; presentation, semantics, link, and adapter. The presentation layer supports the interface between the user and the shared semantics as well as the visualization of 3D contents. The semantics are organized into software modules which encapsulate the 3D content. The presentation and semantics are dynamically bound with adapter at initialization as well as runtime. The context-aware adaptive rendering system is composed of five logical layers as shown in figure 3.2. The network layer provides not only the low level connection between the server and client using TCP or UDP but also the transparent change of subscription end-points for seamless subscription to 3D content when a user switches one device to another. The communication abstraction layer, which is composed of two major components, being the communication manager and event manager, abstracts low-level events to high-level (application-level) events in order to mask low-level events generated by heterogeneous devices. It also marshals/un-marshals incoming events and redirects them to the appropriate components. The adaptation layer on the client side has two components, the context manager and resource manager. The resource manager constantly monitors the current state of the resources in terms of application performance. Based on the information provided by the resource manager,

the context manager can determine the optimal performance by taking adaptation decisions. The decisions are translated into events and sent to the adaptation manager on server-side so that it executes the appropriate adaptation strategy. The presentation layer contains the render engines. On the server-side, a 3D engine is rendering the 3D content and generates images of different quality based on current client resource capability. To transmit images to a client, the frame-buffer is taken into the compression manager. This generates a stream of compressed frames (similar to a video stream) that is being decompressed on the client side and displayed using a 2D rendering engine. The dynamic state management on the server-side maintains the current state of the 3D simulation in order to switch from devices without losing the current simulation. In the application layer the application logic is responsible for client side handling of input and overall client side functionalities. On the server side it is the *data storage* containing the 3D content that is needed for the 3D rendering and simulation.

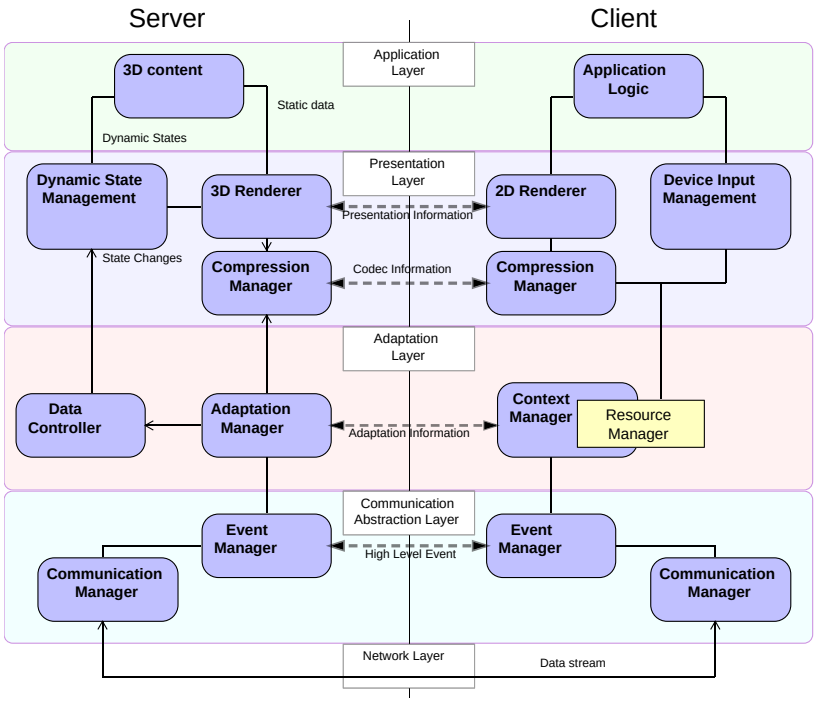


Figure 3.2: Overall architecture of context-aware adaptive rendering system.

3.2.3 Run-time Presentation Adaptation Control

We mainly focus on increasing responsiveness of 3D content taking into account the current context of devices. This means that whenever the user performs an action, the response to this action should be shown to the user within a certain amount of time. Figure 3.3 shows the typical flow of the remote rendering with interaction. The time taken between the creation of an event and getting the results back to the user is the time that we reduce with the adaptation manager. The three main stages are: send the events, perform simulation cycle and send the results. Assuming that the simulation is fast enough to deliver a response within a certain amount of time, the main adaptation takes place after the simulation step. Here a compression algorithm is used in

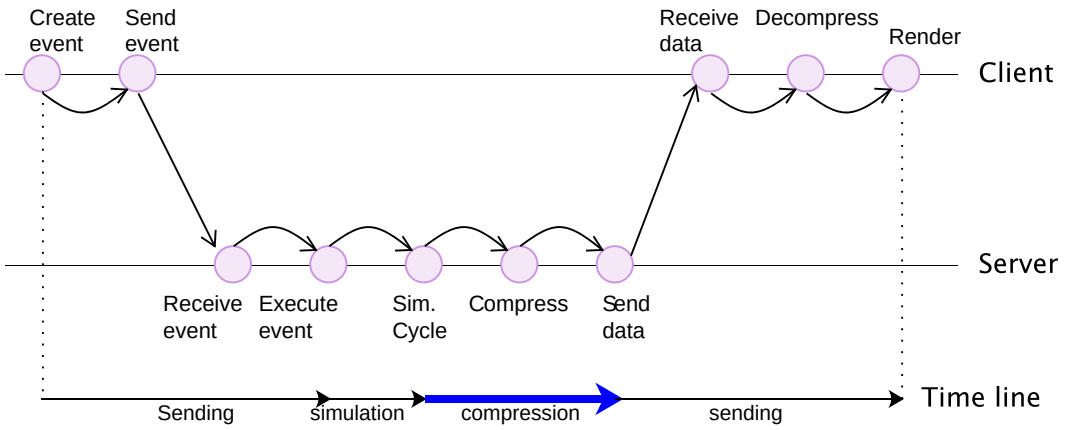


Figure 3.3: Round trip time, from input to visual response.

order to reduce the amount of data that needs to be transferred. For a thin client this is the compression of the image-data into an image/video stream. For hybrid or full client rendering, the system compresses the actual 3D data (e.g. lossless compression of vertices). In figure 3.4, we can see the several adaptation points given by diamond shaped objects. The dynamic adaptation

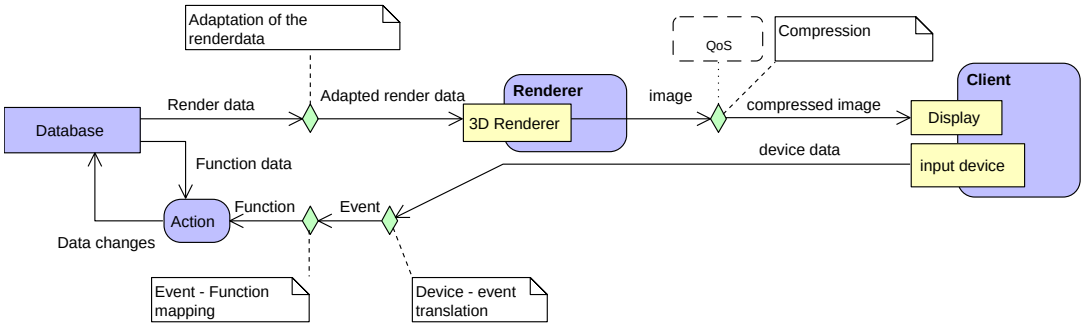


Figure 3.4: Adaptation points, denoted by the diamond shaped objects.

is aimed at the regulation of the output data. We increase the perception of visual interaction with a remote rendered 3D environment by introducing a temporal adjustment of presentation quality adaptation mechanism. By decreasing the quality, the decoding speed is increased on the client side which offers a more fluid experience and interaction feedback can be visualized faster. The decreasing of quality is indicated by the client and only when necessary, i.e. fast interaction and frame rate stabilization. Additionally for thin-clients the frame-rate is increased. Whenever the user interacts with a 3D object, the actions are displayed faster but at the expense of image quality. The decrease of image quality is achieved by switching to a stronger compression algorithm (if the decoding side can keep up), by changing compression parameters and/or if the client supports fast image scaling the actual image resolution can be reduced and up-scaled on the client side. We also use another approach,which detects changes in the rendered frame and compares it with previous rendered frames. If the amount of difference from the previous frame is greater than a certain threshold the adaptation control changes the compression strategy by

reducing the quality temporarily. The third approach is the encapsulation of expected load for functions or performance indicators. This can be done manually or at run-time. Manually by reading the load factor for a function from a predefined configuration file. This is the same configuration file that defines which events are bound to specific functions. The run-time approach is a profiler algorithm that, whenever a function is executed, (based on an incoming event) measures the time that is needed for full execution. The difficulty here is that a function itself can be very fast, but the effects of the function (influencing the simulation) can result in longer times. Either by predefining the load factor or by adjusting the load factor through the measurement of function execution another feature can be introduced, namely load prediction. By anticipating the actions and their resulting time-frame for user feedback the level of adaptation can be controlled to achieve a constant time cycle. Another, more straight-forward, approach to keep a constant time cycle is to keep a constant data rate. This does not take into consideration the quality of the network but only the amount of data being sent. It can be achieved by fixing a specific frame rate and keeping a threshold on the data size for each frame. If the size of a frame after compression is higher than a certain threshold the compression quality is lowered until the threshold is met again. By default, we use this passive adaptation rule. However, depending on the device profile a preferred frame-size is given, which is the actual data size for a frame (after compression) to be sent to the client. An extension to the dynamic adaptation control is based on the network load and should be balanced together with the frame rate. The whole procedure is shown in figure 3.5.

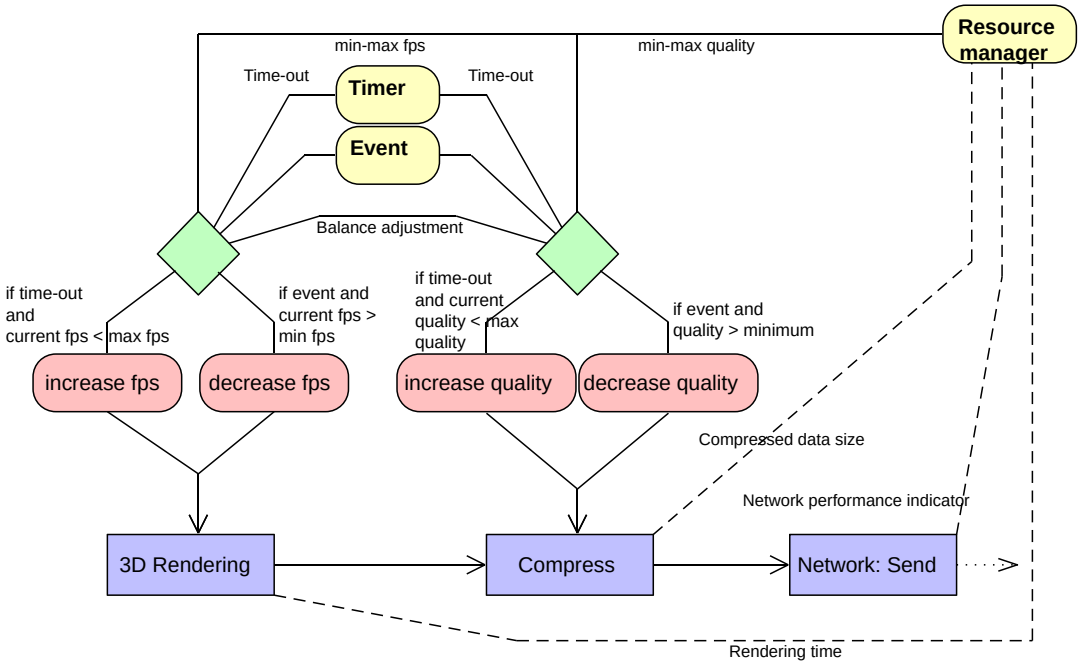


Figure 3.5: Adaptation manager: the adaptation control.

3.2.4 Dynamic Interface Adaptation

In order to overcome the physical heterogeneity limitations in display capabilities and input controls on client devices, we provide a dynamic user interface reconfiguration mechanism for interaction with 3D content. It means to change the way the interface is presented to the user (big screen or small screen brings several design issues with it) and to adapt to input capabilities of the client device. Figure 3.6 shows the details of dynamic mapping. Each Element is an event

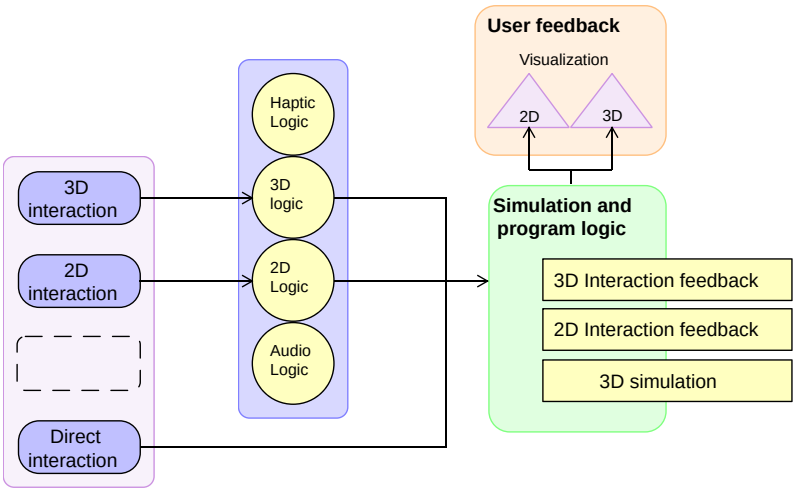


Figure 3.6: Dynamic event mapping of user interface.

handler, and is listening to certain events to which it can respond by creating new events. Each event received from an input device is first handled by the Client Event handler, this module determines if the event should be handled by the client or the server. Then on the server or client side, the raw event is sent to the corresponding Element that is listening to the device. An Element can have a representation, which can be a form of visualization in 2D or 3D, but also in the form of audio or any other form of feedback. The handling and control of the representation is performed by the logic that is assigned to the specific Element. The representation of the element is loosely coupled and therefore it may reside on the server or client side. According to the Element logic, the raw event can be translated into higher level events. These can be a representation update, simulation, adaptation or any other application specific event. This makes an Element a dynamic building block that is used by the User Interface adaptation manager to construct and modify the user interface whenever needed. For example a combo box on a PC is displayed in 2D using Windows, GTK+ or Qt native widgets, but it can also be displayed as a ring selection in 3D, using a different representation but the same logic, and accepting the same events, or accept different events but with the same logic. Currently there are three forms of user-interface-interaction implementations, two with a visual representation and one with no interface visualization as follows:

- Interaction with 2D interface. The default 2D interface providing windows, buttons, text fields and other widgets with its entire well established visual feedback mechanism. This

is an intermediate layer between input device and the 3D simulation, providing the means to perform complex operations and provide the necessary data for it. For example a text-box and a button, might update simulation parameters, which will be more dynamic than an *increase* and *decrease* button, but at the expense of more user interactions (typing and confirmation with a mouse-click or keyboard command, instead of a single click).

- Interaction with 3D interface. This form of interaction is integrated into the 3D environment and therefore provides the user with direct interaction with 3D objects. This is usually achieved by means of a 3D pointer, which is able to hover over 3D objects and whenever the user executes a command to select or perform some other action it is directly executed on the specific object. Aside from the input device, this interaction mechanism boils down to *see the 3D object*, *select the object*, *execute operation on object* and *wait for visual feedback*. The visual feedback should be direct if the simulation behind is a real-time simulation. In essence this approach is similar to the 2D interaction mechanism, but by adding one dimension more it provides us with new abilities as to how we perceive the interaction and handling of virtual objects. The implementation of a 3D interface feedback may have similar features as its 2D version; for example selected objects should be highlighted, or just by hovering over objects information on them should be displayed. This feedback is different from the 3D simulation itself, as it does not intervene with the simulation, but just provides visual 3D feedback.

- Interaction with Implicit Inputs. It has input device bindings directly to an event that changes simulation parameters without showing it in the sense of an interface. For example 3D movement, the camera is being moved around in the 3D world. The visual feedback is that the user gets the impression of moving in a virtual world, but there is no direct feedback from any object in the world. It is exactly the same as the 3D interaction, but without a direct 3D interface implementation. All of these interaction mechanisms can be used with any kind of human interface device (HID), such as a pointing device (e.g. mouse) or keyboard etc. Each input device can be coupled differently according to its input capabilities, the application capabilities and user preferences. The binding between the application and device can be hard-coded and to some degree the user can set its own preferences. However in order to switch from one device to another, the interface mapping needs to adapt accordingly. Another form of adaptation is dynamic coupling in which coupling changes depending on the current context. For example if the connection between device and server is very bad, than 3D rotation can be limited from smooth rotation to fixed rotation (for example front, side, top view).

3.3 Controllable and Adaptable Virtual Fashion Model

At the base of some of the application scenarios lays a real-time virtual fashion model simulation library which is named Virtual Try-On (VTO), we therefore introduce this library prior to the scenario sections. The VTO consists of three interconnected parts: A body sizing module,

a motion adaptation module and a real-time cloth simulation library. The combination of these three libraries provides us with a template model that can be fully customized to match the user's measurements. Any animation that comes with the template body is adapted in real time to fit the personalized anatomy and the cloth simulation library allows for any type of garment to be simulated on the body as well as the real-time adaptation of garment size. This allows the user to quickly evaluate how a certain garment looks on him/her, as well as the real-time visualization of different garment sizes to select a best fit. The body resizing module is based on a parametrized human body model as described by Kasap and Magnenat-Thalmann [70]. Based on a template model we can modify up to twenty-three measurements parameters, corresponding to twenty-three different body areas. If the parameter concerned involves a change in the length of a certain body part, the underlying skeleton used in animation is resized as well. Body sizes are generated to the anthropometric measurement standards, assuring a physically plausible deformation. Since the deformation is based on actual measurements it can be adapted to match the measurements of the user. This process is completely dynamic. The body is modifiable both at the rest pose and during animation. The results of this process can be seen in figure 3.7. Changing body sizes and lengths means that the animation recorded or made for

Figure 3.7: Starting from a template model on the left the body is increasingly adapted during animation to its final anatomy on the right.

the template model does no longer match the new anatomy. A change in lengths of parts of the skeleton will cause visual errors, such as foot skating for example, if they are not taken into account for the animation. And these changes, as well as changes in girth, might result in incor-

rect inter-penetration between body parts. Besides this the animation might become physically implausible because animation based on a small and light body does not look correct on a larger and bigger built avatar. In order to solve these undesirable effects we have integrated a motion adaptation library based on the work of Lyard and Magnenat-Thalmann [71], [72]. Any changes to the parameterized human body model cause the animation to be adapted to the new anatomy resulting a visually pleasing and plausible animation of the personalized avatar. Again all of this can be performed completely interactively and without any significant performance impact on the application.

To complete the virtual fashion model we need to combine the adaptable and animated avatar with a cloth simulation module. Cloth simulation is a complex and computationally intensive process that can easily turn out to be the bottleneck for the performance of your entire application. This can be partially alleviated by reducing the geometric complexity of the simulated garments, choosing an efficient but less accurate simulation scheme or by approximating the collision interaction between garment and body. In this specific case however the need to give an as accurate as possible representation of garment behaviour and its fit to a body puts some restrictions on the abstractions we can make. We have therefore integrated a simulation module as described by Magnenat-Thalmann et al.[73]. This module, which uses an optimized implementation of the tensile cloth model by Volino and Magnenat-Thalmann [74] allows us to simulate garments based on measured fabric parameters with fairly realistic results. It furthermore allows us to switch between different garment sizes on the fly based on integrated grading information. Collision detection is performed using a skinned representation of the garment and its distance to the nearest body vertex as described in [73].

The VTO's three individual libraries have been integrated through a thin wrapper class that hides most of the complex interaction between the three libraries described above, exposing only basic functionality to load and interact with body and cloth. All is built on top of or in close relation to OpenSceneGraph (OSG). We have created several custom callbacks to perform the necessary per frame updates for skinning procedures and cloth simulation calls. The model itself as well as the garments, animation and parameters used in body sizing are stored in a single COLLADA file[1], for which we have modified the OSG plug-in to load custom data. Some examples of the results we obtain with the VTO can be seen in figure 3.8.

In order to put into perspective the use of this simulation we have tested it on a range of hardware. The simulation is implemented using C++ and build for x86 systems. The application performs on simulation step a simulation calculation and rendering cycle. Thus every frame rendered is also a simulation cycle. Later versions use a decoupled simulation and rendering, as rendering is often also related to interface aspects and in order to have a more response interface it can benefit from this decoupling. Table 3.2 shows the results of this test. The UMPC as our slowest unit is barely capable of rendering a single frame per second. In fact the frame-rate capturing is rounding it up to a single fps. The tablet which is far superior in respect to the UMPC isn't doing much better. The only decent results are gotten from desktop systems. Which

[1]Collada web-link: `http://www.collada.org/`

Figure 3.8: The Virtual Try On in action. On the left a front and side view of the base model wearing a size 34 dress and on the right a more voluptuous model wearing the same dress scaled to a size 38.

Table 3.2: VTO performance in frames per second

UMPC (vii)	Tablet (viii)	PC (iii)	PC (iv)	PC (v)	PC (vi)
1	2	20	27	30	30

makes this an excellent example to showcase the benefit of streaming. It should be noted that the implementation of the VTO is a single process and doesn't use threading, this greatly reduces the performance on current systems which all contain multi-core processors. An attempt to overcome this problem is presented by Kevelham [75] and is partially described in section 3.10 where an early build of the simulation is used as a replacement.

3.4 Application Scenarios

As briefly introduced in section 3.1 there are six application scenarios that will utilize the proposed framework. In this section we motivate the choice of applications and look at which stage they were introduced. Per application scenario a background is provided, explaining certain design en development choices taken.

3.4.1 Remote Rendering for Low-end Devices

The first application scenario *Remote Rendering for Low-end Devices*, evolved with the framework and was the first to be utilized, during the initial prototyping and brainstorming for the design of the framework. It started out as a fairly simple TCP based server-client application sending packets of uncompressed images. Extended to use UDP for faster transfer, and as it turned out, it later became a requirement for *Adaptive Rendering with Dynamic Device Switching*. At this point there was no compression, frame-rate control or event adaptation. It functioned as an initial

test, fulfilling some of the requirements of the project for which it was build (see section D.1). Gradually more functionality was added up until the full scheme as shown in figure 3.2 was covered. Although the application, or better said the library used by the application, represented the distinctive layers, it did not adhere to the requirement of offering a scalable solution as stated in section 1.3. So far it covered event handling schemes, remote rendering adaptation and heterogeneous device support. The *real* framework had yet to be developed.

In order to tackle the scalability the notion of the publish-subscribe methodology was applied more strongly and led to the encapsulation of functionality within nodes that can be addresses uniformly. This is described in the next chapter Technical Design And Implementation(section 4.2) in more detail. The original program became obsolete and the new framework was applied. Aside from separating several functionalities, such as frame grabbing and compression, it now offered better scalability due to the uniform approach of encapsulating functionality which led for example to separating visualization from simulations into their own respective processes.

The content for the first application was of no concern as it was more important to have the basic functionality laid out. Therefore a static scene was used and minimal interaction could be performed with it. Aside from the 3D visualization no user interface, neither in 2D or 3D, was presented, and only the virtual camera could be rotated using touch or mouse movement.

3.4.2 Remote Rendering with Augmented Reality

Augmented Reality was shortly introduced after the build of the first application, and was part of a requirement for the project (see section D.1). This led to the extension of the previous application into the application scenario textitRemote Rendering with Augmented Reality and was upgraded in a similar way. The interesting part here is the use of AR which proved to be quite difficult in combination with the required content that had to be displayed. Where in the first application the content was a static 3D model, here it was replaced with a controlled animated avatar. Which was further extended with a cloth simulation (see section 3.3). This made it impossible to run on a mobile device, and therefore the simulation had to be run on a server. At this stage the simulation and rendering where tightly integrated and could not be easily separated (without reprogramming everything from scratch) and left us with simulation and rendering of the avatar on the server. The mobile device, with the camera, provides either AR information or the camera image to the server, which is presented as two designs in section 3.6.2.

The content in this application is based on an in-house developed/researched cloth simulation and avatar rendering framework, the VTO. However it should be noted that this application never used the final developed framework, as due to changes in the project the requirements changed as well, forcing us to look into new directions. Which led to application scenario *Adaptive Rendering with Dynamic Device Switching*.

3.4.3 Adaptive Rendering with Dynamic Device Switching

The initial design for the framework was setup and ideas for the scalability were investigated. In an collaborative effort between several universities a solution was provided to grand more freedom to the end-user. Although it didn't solve the scalability issue, it greatly enhanced the flexibility instead. By utilizing *Mobile IP* technology, the user could change from one device to another, without the interactive session being lost. This technology (provided by a partner within the Intermedia project, see section D.1) however had limited support for network protocols, as only UDP was supported. This made it more difficult to have services built on top of it that use sophisticated user sessions, as basically it would require building a custom TCP implementation on top of it. It did however show that the framework could be easily adapted to this new situation. As at around this time the implementation of nodes was established and the uniform handling of data made it easy to control the flow of data packets. As a secondary sub scenario, the introduction of seamless data transfer was introduced, leading to a scenario where data from a phone was transferred to a interactive table, where the application could facilitate multiple users on one screen using touch and at the same time stream to external clients (e.g. mobile devices). However due to complexity of all the third party components, provided by the project partners, this setup was hard to maintain and could not be replicated after a while, due to ending of the supporting project.

The content for this application presented is based on an in-house developed/researched cloth simulation and avatar rendering framework (the VTO). A first look at user interface adaptation was done using Qt interface language as a User Interface Mark-up Language (UIML) send from the server to the client. Where upon connection a (UDP) packet was send to the server containing the profile of the device. Whenever the user would switch from device, the new device would announce itself to the server by providing a new profile.

3.4.4 Service Distribution and Render Context Switching

This application scenario directly made use of the presented framework (as it is described in chapter 4. Its content, alike previous, based on the VTO now offered more functionality towards the user. An interface is presented that enables the user to change the appearance of the avatar, all the while the cloth simulation continuously updating the garments. It is the first scenario where partial implementation of the framework ported to other languages, such as Java and C#. Several interface adaptation solution were analysed, depending on the implementation and libraries used. In continuation of the user interface adaptation it was extended with a Java implementation using a custom protocol with a full client side implementation that can adapt to the server interface protocol and last a fully server side rendered user interface (presented in 3D space), which is then included in the streaming.

This scenario was build in several stages and presented within several projects (see sections D.3 and D.4. The first implementation concerned the service distribution and having multiple server/client connections and was a full C++ implementation. Where the second extended to a

web based service for which we used Java applets (albeit other options such as ActiveX, Flash and HTML5 are feasibly). This allowed the user to access the provided service through a browser. The last stage was providing a multi-user experience with a single service instance, meaning where previously only one user could customise the 3D avatar, now multiple people could see the same and interact with it independently. Albeit the Java client was not updated to this implementation, a C# implementation was deployed on a mobile phone instead.

3.4.5 The Collaborative MRI Segmentation

Another project (section D.2) provided a different look at how to use the framework. With a more focus on collaboration and less on the 3D visualization (albeit still present) it shows that it can be deployed in different domains. Therefore this project offered a unique opportunity and after some investigation it showed that there is a lot to be done in Telemedicine. As a requirement for the project MRI data had to be segmented and this is done manually and supported by automatic segmentation algorithms[76]. This became our leading point for collaboration, offloading the datasets to dedicated servers and support thin clients a with segmentation application.

The content provided comes from a prior manually segmented 3D surface model, that is being adapted/reshaped conform a different MRI data set. The deforming 3D shape can be visualized, however the editing itself occurs on a 2D plane (on top of a MRI slice) and therefore is solely presented in 2D. The application scenario has been build in two stages. The first build used a preliminary version of the framework and offered a limited form of user interaction. Restricted to placing points that influence the simulation, there was no direct influence by other users other then removing points placed by others. All data was provided by a single service point that also runs the simulation. Furthermore the MRI visualization and 3D rendering was streamed using JPEG compression. The second implementation improved on the first by modularizing functionality in several services, such as the mesh deformation simulation, MRI slice provider, a mesh cut through visualization and a publish-subscribe data sharing. The client offers more complex interactions and uses a better adaptive rendering for the streaming of MRI data. Albeit in essence the same, it offers more depth and is capable of showing more the benefits of the presented framework.

3.4.6 Collaborative Services with shared Data Models

The content for the VTO is produced by an in-house developed/researched software called Fashionizer [77][78]. This formed the basis for this scenario. The original intention was to extend this software into a collaborative editing service. Due to legacy requirements, this unfortunately turned out to be prohibitively impractical. Therefore a subset of the features was taken and a new pattern designer service was implemented. As a consequence this limited the scenario as a whole due to missing features, however it's main purpose is to utilize the framework and this is still being presented. Albeit the overall design shows a full system (section 4.8, figure 3.24), it is shown that implementation wise it didn't get that far. Instead several services are isolated and tested in two smaller deployment stages. The first is the *collaborative pattern designer*, which

allows to, as the name implies, collaboratively create patterns, that can be exported into a format that is supported by Fashionizer and a newly created (and still under research/development) GPU based cloth simulation. It makes full use of the publish/subscribe data sharing and illustrates this with a complex data model (data elements with dependencies). The second stage is the simulation of the cloth and streaming the 3D rendering to the end clients. This is based on the previously mentioned GPU accelerated cloth simulation running on a dedicated server with a multitude of graphics cards, which sends the deformed mesh to a render client. The client then can function either as the end client or as a proxy streaming service for further propagating the visual output to an external end client (such as mobile devices that can decode the video stream but would not be capable to render the 3D mesh itself).

3.5 Remote Rendering for Low-end Devices

As the initial step towards a functional framework based on the proposed architecture, we focus first on the remote rendering approach with adaptation capabilities for providing a stable stream of screen information. This setup can be layered as shown in figure 3.9, provides a streaming of image data to several clients and in return the clients can send input events for interaction with the 3D environment. A minimal protocol was set for controlling several parameters in the service side. Such as frame rate or update rate, switching between a push and pull mechanism for updates and for packing interaction events, such as mouse/touch movement for cursor/3d camera control. The design for this setup is therefore kept minimalistic. The service as well as the client

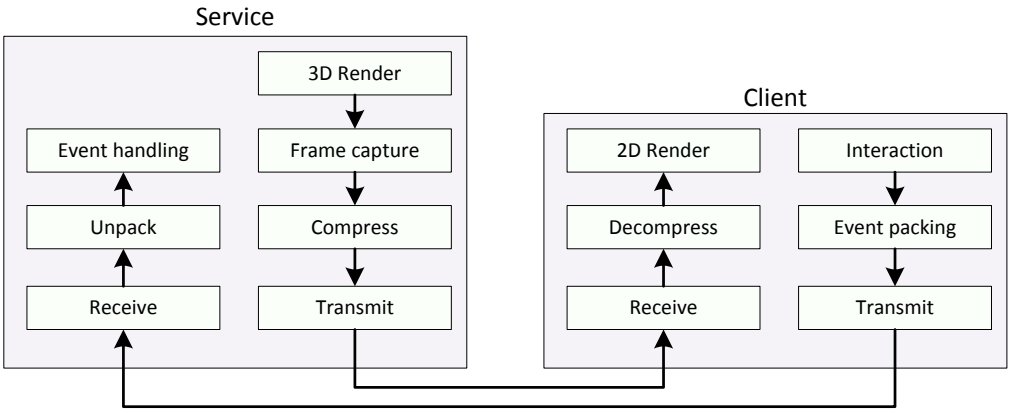


Figure 3.9: Overview of remote rendering in its basic form.

have a single input and output stream. Whereas the output of the service can be defined as the visualization data for the client (as input) and takes input from user interactions made available from the client. The 3D rendering in its basic form does not involve any in-engine optimizations, instead it can be seen as an independent module that outputs uncompressed images of the 3D scene. This is done by the frame capture and can be done in several ways (see for implementation section 4.2.4 for compression options and 4.3 for server client implementations). The image than is being compressed, either lossy or lossless and provided to the transmit module. This already involves all the layers as presented in figure 3.2, where the application layer is the 3D

content (including the virtual camera), the presentation layer the 3D rendering and capturing, the adaptation layer the compression and the communication layer the packing of the data into a stream, and finally put it into the network layer. Upon receiving at the client it travels all the way up in the layers, up till the presentation layer, where it is being displayed on screen. On the client side in the application layer the logic enables the input of interaction from the client device, depending on the type of interaction it follows a certain event packing scheme. E.g. mouse movement is packed into a fixed event rate, accumulating the in-between event, a mouse click however is send directly. The event rate transmission can be controlled either by user setting or automatic rate speed. In this application scenario the functionality of interaction is pre-defined, as the user can only move the 3D virtual camera, this either gives the impression of navigating in a 3D environment or the rotation of a single 3D scene, depending on the virtual camera control scheme.

3.6 Remote Rendering with Augmented Reality

In this application scenario we extend on the previous one by introducing AR and the use of a *heavy* simulation which the user can see the output from (the VTO, section 3.3). This brings a great challenge to mobile devices as it combines two things, the need for remote simulation and the handling of merging the rendering with a mobile camera. To be more clear on the mobile camera, it can be any kind of camera attached to a mobile device, e.g. camera on a smart phone, tablet or a web-cam connected to a laptop.

3.6.1 Trends in Augmented Reality

To allow for AR applications to be deployed on mobile devices, a very recent trend in mobile AR systems is the usage of UMPC[2]. A number of researchers have started employing them in AR simulations such as Wagner et al [79], Newman et al [80] and specifically the Sony Vaio U70 and UX180, as well as Samsung Q1. Elmqvist et al [81] have employed the Xybernaut Mobile Assistant, which, although shares some common characteristics with UMPCs, does not belong in the UMPC category. Wireless Local Area Network (WLAN) have already being utilized in AR applications. Human Pacman (Cheok et al. [82]) is an interactive role-playing, physical fantasy game integrated with human-social and mobile-gaming that emphasizes on collaboration and competition between players. By setting the game in a wide outdoor area, natural human-physical movements become an integral part of the game. In Human Pacman, Pacmen and Ghosts are human players in the real world who experience mixed reality visualization from wearable computers. Virtual cookies and actual physical objects are incorporated to provide novel experiences of seamless transitions between real and virtual worlds and tangible human computer interface respectively. Human Pacman uses WLAN technology to enable mobility in small scale environments. While the transmission range of one WLAN base station or access point is typically 100 meters, a number of access points can be used to provide coverage in much larger areas. Using the approach, currently there are efforts underway to provide WLAN coverage in entire cities

[2]UMPC (vii)

which will contribute significantly in the proliferation of AR applications. Virtual Characters have already been synthesized with real actors in common non-real-time mixed reality worlds, as illustrated successfully by a number of cinematographic storytelling examples. One of the earliest research-based examples was the virtual *Marilyn Monroe* as appearing in the research film "Marilyn by the lake" by MIRALab[3], University of Geneva as well as Balcisoy et al. [83]. However these compositing examples involve non-real-time (offline) pre-rendered simulations of virtual characters and mostly are rendered and post-processed frame by frame in an ad hoc manner by specialized digital artists or compositors as they are termed in the industrial domain of special effects (SFX). The active SFX sector with applications in film, television and entertainment industry has exemplified such compositing effects in a constantly growing list of projects. In this scenario we study the recent surge of employing virtual characters in mobile AR systems. A first such real-time example in a mobile setup employed virtual creatures in collaborative AR games (Hughes et al. [84]) as well as a conversational and rigid-body animated characters, during a construction session (Tamura et al [85], Cheok et al. [82]). In cultural heritage sites, a recent breed of mobile AR systems allows for witnessing ancient virtual humans with body animation, deformation and speech, re-enacting specific context-related scenarios (Papagiannakis et al. [86]) as well as allowing visitors to interact and further inquire on their storytelling experience (Egges et al. [87]). An important aspect of such AR examples is that these virtual characters are staged in scenario-based life-size scaling, position orientation as a result of marker-less AR tracking and registration. Further recent examples of marker-based tracking such as ARToolkit[4], various researchers employed dynamic content on top of such markers, such as 3D storytelling book content (Billinghurst et al. [88]) and other interactive characters reacting to user's actions (Barakonyi et al. [89], Wagner et al. [90]). Very recent examples include also the use of virtual characters as outdoor navigation guides (Schmeil et al. [91]).

3.6.2 Two Designs

Even though the VTO itself runs adequately on a dedicated workstation, preliminary tests confirmed our suspicion that the UMPC used[5], as our low-end device, does not have the computing power needed (VTO performance table 3.2). It reached its limit on just a couple of frames per second and is heavily limited by its Central Processing Unit (CPU). Besides running the VTO several other tasks are needed to run at the same time, which are some form of marker tracking and perform graphical operations to provide an augmented reality visualization. Tasks to be performed:

- VTO simulation. A CPU intensive process which is combined with a rendering engine, the output should be combined with the web-cam image.
- Web-cam image capture. The images are to be used for the tracking of a marker, should be rendered to the UMPC device and underlay the rendered image of the simulation.

[3]Marilyn by the lake, `http://www.youtube.com/watch?v=tr6SPD4C_ns`
[4]Artoolkit web-link: `http://www.hitl.washington.edu/artoolkit/`
[5]Asus UMPC (vii)

- AR tracking. Using fiducial markers for tracking. The tracking works on a frame by frame basis and outputs a transform matrix relative to the *camera*. The transform matrix is used by the rendering part of the VTO simulation, to offset and rotate the model accordingly.
- Blending and render. The image from the web-cam and the rendering from the simulation should be blended together and visualized on the client device screen. This means that the simulation output is laid on top of the web-cam image and therefore should contain information about the transparency. There are two common types, using a colour key where a single colour marks per pixel transparent or not or an additional layer (e.g. RGBA) with per pixel blending information.

We already have established the requirement of having the simulation on the server side, and therefore also the rendering output. Streaming the output has been shown before in the first application scenario. Here however the avatar with the clothing has to be positioned accordingly in respect to the camera position and the marker. Therefore the transform from the tracking algorithm should be provided. This can be done either on the client and send the transform matrix to the server or have the web-cam image directly send to the server and have the tracking performed on the server as well. This led us to two different designs as some of these tasks are quite performance intensive themselves, and are implemented and tested separately. Each having at least the VTO running on a dedicated server and streaming rendered images to the client application running on the UMPC.

The first setup is given in figure 3.10. The client is merely responsible for camera capture, sending the result to the server and receiving and displaying the final augmented result.

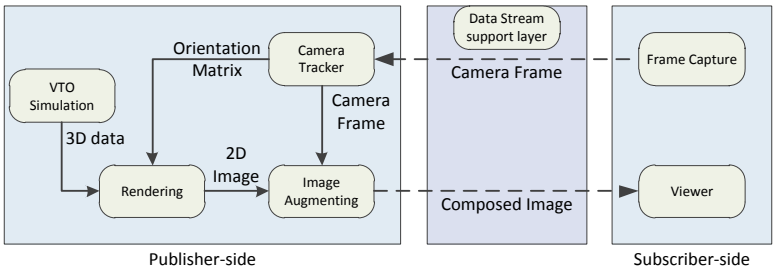


Figure 3.10: The server performs camera tracking, simulation and rendering of the VTO, as well as final augmentation of the camera image.

The *subscriber* is the client application with the web-cam. In this design the web-cam images is taken by the *frame capture* and send over the network to the server. The image is first processed by the *Camera tracker* using a tracking algorithm to orient itself in the given space by recognizing the fiducial marker. The orientation matrix is then given to the rendering module and the frame itself forwarded to the *image augmenting*. The *VTO Simulation* is an ongoing process from which the deformation of the 3D mesh models is used in the rendering. The image from the rendering is then blended together with the web-cam image in the *image augmenting*. The combined result is send back to the client.

The second design is given in figure 3.11 and shows that several components have been moved to the subscriber side.

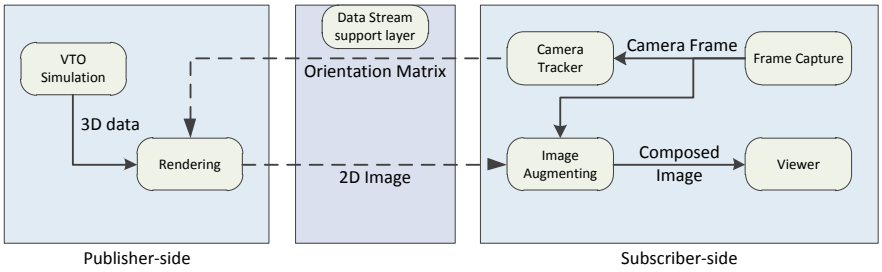


Figure 3.11: In this setup all tracking and augmentation tasks are performed on the client. The server is responsible for running the VTO.

In this case the tracking is done on the subscriber side and the resulting orientation matrix is send over the network to the publisher and directly inserted into the *Rendering* module. Which alike the previous design, outputs the simulation rendering and is send back to the subscriber, but without compositing the final result. Which is now performed on the subscriber side.

Both approaches have their benefits and negative side effects which are highlighted in the implementation and experiment sections.

3.7 Adaptive Rendering with Dynamic Device Switching

In this scenario the main focus lies on bringing 3D content not only to any device, but being able to switch from device to another device without losing the session. This application scenario fell in line with the requirements within the Intermedia project (see appendix D section D.1). The challenge to bring together several services into one, where each of the services was treated as a black box, yet providing 3D remote rendering able to switch from device to another. The scenario is subdivided into two, the first with a focus on the integration with several technologies that enable the switching of a networking session between devices, utilizing a localization of the user near a device and provide a layer of added security.

3.7.1 Real-time switching device

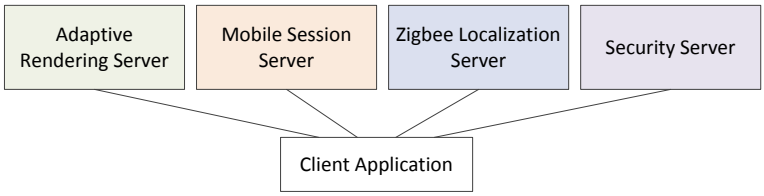


Figure 3.12: The client application and communication service servers.

The adaptive rendering server is based on the previous application scenario's remote rendering structure and streams to a given IPv4 address. This address, also known as a Personal Address (PA), is provided by the Mobile Session Server which is based on an implementation of the

Mobile Ip[55]. The SAIL system [92] provides localization through the use of Zigbee modules in its surrounding and is given as the *Zigbee Localization Server*. The security server authenticates the device being used and allows or prohibits the connection with the streaming server (in this case it allows the device to use to switch to the given PA. As addition to this application scenario

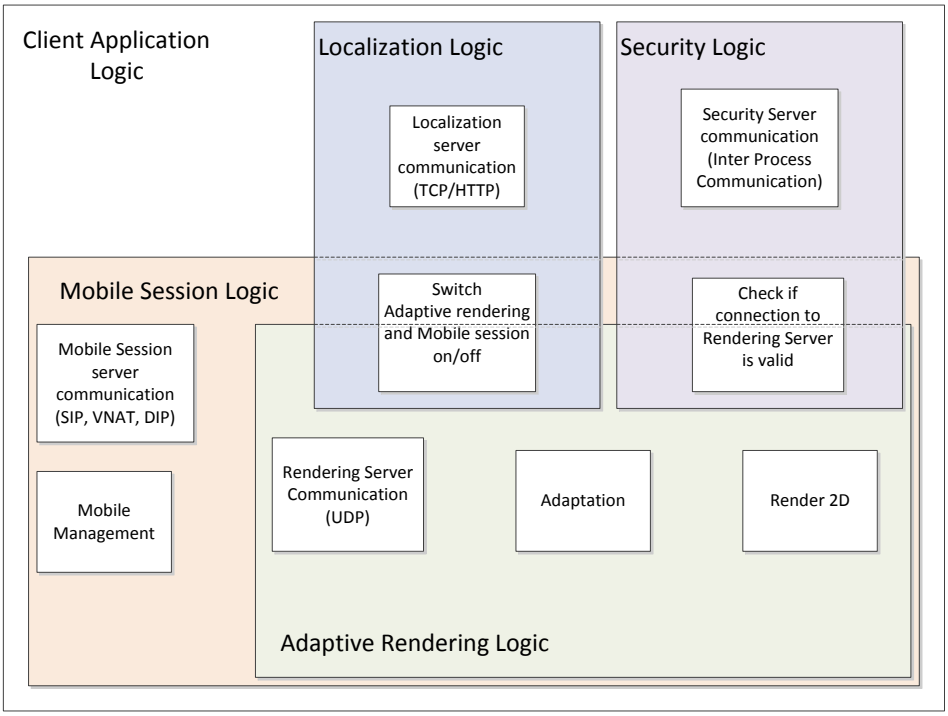


Figure 3.13: Client internal components, where overlap shows the connectivity with other components.

the user interface is defined on the service side, upon device switch the device introduces himself to the service upon which it may send an updated user interface. This is utilizing the scheme presented in section 3.2.4.

3.7.2 Data sharing among switching devices

In addition to switching devices we look also at how to switch from a *single-user* device to a device which can be operated by multiple people. We can say that mobile phones are personal devices and hold various personal contents such as photos, videos, contacts, etc. Viewing this content on the small screen display of phones is acceptable for a single user and for assuring the privacy of browsing. However, when it comes to sharing the viewing with other users simultaneously, the limited phone UI and display fail to deliver acceptable user experiences as those in single-user scenarios. The small screen allows for two or three users to have a clear view of the displayed material without the hassle experienced when trying to locate and move more viewers around the phone to have direct line-of-sight to its screen. Several approaches and technologies have emerged to overcome this *over-the-shoulder* form of interaction. The most noticeable is the use of projectors, whether built-in or externally attached, to allow the projection of content onto any surface [93], allowing thus a multi-user access. Another solution relies on

pairing the phone with a public display to present the content to be shared [94]. Both applications however encounter the limitation of having only one user in control of the interaction with the displayed content. A more conceivable approach is based on integrating the mobile phone with a multi-touch interactive table [95] which receives the content from the phone and lays it on the surface for multiple users to view and interact with; hence, removing the burden of the limited access. The research conducted here focuses on this integration and it utilises Bluetooth, near field communication (NFC), 3D content and 3D manipulation to facilitate it for a group learning interactive experience in the medical domain. The system presented demonstrates how mobile phones and interactive tables can cooperatively work together for more collaborative interaction experience for medical students with medical content on their phones. Unlike most existing systems, it manipulates 3D content rather than 2D ones. The student basically carries this content on her phone and shares it with others over the table to promote a group discussion. She does so by touching an NFC tag on the table with her NFC-enabled phone to identify the Bluetooth address of the server to connect to. Once identified and connected the server receives and displays the transmitted 3D content on the surface of the table. Here a detailed anatomical model of the upper leg muscle and bone construction segmented from MRI scan data has been used. Students are capable then of manipulating the model and its components through different 2D gestures on the table. They can move, rotate, zoom in/out, enable/disable its parts, and change their transparency.

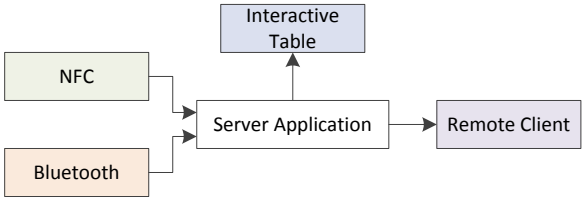


Figure 3.14: Overall work flow of the system.

The OSG graphics library has been used to implement the viewer that renders these 3D models and facilitates its manipulations. This library handles windows-based single-input events through one mouse pointer. It cannot handle multiple input pointers and this is not useful when it comes to interacting with multi-touch tables. This limitation has been resolved here through the use of the TUIO[6] open-source framework as shall be detailed next. A screenshot of the actual application visuals is shown in figure 3.15.

[6]TUIO open-source framework`http://www.tuio.org`

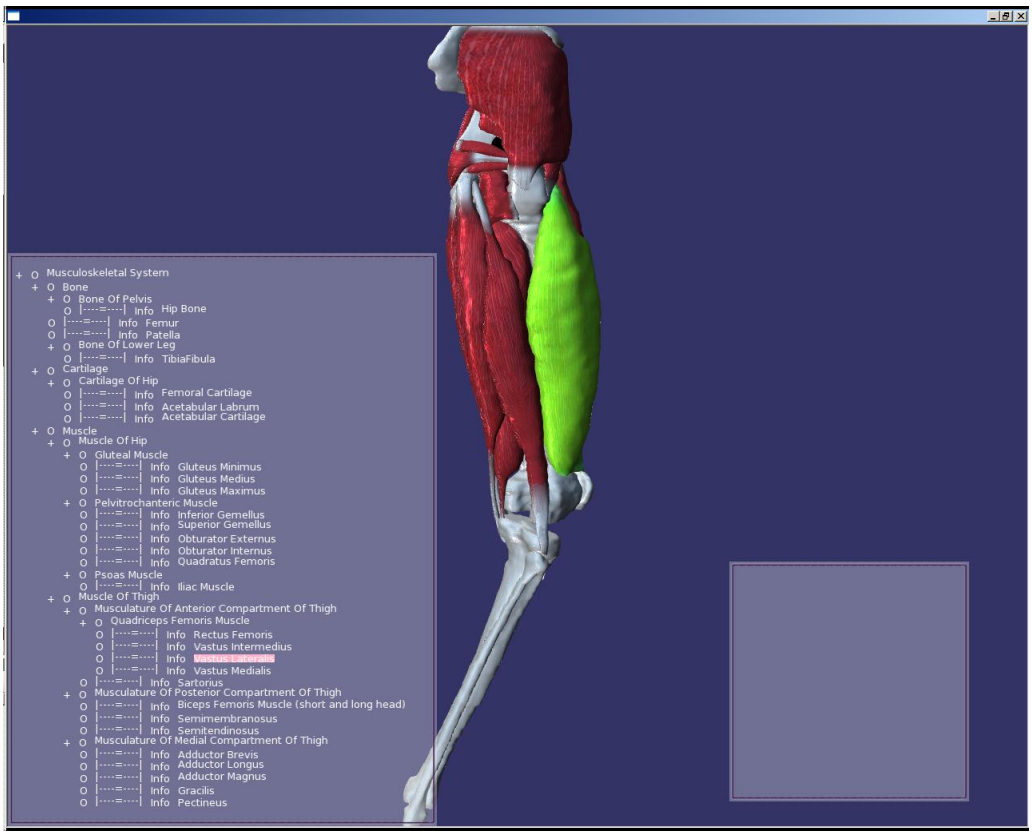


Figure 3.15: The interactive table application showing leg muscles and bones and the corresponding ontology.

3.8 Service Distribution and Render Context Switching

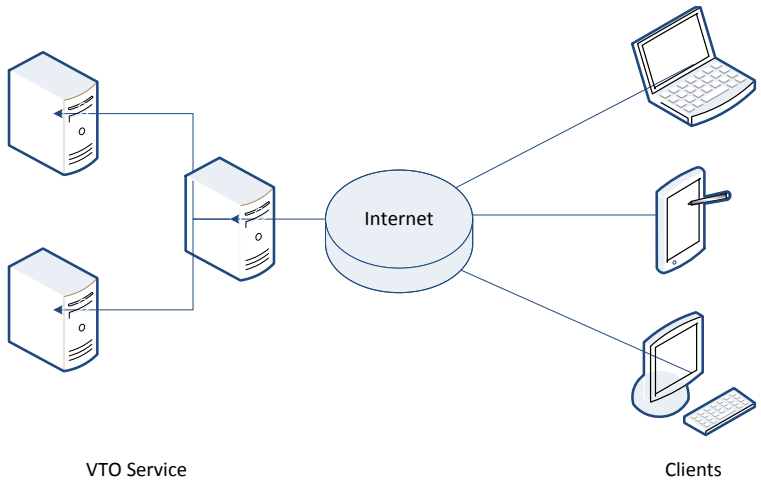


Figure 3.16: The VTO service accessible through the Internet.

In figure 3.16 the concept of the VTO service is given, where on the left side is the providing infrastructure which offers the VTO through the Internet to the clients on the right side. As shown the client devices may differ and therefore have different interface requirements upon

which the VTO service has to reply and provide an adapted interface for the specific device. This is done through device profiles. The service offers two types of visualization types and several stages of adaptation in order to provide an optimized visualization based upon the device and network constraints. These constraints include limited computational capabilities, specific display sizes, network bandwidth and transfer speed. In continuation of the work in [96] this system extends on the ability of switching between rendering and streaming methods. The base adaptation provides static image transfer on request or as a pushed stream of images with variable quality, interval and different image formats are supported. The optimal streaming is based on the VP8 codec[7] implementation and provides streaming of the 3D content utilizing a micro-buffer in order to reduce stutter yet retain a real-time interactive experience. The client

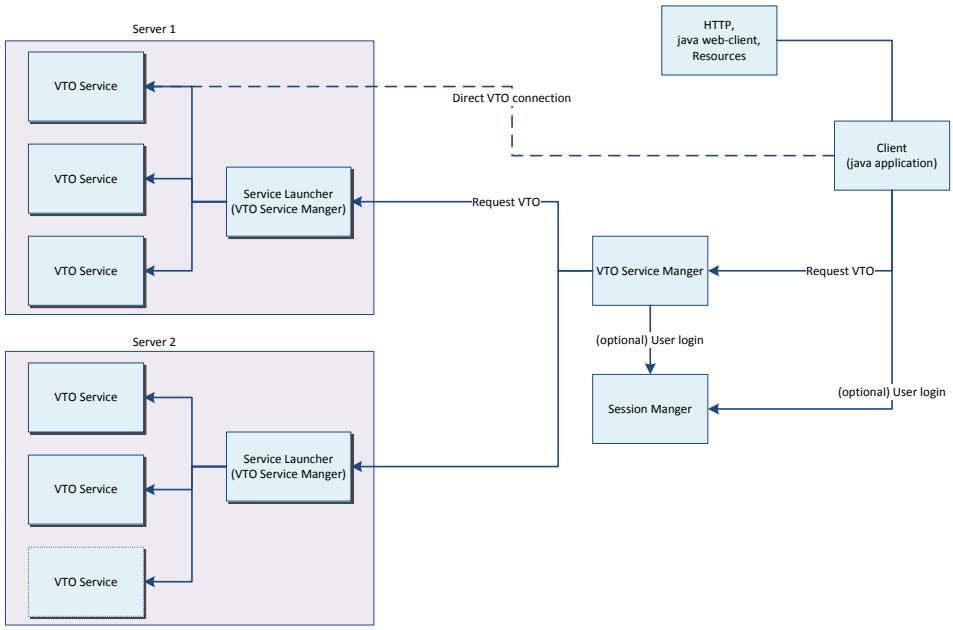


Figure 3.17: Top level services, providing VTO deployment in an on-request load balanced setup.

application is implemented in Java and is available as a standalone application and as an applet and provided by a HyperText Transfer Protocol (HTTP) server and runs within the browser. In both cases it is dependent on Java and Windows environment. The decoding module at the client application is provided as a C++ dynamic library and is loaded through a Java Native Interface (JNI) wrapper. Once the client application is loaded it connects to the VTO service manager and sends a message requesting for a VTO Service. The manager keeps a list of service launchers and forwards the message to one of the Service Launcher using a *least-burdened* load-balancing scheme. The service-launcher than can assign an already active VTO Service back to the VTO-Service manager or execute a new VTO-Service process and return its connection address. For each server there is one Service Launcher and basically is a generic process starter, using inter process communication the Service Launcher keeps track of the spawned services and monitors their activity. The connection information given back to the VTO Service manager is repacked

[7]WebM & VP8 codec web-link: `http://www.webmproject.org/`

into a message to the client, upon which the client directly connects to the VTO Service. The VTO Service than informs the Service Launcher of being in use. The Service Launcher registers the VTO service of being in use, using a time-out and a periodic testing of the state of its spawned services in order to avoid double assignment of users and zombie processes. Optionally, the user can provide login credentials and load preferences from a former session directly. Whereas the Service Launcher resides on the same computer as its spawned processes, the VTO Service Manager and Session Manager can reside elsewhere in the network. In case of service failure, the VTO service process is killed (in case still active) and a new VTO service is spawned with the same access parameters. The client will automatically reconnect (periodic based) to the service and provide the current user state back to the Service upon connection. In figure 3.18 the VTO

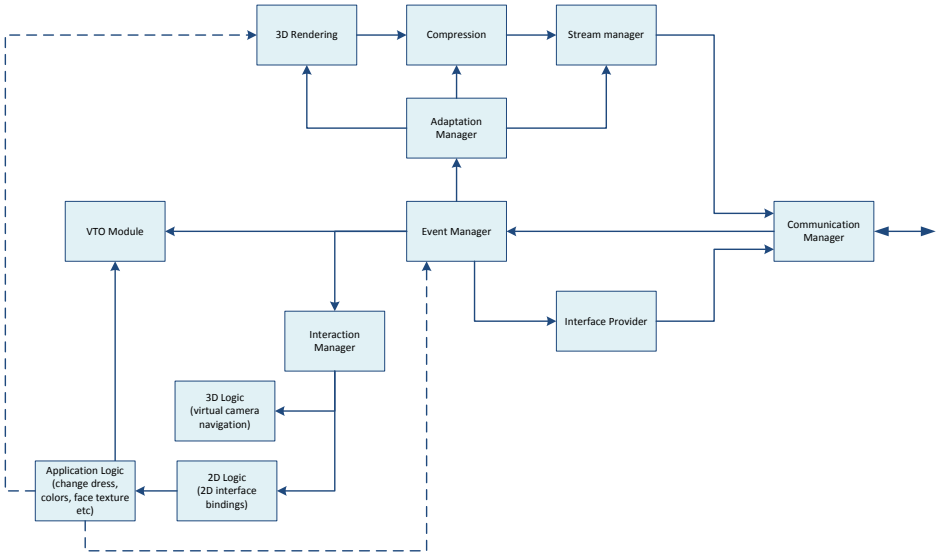


Figure 3.18: Server side internal modules.

Service internal components are depicted. At the base there is the *Communication Manager* which is responsible for marshalling an unmarshalling messages coming from the Service Launcher or the active client connection. These are two separate connections, whereas the connection to the Service Launcher is an IPC based and the connection with a client Socket (using TCP) based. The message is translated into an event object and given to the *Event Manager* which inspects the event and decides how to process it. There are four types of messages, *Adaptation*, *Simulation*, *Interface* and *Interaction*. Adaptation events are messages that come from the client informing the adaptation manager to change certain parameters (these could be frame rate, resolution, compression type and quality). Simulation events are specific functions directly called by the client, often without any user input but rather the client informing the simulation to be set to certain parameters (initial setup, or recovery). The Interface event is a single event requesting the service to send the interface elements. The client provides what kind of interface it needs based on the requested display size. The interface elements are generated dynamically depending on the VTO Module capabilities (e.g. body sizing parameters are translated in to a set of sliders). A basic set of 2D interface elements, such as sliders, buttons and selection boxes, are supported.

Interaction events are separated in two types' 3D interaction and 2D interaction. The main difference is the 3D interaction tends to be a stream of events, e.g. handling the virtual camera and therefore has a different handling scheme. Whereas the 2D interaction is a translation of the client state's 2D interface element to a set of parameters which is provided to the function bound the application logic.

3.8.1 VTO Module

The base of the VTO is similar to what is described in section 3.6, however now the focus lies more on the integration with a distribution system and serving multiple sessions at once instead on the blending of AR. The VTO service internal components are shown in figure 3.19. Starting

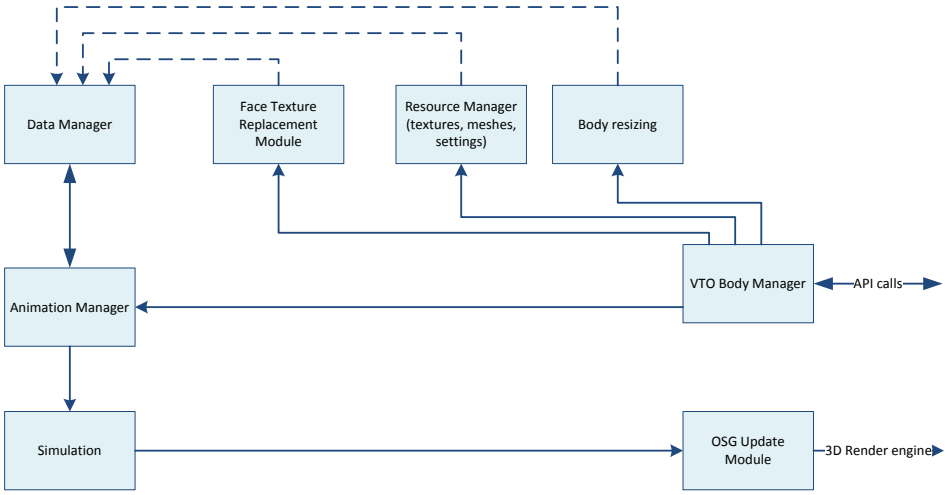


Figure 3.19: Virtual Try-On Service.

with the VTO Body Manager, which grants access to the internal functionalities of the simulation, it's resources and additional functions, such as the change of animation, body sizing and face texture based on a provided image.

3.9 The Collaborative MRI Segmentation

In this section we present a collaborative Telemedicine system for real-time and interactive segmentation of volumetric medical images. Unlike the conventional strict locking mechanism, we use an optimistic approach, relax the temporal and spatial constraints for collaborative editing, where multiple users are allowed to simultaneously update the shared medical images without waiting a lock to be released by a user. To achieve this, we introduce a two-tier sharing architecture [44] with iterative simulation process to avoid roll-back problem [97]–[99] usually caused in the optimistic approach, i.e., multiple users can request updates on the shared data within a time range (simulation time period), where these requests are used to the simulation parameters and let wrong inputs be amended by simulation algorithm in a iterative and progressive manner. In order to utilize the diverse medical resources (data) scattered in the distributed sites, we relax a

spatial constraint in the traditional server-client architecture exploiting the previously described publish/subscribe paradigm [100] with *group* concept.

In our approach, each site can be a server node if it has the medical data to be shared by a group of users. A user can load the medical data, initiate a session and declare it to a multi-session management component. Users can join to the session at any time. A Group concept is introduced to provide functionality for efficiently formulating a group session and interaction among users in a session.

We present two case scenarios, representing the potential usage of the collaborative Telemedicine system, and are evaluated with an user survey to measure the usefulness and the practicability of our system in real situations (section 5.6). The two scenarios are (*i*) An experienced physician is over-looking and guiding the iterative segmentation of an image by one or more trainees (*teacher-students* scenario); and (*ii*) Two or more experts are simultaneously segmenting the same volumetric image data (*expert-expert* scenario).

The collaborative Telemedicine system consists of two architectural layers: the collaboration support layer and medical semantic layer as shown in figure 3.20. The first layer provides mechanisms to enable real-time interactive collaboration among users by relaxing temporal and spatial constraints of the conventional systems. The second layer contains the simulation of medical data and user interface components that controls the simulation.

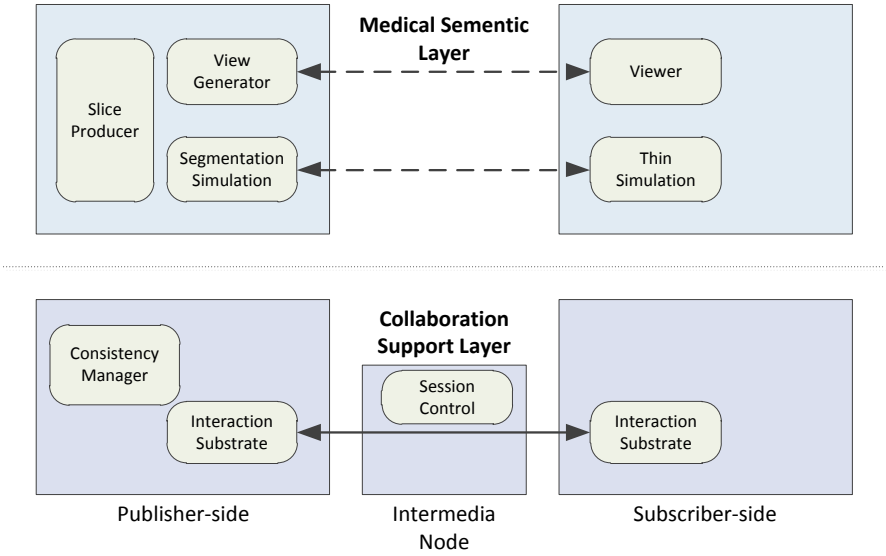


Figure 3.20: Overview of the system architecture. The medical semantic layer (top) handles the simulation and user interaction while the collaborative support layer (bottom) provides the collaboration mechanism.

3.9.1 Collaboration Support Layer

To make users to exploit medical data in their local site for collaboration, which means that all medical data does not need to be located in a specific server but a user who has the medical data can be a host, we use the event-based publish/subscribe paradigm [100]. It clearly decouples

the communication entities but it does not provide sufficient methods to formulate a group for collaboration and to customize interactions among users in the group adapting to application semantics or policies. For this a *group* concept is introduced on top of the publish/subscribe paradigm. A Group is defined as a collection of distinct entities that share the same contexts satisfying constraints, according to: Group = $e|f(e) : constraints$, where e is an entity and f is a context function. An entity is a user or an object in the shared data. Each entity has its own property and can be included in multiple distinct groups. A context function is defined by developers to create a group based on application semantics using properties of entities and external semantics such as device capabilities and network conditions, i.e., a context function can be defined as "users who share the same interest on a specific medical image for interactive segmentation". The Collaboration support layer consists of three main components: Session control, Interaction substrate, and consistency management which utilize the *group* concept to relax the spatial constraint. A session implies a basic unit which is an independent collaboration among users. It provides users with the interfaces for entering or leaving its collaborative editing of medical data and membership management, and defines specific rules applied to a session. It is inherited from group component. We define a session as a group of users who share the same interest on medical dataset. Session can easily control membership using the basic functions of the group (ex, add and remove). Sessions are managed in an intermediate node which runs on a separate machine. Multiple sessions are also supported. For dynamic management of sessions, the multi-session controller holding a reference list of sessions provides users with the interfaces for initiation, termination, selection, join, leave, creation, and deletion of a medical collaboration session. Once a user initiates the collaboration session, i.e., loads the image, declares the new session to Session control component, multiple users can connect to the session and start the collaboration. When one of the users selects a segmentation command, an iteration of the segmentation is executed and the results are relayed to all the users in the session. The results are composed of the image data (slice) and segmentation data (segmentation overlay). At any point, any user can collaboratively segment the image by adjusting the segmentation parameters via interactive image annotation. These parameters are then inserted as new constraints in the segmentation and used in the next segmentation iteration process which will be described in detail in section 3.9.4. To enable users to interact with each other, a set of well-defined interaction protocol is required. Group provides basic functions such as union, intersection and difference, to create a group. Application developers can define their own functions to create a group based on application semantics. For example, to make a subset of members interact with each other, filter function can be defined as well as to broadcast to other groups, aggregation function can be defined as follows: A filter is a method that makes a subgroup from a group, which holds the following conditions.

- If x is a member of a group G, then x is also a member of group B, where $B = f(A)$
- $B \supset A$, where $B = filter(A)$

An aggregation is a method that makes a super-group from groups, which holds the following conditions.

- If x is a member of a group G, then x is also a member of group B, where $B = f(A)$
- $A \subset B$, where $B = aggregation(A)$

Apart from traditional DVE systems [34], [101] which mostly exploited spatial relationship, i.e., region of interest [34], [99], to filter interaction events, in our system, context function can be described with the current contexts of devices, such as network and computational resources, display size, etc. It gives freedom to developers to make filters with specific application requirements. An exemplary context function has to be implemented, which allows users to participate in a collaborative session with diverse devices e.g., UMPC, desktop, without degrading interactive performance of users. One of key issue in consistency management is to provide ways to manipulate the shared medical data concurrently and to hare consistent views among users. The proposed system provides monolithic view as well as polymorphic view sharing among users in a session. The former implies a group of user shares the exactly same view of the shared data and the later implies a group of users has the different views of the shared data but user interactions are shared among users in the both approaches. The proposed system provides a policy-based optimistic concurrency control scheme which allows users to update objects without conflict checks with others and thus to interact with objects more naturally exploiting application contexts. More details are described in section 3.9.3.

3.9.2 Medical Semantic Layer

The medical semantic layer provides the segmentation simulation, rendering of the 3D medical data and the application policies to the collaboration support layer, which allows customizing the interactions among the users and the collaboration algorithms. The core components are the Segmentation simulation, View generator, Slice producer, and the Viewers, as shown in figure 3.20. The Segmentation simulation is encapsulated into a service and continuously provides the deformed mesh after a simulation cycle to its subscribers. The subscribers are the depended services Slice producer and View Generator and from the subscriber side the Thin simulation. The first two only take the output of the simulation and process it further. The Thin simulation on the other hand is capable of sending commands to the simulation in order to modify its parameters. The Thin simulation also subscribes to the Slice producer for the visualization of the current slice. Through its interface, input taken from keyboard and mouse is translated into simulation commands and send to the Segmentation simulation service. The slice producer processes the deformed mesh into a contour plot for subscribed Thin simulations. Whenever a subscriber changes from slice, the service will send the corresponding MRI image to the subscriber, which is then visualized on the subscriber. The collaborative *Viewer*, which provides 3D visualization of the full mesh (segmentation), is separate from the Thin simulation. This is a remote rendered 3D representation provided by the View generator. The View generator takes the deformed mesh and this then renders the 3D mesh. The resulting 2D image is compressed and sent to the subscribers. A secondary service of the view generator is the Interface manager. This sub module provides user interfaces to subscribers adapting to the current device capabilities.

3.9.3 Collaborative Editing Mechanism

To enable real-time and interactive editing over the conventional methods [34] such as *strict locking* concurrency control, two-tiered sharing approach [44]: semantics-tier and presentation-tier sharing, is used as shown in figure 3.21. It enables the semantics (3D Image + 3D deformable

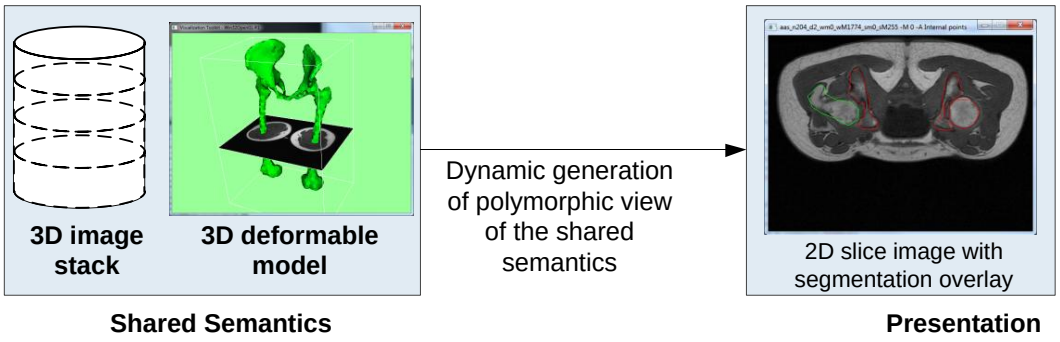


Figure 3.21: Presentation-Semantics split model. Supporting group collaboration (segmentation overlay) on the polymorphic 2D image among users.

models) to be shared and thus can instantiate more than one presentation (2D slice image + segmentation overlay), which is in term, shared among all the users in a collaborative session. A polymorphic presentation of the shared semantics is dynamically generated when a group of users request edits (e.g., new slice position on the stack, B/C change, segmentation constraints) within a simulation process. The generated presentation is replicated to each user of the group, not only to support direct manipulation on the replicated presentation, but also to continuously support the users to collaborate even in the event of a transient network failure. Users can temporally manipulate, update and annotate on the presentation within the simulation process. In our approach, rather than accepting inputs from only a single subscriber within a time frame, multiple parameters (changes) from multiple subscribers are used in the segmentation process. Parameters can be global (e.g., weight coefficients) or local (i.e., local constraints as used in the examples in section 3.9.4. This mechanism allows the users to manipulate objects without waiting to get ownership of the lock to update the segmentation overlay. However, in collaborative editing, it is difficult to identify which input parameters are responsible for the segmentation evolution, especially when the segmentation result is unexpected, due to all inputs from the users having an influence on the segmentation evolution. A common method to track the parameter for segmentation is to implement a rollback mechanism [99]. Rollback is the process of recording all the input parameters which can be used to roll back to the previous input in the segmentation evolution. Nevertheless, it is hard to apply such rollback mechanism because this makes the system complex for the users in undoing and redoing of previous. To overcome this, three strategies are proposed in this study. Firstly, segmentation algorithm must be iterative, thereby enabling different users to amend, stop and resume the segmentation process. Most importantly,

each iteration has to provide intermediate results that provide sufficient feedback to the user for correct interpretation. This provides efficient monitoring of the algorithm evolution. Secondly, changes made by users have an immediate influence in its local view so that the user may receive instant feedback. Thirdly, weight factor policy is used to fuse inputs from users. For example, inputs from an expert have higher weight than inputs from student in our Teacher-student use case scenario.

3.9.4 Multi-user Iterative Image Segmentation

Our segmentation algorithm is based on an iterative and progressive evolution of physically-based discrete deformable models [76], [102], [103]. In our case, these deformable models are represented by 2-Simplex meshes [104] that deform under the influence of *forces*. Each mesh vertex is considered as a particle with mass whose state (position and velocity) is derived from the Newtonian law of motion and the applied forces. At each iteration step, the particles state is updated by an implicit Euler numerical integration. External forces are based on image information (e.g., gradients, intensity distribution) to drive the model towards the desired anatomical boundaries. Conversely, internal forces enforce the mesh geometry to respect smoothness and shape constraints. Shape constraints derive from statistical shape models (SSM) that ensure that meshes can only adopt valid configurations expressed by statistics inferred from a collection of training shapes. SSMs proved to be very efficient and robust in medical image segmentation [105] and have been successfully applied to segment a wide variety of structures (e.g., bone [76], [106], liver [105] and bladder [107]). Our segmentation algorithm allows the simultaneous segmentation of various structures of interest. To cope with models inter-penetration, efficient collision detection and response are implemented. Coupled with a multi-resolution approach (from coarse to fine), a fast and interactive segmentation algorithm is derived. In order to monitor and possibly correct the algorithm evolution, interactive control must be provided to users [108], especially in our context of collaborative work. This is achieved by the means of *internal*, *external* or *frontier* constraint points that deform the mesh so that the points are respectively in the interior, at the exterior or on the surface of the mesh (See figure 3.22 and [66]). In practice, constraint points

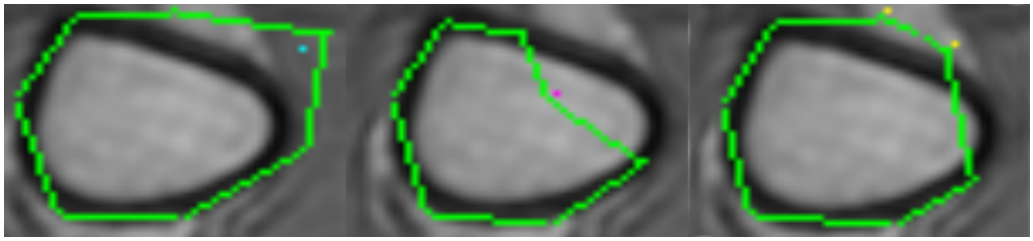


Figure 3.22: Example of constraint points on MRI images. The blue point (right) represents the internal points (marked inside the ROI shape in green, the red is the external, and the yellow points are the frontier points.

attract or repel meshes by creating forces on some vertices. An example of an internal constraint point P is depicted in figure 3.23. The closest face of the mesh, here represented by two vertices

P_0 and P_1, is attracted under the action of two forces f_1 and f_2. Each force f_i is computed as:

$$f_i = \alpha * w_i(P - P_\perp) \tag{3.1}$$

where w_i is the barycentric weight computed from the projection $P_\perp$ of P_p on the face, and α denotes a global weighting coefficient specific to the constraint point type (internal, external or frontier). These external forces have a local influence (closest faces are only affected) while the

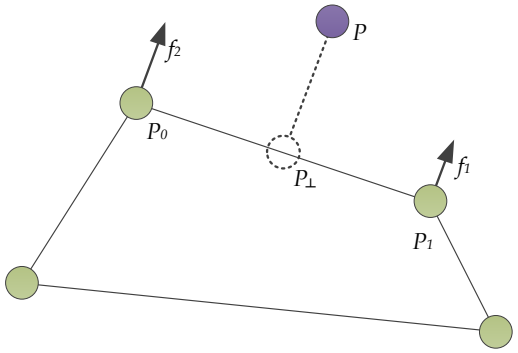


Figure 3.23: Illustration of internal CPs): The closest face (P_0; P_1) to the CP P is attracted by creating 2 forces f_1 and f_2 on P_0 and P_1 respectively, whose calculation depends on P and its projection $P_\perp$ on the face.

modification of the force weight α can globally affect the segmentation since all constraint point forces of same type share the same weight. The next section will explain how such weights can be tuned to account for the various collaborative segmentation scenarios. This segmentation algorithm is thus a good candidate for our collaborative application as it fulfils the requirements defined in section 3.9.3 and allows the concurrent segmentation of multiple structures. In this case, the models contour and the constraint points are overlaid in the slice and represent what was previously denoted in section 3.9.3 as *segmentation overlay* (See also figure 3.21).

3.10 Collaborative Services with shared Data Models

In this application scenario we integrate all aspects of the architecture and present it in three parts, where the first part focuses on the collaborative aspect and sharing data between users, where consistency is high importance. The proposed application is based on generating 3D meshed, using 2D pattern drawing. The second part takes the data output from the first and uses it at a real time basis for its simulation services, together with single access to users for remote rendering of the simulation results. The 2D patterns are transformed into 3D meshes and for each mesh a cloth simulation service is started and starts simulating the given 3D mesh. Any updates to the base 2D pattern, resets the simulation and updates the 3D mesh. Simulation parameters can be changed by outside services that subscribe to it. This involves 3D adaptive rendering and Simulation Control services. Whereas the 3D Adaptive Rendering service provides the rendering either remotely or can be integrated as the client application. The Simulation Control service provides the means to change several aspects of the simulation, depending on the type of simulation the internal parameters can be changed, but also more general parameters

such as the simulation speed and update rate to the subscribers. The third part takes the output from the second part and uses it as a means populate a virtual environment. The output from the second part is the transformed mesh, which than is used either as an NPC or as a user's avatar representation. User can connect to the DVE through different clients that as described in the second part, either integrate th 3D Adaptive rendering or subscribe through an 3D Adaptive Rendering service. This application scenario shows the data propagate and mutation from it's initial creation to it's deployment and usage within a Virtual World.

Interactive performance in terms of responsiveness is one of key challenging issues for remote interactive 3D applications. We introduce a run-time presentation and dynamic interface-adaptation mechanisms which aim to preserve the real-time interactive performance of 3D content, taking into account heterogeneous devices in user-centric pervasive computing environments. To support perceptual real-time interaction with 3D contents, temporal adjustment of presentation quality adaptation is used. In other words it dynamically adjusts the quality of presentation on client devices according to the current device context. As to overcome the inevitable physical heterogeneity in display capabilities and input controls on client devices, we provided a dynamic user interface reconfiguration mechanism for interaction with 3D contents. We then extend these concepts to a Multi-user environment, offering a variety of services which are linked together by their functionalities. The full scenario is shown as a service diagram in

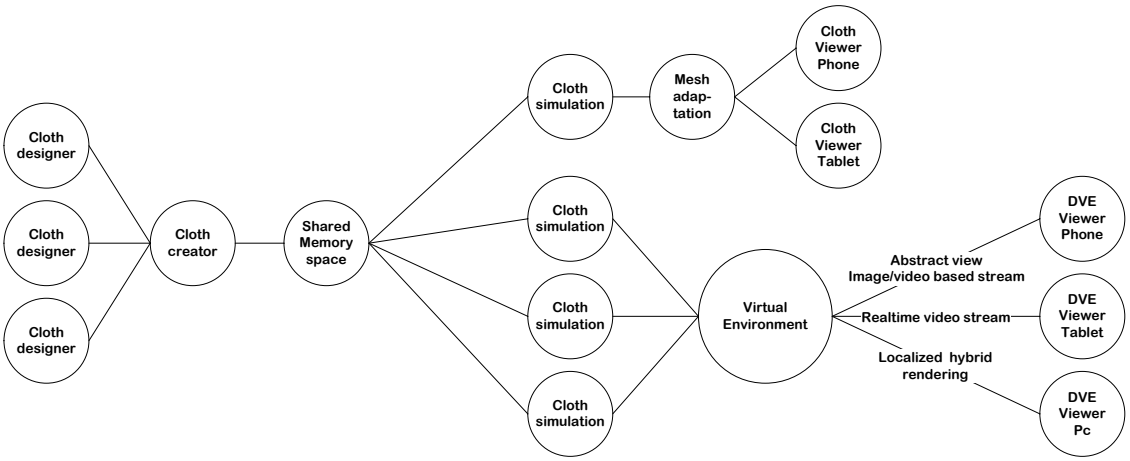


Figure 3.24: Deployment of the architecture.

figure 3.24, where three Cloth Designer client applications are connected to the Cloth Creator, which maintains the connections, consistency and interactivity notification. It either connects or integrates the Shared Memory Space (SMS) service, which basically is a database containing any data a subscriber wants to store in it. In this case the Cloth Creator is the actual publisher to the SMS, and does not directly act as a subscriber. However since the data is shared with the other Cloth Designers, for consistency, depending on the implementation either the Cloth Creator directly notifies the Cloth Designers or the Cloth Designers are subscribed to the SMS and are updated upon data modification. The possibilities of implementation are further explored in section 4.8. The Cloth Creator stores the 2D pattern as well a generated 3D mesh constructed

from the 2D pattern. Then the Cloth Simulation services take the data and start simulating it. However without control, there is nothing much going to happen. Therefore the Cloth Simulation can be controlled through outside services, as shown in the figure these are the Cloth Viewer Phone, Cloth Viewer Tablet and the Virtual Environment services. The Mesh Adaptation, does not control the direct input to the Cloth Simulation, just the output, this can be done through Adaptive Rendering, or LOD adjustments and stream the content to the Cloth Viewer for rendering. The Virtual environment is shown as the biggest module. This merely indicates that is an DVE and uses IM mechanisms to keep the data rate to its subscribers optimal, load balances the VE among its resources and provides scalability. The next three sections explain in more detail the three parts of the application scenario.

3.10.1 Collaborative Shared Workspace

This demonstrator highlights certain aspects of collaboration. Multiple users can interact at the same time, using different kinds of devices and networking capabilities. This demo aims at showing a synchronized view and data, multiple types of collaboration (for user testing), 2D interface adaptation and event handling. The pattern designer is a simple 2D drawing application

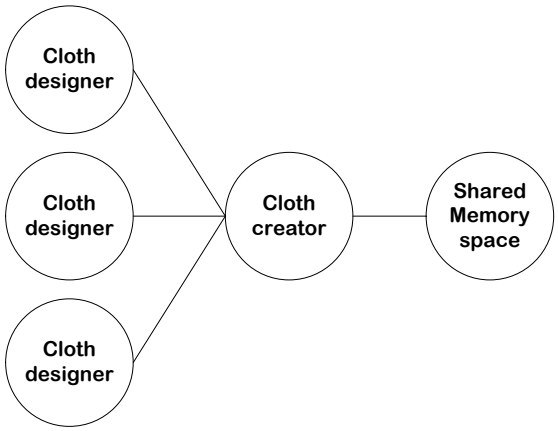


Figure 3.25: Cloth pattern designer modules.

for creating the cloth patterns. Each Cloth designer is a client application communicating with the Cloth creator capable of running on different kinds of devices. The Cloth creator offers the service of taking patterns and triangulates them into the proper cloth format for simulation and rendering. Each mutation done by the clients is echoed back to the other clients, as multiple clients can work on the same pattern. Whenever a triangulation is done by the Cloth creator the client applications are updated with the data in order to visualize the outcome. A command can be sent by any of the client applications to store the pattern and triangulated data into the SMS.

3.10.1.1 Cloth Designer Client

In order to show the several aspects of collaborative environments, we choose to take an constructive approach towards the creation of content that can be used in later stages. In this case the creation of Cloth patterns. This is partly based on the MIRALab's own Cloth creation application

Fashionizer [78]. The collaborative version inherits a subset of functionalities from Fashionizer. Supporting the making of 2D patterns through making 2D line shapes, where lines support a linear or parametric curved relation, as shown in figure 3.26. It is targeted at desktop environments and tablets. Albeit the underlying structure supports the usage of arbitrary sized displays, the focus lies on a working environment (desktop pc) and presenting environment (tablet pc). The modular logic is presented in figure 3.27 and shows a common MVC approach. It keeps a

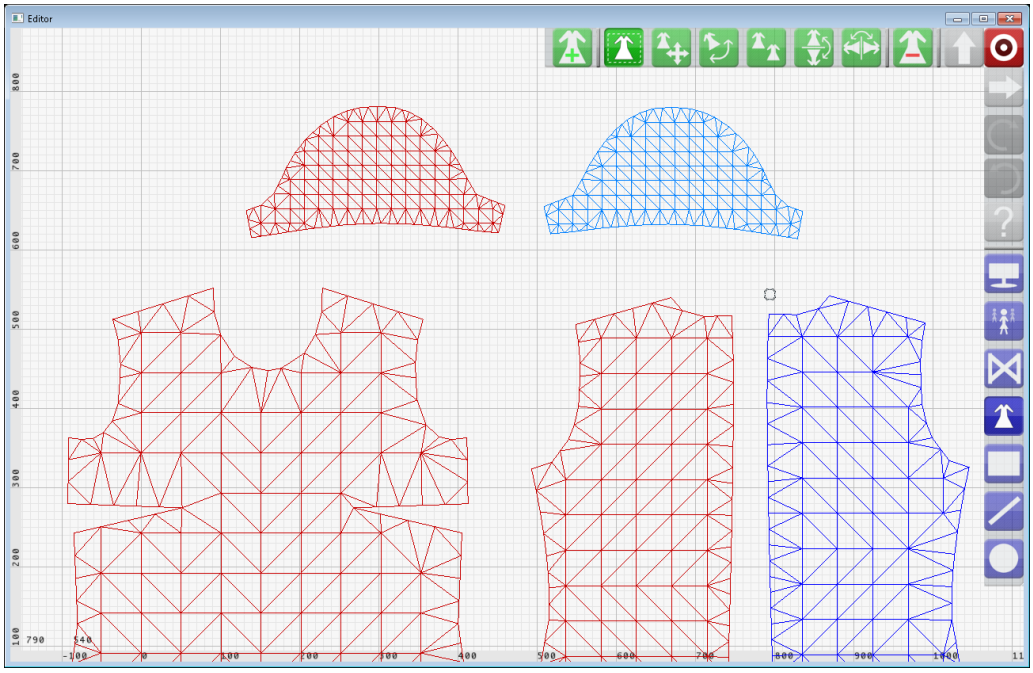


Figure 3.26: Pattern designer.

local copy of the current data set that the user is working and provides functionalities for the interaction. There are two communication channels, one with the *Cloth Creator* and one with the SMS service. Note, that the SMS service could be provided through the *Cloth Creator*. The *Interface* provides core interaction schemes for mouse and touch based input. An earlier prototype will also be discussed which depended more a keyboard and mouse approach, yet this is not cumbersome on a tablet device and therefore a pure visual interaction scheme is used. When looking back at figure 3.2 and consider the layers we can see the following data flows. At the application layer, interaction from the user device is taken and provided to the presentation layer where the device input manager translates propagates it through the context manager to the event manager. Interactions here are interface elements, such as buttons, sliders and other graphical widget interactions, together with client specific interaction logic for handling direct input for pattern drawing. Compression can be used for transferring pattern data with the SMS service, using a pre-specified protocol.

3.10.1.2 Cloth Creator Service

The Cloth Creator Service that monitors 2D patterns in the SMS, and whenever a 2D pattern is changed it transforms the 2D pattern into its 3D counterpart and stores this (elsewhere) in the

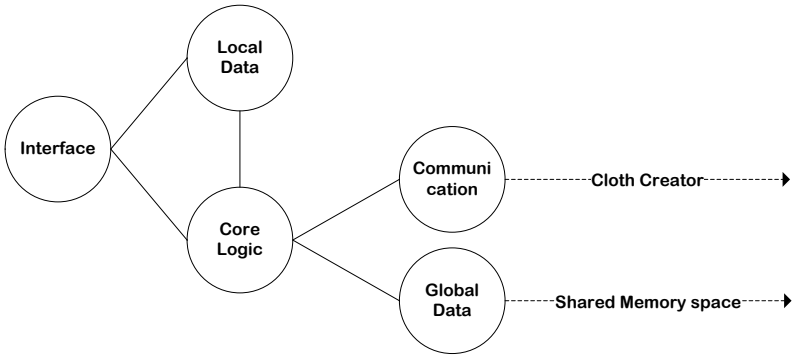


Figure 3.27: Design overview of the cloth designer.

SMS service again. Note that this does not have to be the same SMS service, but can be pre-defined through the use of a configuration script. The Cloth Creator, does not take any direct

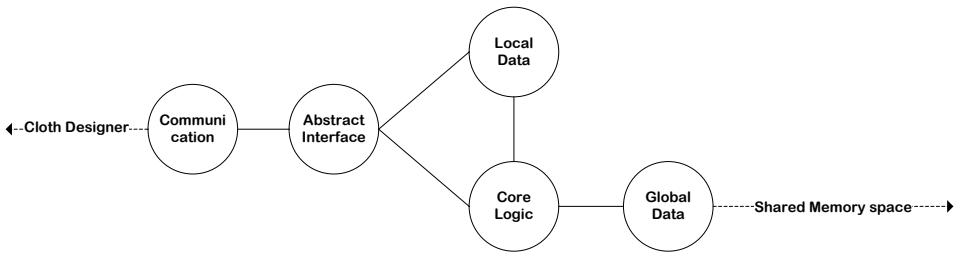


Figure 3.28: Design Cloth Creator.

user input or any other function than just subscribe itself to the 2D patterns and update the 3D counterparts. This is shown in figure 3.28, where we employed the MVC approach. The Model is the *Local data* which is a shadow copy of the data it has subscribed to in the SMS, the Controller is the *Core Logic*, which basically just takes the 2D pattern and transforms it into the 3D mesh and updates its *Local data* and the SMS. The Viewer is the *Abstract Interface*, which is a Networked connection (publish/subscriber approach) which takes protocol pre-defined commands for updating and controlling the data (either local, or redirected to the SMS).

3.10.1.3 Shared Memory Space Service

The Shared Memory Space is a generic module for storing in a publish/subscribe way data. The Idea of the SMS is to keep it very simple to store and share data. Basically the SMS employs a basic protocol for adding, removing and modifying data. The service can be deployed as stand alone or can be integrated into a client/server application. This creates a p2p approach to sharing data. As given by figure 3.29, it takes publishers as incoming communication and subscribers as outgoing communication. It creates a network of SMS modules. The internal structure is a tree of data, forming caches and leaves which contain custom data.

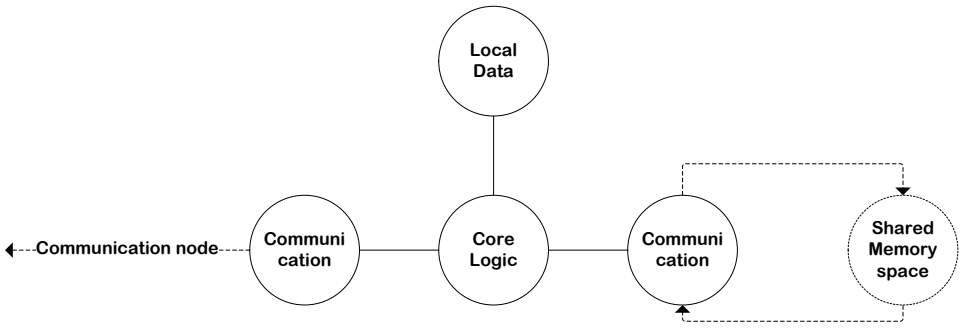


Figure 3.29: Design overview of the SMS.

3.10.2 Remote Rendering Adaptive Rendering

This demonstrator targets at a single user experience for viewing 3D data on different kinds of devices. Target devices are PC (high-end), Tablet (mid-range), Smart-Phone (low-end). The main demonstration is the adaptation of 3D mesh data and optimizing it for the different kinds of devices. A real-time algorithm will check for the most optimal path and provide the data either as raw 3D data for local rendering on the device or as a stream of images (video compression). Interaction events will mostly involve on handling the 3D camera. Next to the 3D interface interaction handling, some extended 2D interface elements will be presented in the form of a material editor. This means the user will be able to change diffuse color, reflective properties and select pre-made OpenGL Shader programs for rendering. The interesting case for using shaders here is that not all devices support Shader programs, so how do we display the mesh. A real-time adaptation switching must occur to handle this. This involves certain user-interface aspects, and concerns usability testing. A Cloth viewer can request for a cloth to be viewed. It will be loaded

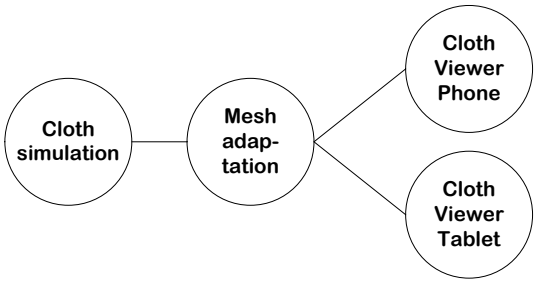


Figure 3.30: Single view on the simulation of cloth.

into a simulator (Bart cloth simulator) which is then connected to the Mesh adaptation service for handling the data stream between simulation and end-device. Streaming of the mesh is currently possible (Java client) and together with the possibility of encoding to a real-time video stream support of any OpenGL shader-programs is possible. The mesh-adaptation module has to be made so that it recognizes the device and application specific state and act accordingly. This involves changing the level of detail for 3D mesh data streaming and adaptation switching between image stream and 3D data stream.

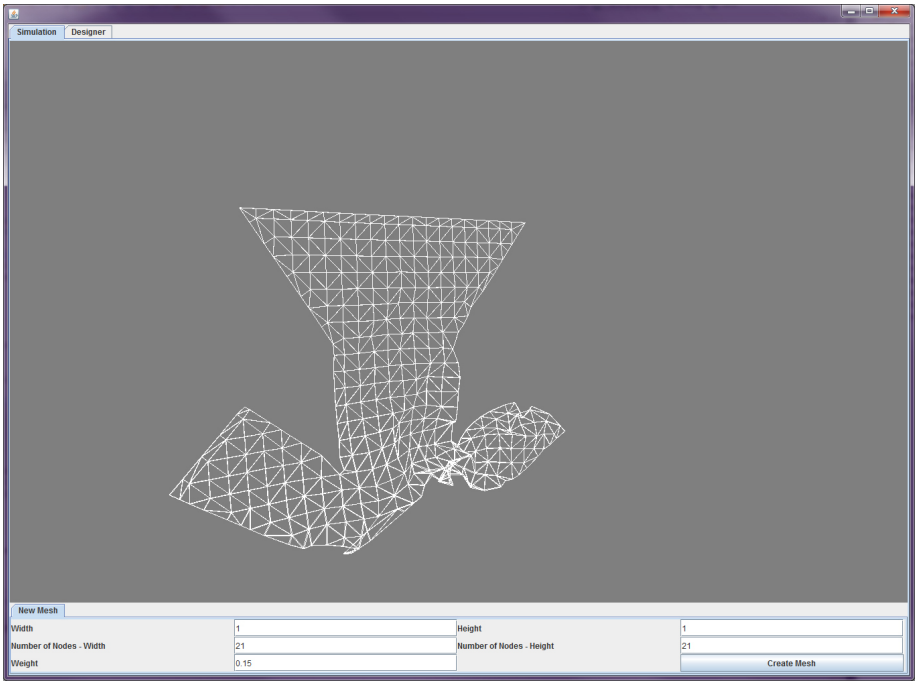


Figure 3.31: Early prototype of the cloth simulation viewer

3.10.3 Distributed Virtual Environment

The last demonstrator incorporates the previous two core functions and extends it to a full DVE. This means that multiple people can roam around in a virtual world, using different devices, where multiple cloth simulations are rendered. Depending on the capabilities of Bart's simulation the DVE interaction scheme will change accordingly. For now it will be showing multiple cloth meshes, real-time simulated, and offer multiple people personalized views and the ability to interact with the simulated cloths. The main component is the DVE, which takes the

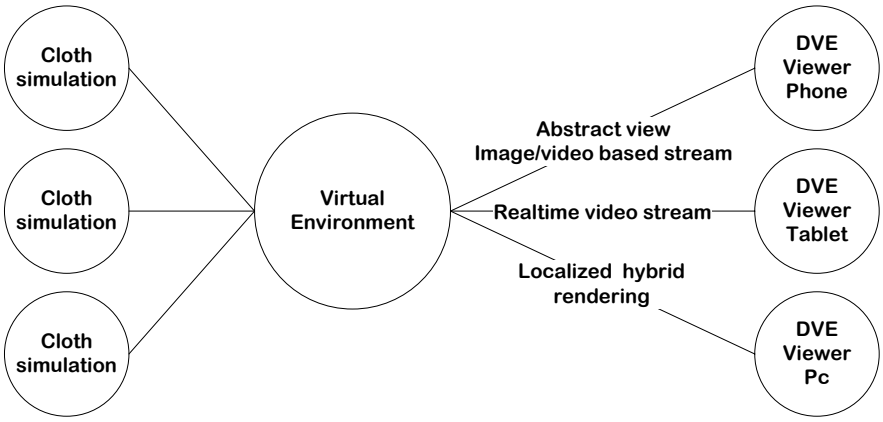


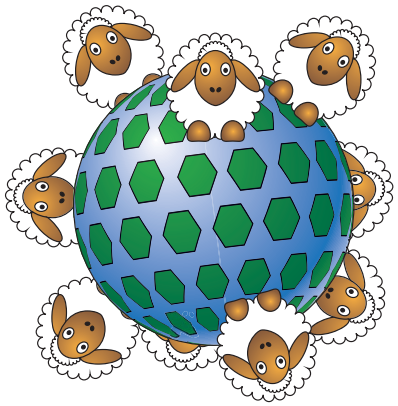
Figure 3.32: Multi-Cloth simulation in a multi-user virtual environment.

approach of common known Massively multi-user virtual environment (MMVE) applications. It can be hosted on several servers based on a known Interest Management filtering - Region based

filtering system. When a user moves around it switches servers whenever entering another region (the users won't notice any difference and happens completely on the background). Each server within the DVE handles the users in his region of responsibility and takes 3D mesh data from attached Cloth simulations. E.g. we deploy a DVE of 3 servers and each server hosts a part of 3D virtual world, where each part contains 10 cloth simulations. Thus therefore each server there are 10 cloth simulations attached. Users connect to one server initially and can see updates for 10 cloth at max. When the user switches to a different region, it will start receiving updates from 10 different 10 cloth simulation (attached to that region). Possible interactions are moving the cloth around. Depending on the progress of the cloth simulation other types of interaction can be proposed. Most interesting here is the adaptation between the DVE and the client device, as again it ranges from Smart Phone to PC.

Chapter 4

Technical Design And Implementation



4.1 Motivation

In order to realize the methods as described in the previous chapter a technical base has to be provided, this technical base however has more restrictive dependencies, such as actual deployment on variations of hardware, operating system, development environment and tooling software. Whereas a method can be generalized and is free from these dependencies, here we must adhere to them and provide solutions that accommodate the targeted method. We go into the details of the design and its implementation of the framework, which is called *Herd framework*. Outlining the core architecture and a set of developed modules which are used in the use-cases.

> *Definition of HERD*
>
> 1. (a) A number of animals of one kind kept together under human control.
>
>   (b) A congregation of gregarious wild animals.
>
> 2. (a) i. A group of people usually having a common bond <a herd of tourists>.
>
>   ii. A large assemblage of like things.
>
>   (b) The undistinguished masses : crowd <isolate the individual prophets from the herd – Norman Cousins>.

[1]

The Herd framework is in essence a middleware, established around a core set of functionalities with a plug-in system that maintains and offers services. Following the design concepts and principles from previous work, the Virtual Human Director (VHD) framework, by from Ponder et al. [109], yet more oriented at the networking aspects and based on top of the concepts of ATLAS [34]. The VHD framework offered a highly modularized approach and had limited capabilities for network rendering using IRIS Performer[110] libraries and output to Virtual Reality Modelling Language (VRML)[2] clients. Where ATLAS is a pure middleware framework with a great focus on VEs, handling a great amount of users and offer interest management methodologies. The Herd framework, like ATLAS, follows a layered approach, as was given in figure 3.2, which always has a trade-off between the freedom of users friendly and simplicity of development[111]. It is the balance between offering low-level functionality and encapsulated functionality that act like black-boxes. With low-level access the developer is granted more freedom in implementing the needed functionalities as desired, which can be more efficient or more coherent to an existing code-base. Higher level functionality provided through APIs restrict this freedom and therefore flexibility. It can greatly speed up development and ease of use, but limits the use for non-intended usages and therefore create obstacles during development. Several fundamental requirements were identified[112] which could efficiently help developers both with high and low level features.

[1] Webster `http://www.merriam-webster.com/dictionary/herd`
[2] VRML `http://www.w3.org/MarkUp/VRML/`

- communication architecture, three types were defined, previously shown in figure 2.6, where the client/server is the most traditional and still most used on the internet offering a simple consistency and security mechanism at the server side. The peer to peer offers more flexibility by decreasing hop delays and single point of failure, but is hard to maintain and security measures has to be applied to all peers. The third is a hybrid of these two, having servers in full data-model replication and clients connected with partial data at a "need to know" basis. With the framework presented here each of these types can be established and therefore capable of adapting to an application needs.

- user membership management, it has shown that in order to maintain a large amount of user to group them into smaller groups for interactive among them, this is widely adapted by MMVE applications. The main reason for this is to filter data unintended for the user, which would normally causing to flood the user, yet it can be quite expensive to filter on a per user basis, therefore groups are formed with common interests. There are many Interest Manager methodologies, that take mostly only VEs into account, dividing the virtual world on certain criteria. We extend on this by including the client device and networking capabilities, as a form of context awareness.

- transmission scheme, the base type of how to send a message to the other side boils down to using TCP or UDP based transmission. Where TCP offers a whole range of features (ordering of data packets, error checking, read as a stream, congestion control, etc.) and UDP is a basic send a message into the net offering more speed. Both have there usages, whereas TCP is most used for multi-user and data-transfers and UDP for streaming of data where speed is favoured over reliability (character movement in a VE or live media streaming). Higher level transmission schemes are placed on top of these and can bring a fine-grained transmission, example protocols are HTTP, FTP, VOIP. The presented framework uses pure TCP and UDP, but offers the capabilities to encapsulate the previously mentioned protocols.

- event management, eventually what transmission schemes accomplish is the pass-through of events containing the data, even if for example TCP establishes a *session* for each connection, it does not control the application. These events are the actual communication and make the application act upon receiving them, however a set of control messages are needed for the entire system to work properly. We differentiate between core control events and application/service control event. The core control events, is a fixed set of events that are used to identify and establish a *session* on application level, to which additional information can be given, such as id login etc. The application/service control events are specific to each application providing a *remote* access to application functionality and extend the core events.

The implementation and design concepts, showing implementation examples as coding blocks and class diagrams given in this chapter reflect the state of implementation and are abstracted where necessary. There always remains a part of implementation trivial to the case presented,

yet we try to take these also into consideration, e.g. the graphical interface to the kernel API. To give a broader view on the capabilities and flexibility at which we aim with the framework.

4.2 The Architecture - The Herd Framework

The Herd System is a framework for setting up a scalable system for offering service that can be concatenated. The base of the Herd System is a node graph architecture maintained by a kernel. Each node is a self-contained module which can communicate with its parent and children nodes. Additional functionality can be implemented within a node. A node can be run in a thread or remotely started into its own process. Extra nodes with newer functionality can be added dynamically through the plugin system.

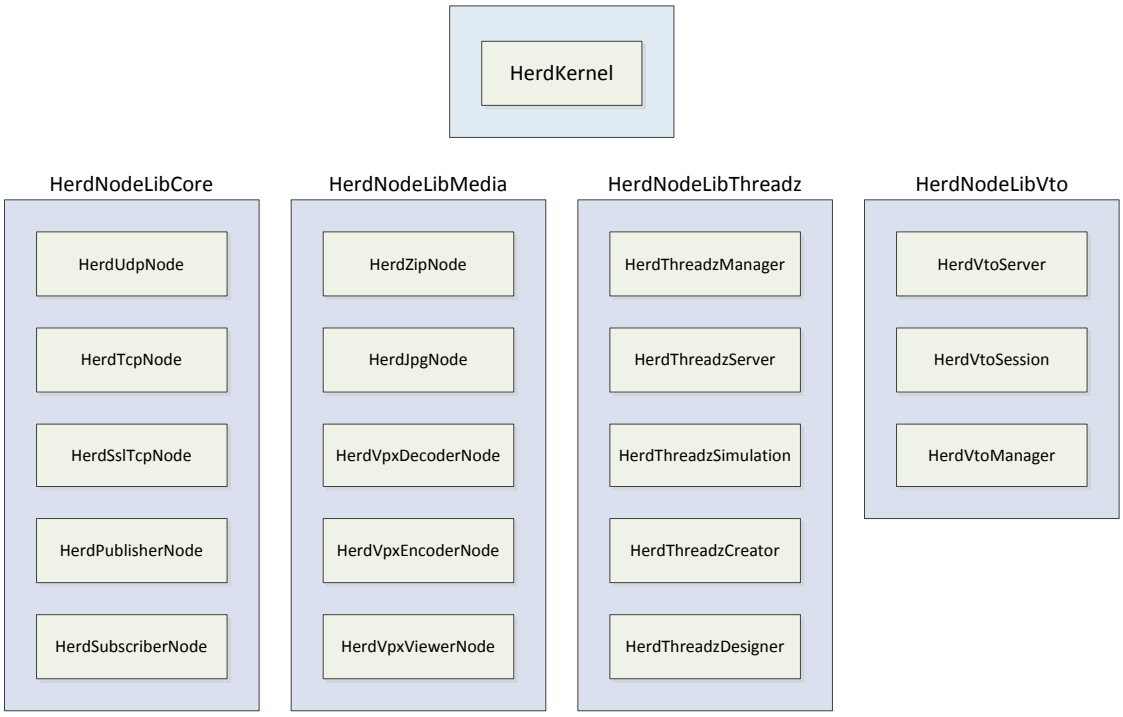


Figure 4.1: Herd Kernel and Node libraries.

Figure 4.1 shows in abstract view the current components within the Herd Framework. On top there is the Herd Kernel, which is the heart of the framework and consists out of a minimal set of management modules and template classes, which are used by the *Node Libraries*. The *Core* Library consists of the most used nodes, such as the TCP and UDP communication and the networked memory cache system. The *Media* Library contains several nodes for encoding/decoding several data formats, either for lossy or lossless data streams. The *Threadz* and *VTO* libraries are specific node libraries created for the discussed application scenarios and are described in their respective scenario section 4.6 and 4.8. Note that it is not mandatory to encapsulate all functionality into nodes, this will be shown by the other application scenarios which use an embedded approach to using the framework and will be discussed in the specific application implementation sections.

4.2.1 Kernel

The kernel is a lightweight management module that offers access to node libraries and load them dynamically at runtime using a plugin system. The internal classes of the Herd Kernel are shown in figure 4.2. The kernel can be divided into three parts, the management by the kernel, loading of plugins and providing of a template for services that can be packed into plugins and loaded by the kernel.

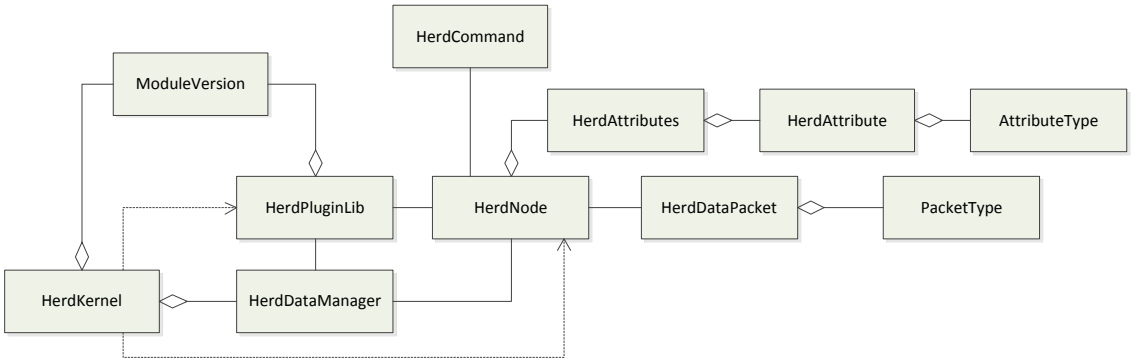


Figure 4.2: Herd Kernel internal structure

The main management class is *HerdKernel*, together with *HerdDataManager*. The plugin class is *HerdPluginLib* and the service template class is *HerdNode*. The template class *HerdNode* contains a partial implementation needed by the *HerdPluginLib* and *HerdKernel* and serves as a base class for extended service implementations. Thus, if we look at all the nodes in the node libraries listed in figure 4.1 they are all derived from the *HerdNode*. The kernel has no specific knowledge about these nodes and only operates on the *HerdNode* class. Through the use of a plug-in system (*HerdPluginLib*), the kernel is able to load and unload specific plug-ins at runtime and using the *HerdAttributes* each *HerdNode* can contain it's own configuration without specific knowledge about the Node functionalities. In the following sub sections we take a deeper look into the plug-in-system, the *HerdNode* class and depending classes will be highlighted and the main class used for data exchange, the *HerdDataPacket*, is given in full. But first the *HerdKernel* class itself.

The kernel as given in figure 4.3 is a straight forward management class for maintaining the libraries, fetching and creating nodes. With several overloaded functions to provide some flexibility in the way plugins and nodes are loaded or created. In order to create a Node a Kernel is mandatory, however it is not mandatory to keep the kernel existing. It is possible to have a node loaded without letting the kernel taking care of (such as clean-up afterwards). Such is the case when using a embedded approach, e.g. an existing application just needs to use a node-service internally, it can create a kernel, create the node and discard the kernel again without losing the node. However the application itself has to clean-up afterwards, which includes the removal of temporary files of the dependent plugin libraries. This will become clear in section 4.2.1.1.

In code snippet 4.1 an example application is given that uses the Herd framework to start a service. A kernel is created, upon creation the kernel directly initializes. Then through the function

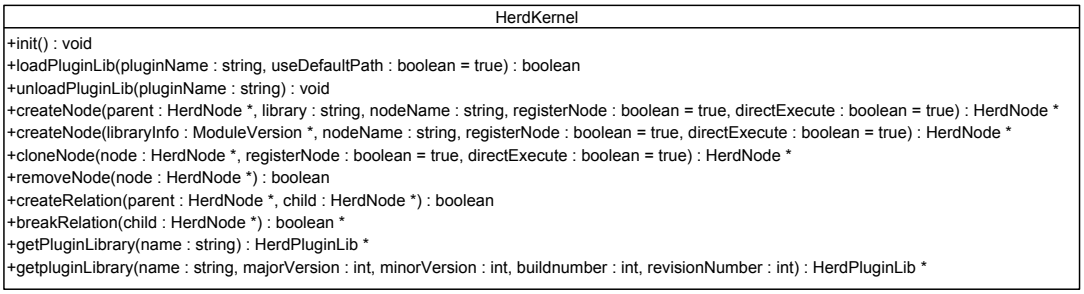


Figure 4.3: Kernel class.

createNode a specific node can be requested. In the example this can be given as a command line parameter thus "application.exe libname nodename" and if no parameters are given it will look into the *HerdNodeLibThreadz* library for node with the name *HerdThreadzDesigner*. As given in figure 4.3 several overloaded *createNode* functions exist with the purpose of flexibility. By default the kernel will look into the executed directory for a *Data* folder and in there for a *Plug-ins* folder. Through the use of the other functions custom library paths can be given. The given *createNode* function here takes five parameters, the first the parent node. As nodes can be linked in a parent-child relation a prior node can be given, however in this case it is the root node, thus providing *0*. Then the library name and the node name are given, and the fourth parameter concerns the registration of the node by the kernel. The kernel keeps track of its created nodes, in order to stop them and clean up after being done with them, however the registration can be avoided by setting the fourth parameter to false. The last parameter tells the kernel to directly execute the node. This is not always wanted, as it can be desired to change certain attributes of the node or might be not needed if the node mainly contains static functionality that has no need for a continues execution, either in a thread or it's own process.

```
1 #include "HerdKernel.h"
2 #include <conio.h>
3
4 int main(int argc, char* argv[]) {
5  string libName = argc==3 ? (argv[1]) : "HerdNodeLibThreadz";
6  string nodeName = argc==3 ? (argv[2]) :"HerdThreadzDesigner";
7  Herd::HerdKernel kernel;
8  Herd::HerdNode node = kernel.createNode(0, libName, nodeName, true, true);
9  if (!node)
10   return 1;
11  while(node->isRunning()) //wait till node is finished
12   Sleep(100);
13  return 0;
14 }
```

Code 4.1: Example application for running a specific node from a library.

In the given example, the application will wait until the service thread is terminated, which will flag *isRunning* to false and the application ends. Upon destroying the kernel, and therefore going through the destructor of the kernel, it will clean up any registered node and plugin data.

4.2.1.1 Plugins

The use of runtime plug-ins has many advantages and is a common usage by a major group of software products use this approach to create a greater flexibility for deployment, easier update and replacement without the majority of the software. A plug-in can be send as an automated update to another *HerdKernel* to be loaded on the fly and start offering services available from the library. Through this process services can be offloaded dynamically to other server-hardware without the need of reconfiguring. The functionality for this is not encoded into the *HerdKernel* or the Plug-in class itself, as this in itself is a service and is embedded into a node.

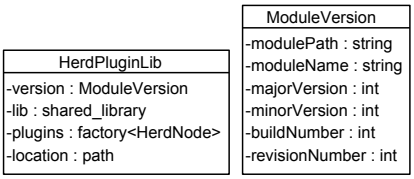


Figure 4.4: The Plugin and ModuleVersion class.

The plug-in class is given in figure 4.4 and uses the Boost extension classes *shared_library*, *factory* and *type_map* to define a shared library. The *ModuleVersion* class is used by the kernel upon loading the library as it extracts the given information from the binary, based on the *VS_VERSION_INFO* data block (this is a Visual Studio resource block).

```
#include "HerdVpxEncoderNode.h"
#include "HerdVpxDecoderNode.h"
#include "HerdVpxViewerNode.h"

//boost
#include <boost/extension/extension.hpp>
#include <boost/extension/factory.hpp>
#include <boost/extension/type_map.hpp>

BOOST_EXTENSION_TYPE_MAP_FUNCTION
{
 using namespace boost::extensions;
 std::map<std::string, factory<Herd::HerdNode>>&nodeFactories(types.get());
 nodeFactories["HerdVpxEncoderNode"].set<Herd::HerdVpxEncoderNode>();
 nodeFactories["HerdVpxDecoderNode"].set<Herd::HerdVpxDecoderNode>();
 nodeFactories["HerdVpxViewerNode"].set<Herd::HerdVpxViewerNode>();
};
```

Code 4.2: Example code for creating a shared library.

In the code snippet 4.2 an example is given for constructing a shared library. In this case the *VPx* nodes are taken into the shared library, as each node header is included, followed by the specific *Boost* headers. By using a combination of a *type_map* set, with the node name as the key and the factory class signature. It is possible to use the node name to fetch the factory for creating the specific node. For a full overview on the use of *shared_library* implementation see the boost

extension documentation.[3]

We take advantage of the plugin system in order to dynamically spawn processes from these libraries. It opens many possibilities such as sending the appropriate plugin over the network and have it directly launched without the main program having to restart, granting the setup of a flexible and scalable system. Further more, upon launching a plugin-node, the dependent plugin libraries are copied first into a temporary directory and the actual node is launched using the temporary plugins. This has the benefit that launched node has its *own* in-use plugins, and newer version of the plugin can be uploaded and replace the old ones, without stopping current running nodes. The idea behind this was a dynamic and autonomous upgrade system. Where a node could store a state of the node, shut-down and relaunch with the new version. The new version then should take the state and convert where necessary to it's own active state. Albeit this is very possible with the given implementation it is very heavily application dependent. That serializing its state and upgrading into the new version can only be given as a concept instead of a fully automatic implementation.

4.2.1.2 Node

The Plugin libraries are the containers for the nodes, and the node is the container for the actual service. Where the plugin is ultimately a plugin file (e.g. Windows a dynamic link library (DLL), Linux/unix based systems a Shared Object (SO) file).

The node is the basic hull for a service and offers several methods for execution and exchange of data with *outside* program logic.

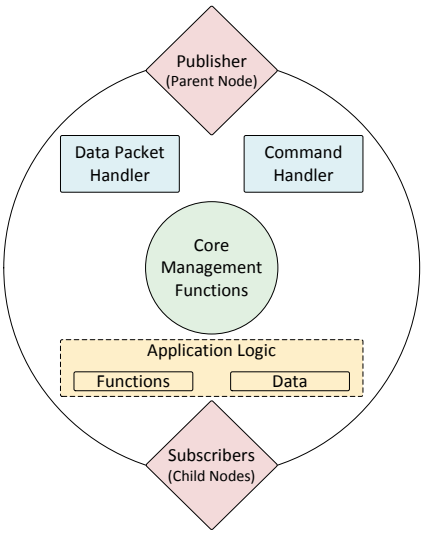


Figure 4.5: Node design in the Herd framework.

The skeleton class diagram is given in figure 4.6 and shows the relevant public and protected functions. An implementation based on the *HerdNode* class can be a *static* collection of functions

[3]Boost extension `http://boost-extension.redshoelace.com/docs/boost/extension/index.html`

that through the use of commands and the corresponding data packet can manipulate data or may be deployed as a dynamic autonomous process.

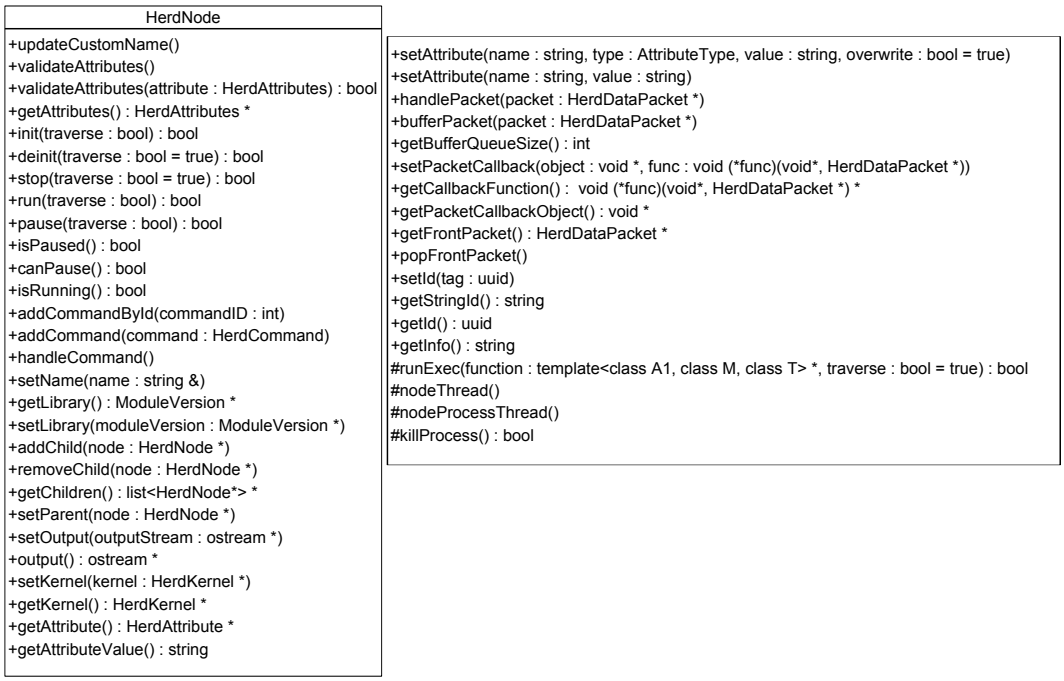


Figure 4.6: Node class.

By default a node has one thread running, the command thread and has two inputs, *addCommand* and *handlePacket*. The *addCommand* takes *HerdCommand* objects, which will be put unto a queue. The execution of the commands by default is done in the running command thread which is calling the *handleCommand* function. Another function special to the command execution is the *addCommandById* function. Which makes it possible to have a factory approach to the creation and adding of commands. The benefit here is that an outside party only needs a shallow protocol on the execution of commands, through simple events. The *handlePacket* is a special function that takes a *HerdDatapacket* as its input. However the node does not know from where is comes, either top level application logic, parent node, child node, from internal node logic, or provided by the *HerdCommand*. It just handles the packet. The packet itself may contain some information on its origin, but it is up to the implementation on top of the node that decides on what to to with it. The output from the node, other than direct function calls to other parts in the program, is done by providing a *HerdDataPacket* object, which it can push to its own *handlePacket*, a callback function or stack it upon a buffer. As mentioned before, by default the node thread is the main execution thread and its base implementation is a loop-structure, which continues while the flag for being active is set. Within the loop-structure the *handleCommand* function is called in order to wait for incoming commands, and as a result to be executed.

In order to start the thread the function *run* is called, either by higher program logic or the HerdKernel object upon creation. The base implementation is shown in code snippet 4.3 and consists out of a single line, pointing to the internal function *runExec*, which is a special function that takes a function pointer.

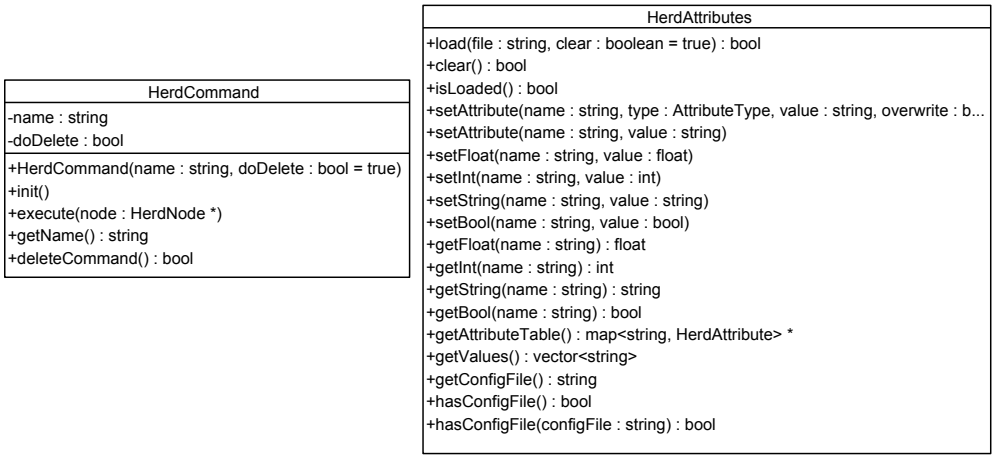


Figure 4.7: The Command and Attributes class.

```
bool HerdNode::run(bool traverse) {
 return runExec(&HerdNode::nodeThread, this, traverse);
}
```

Code 4.3: The Run function within the Node class.

There are two advised ways to deviate from this approach by overriding the specific function (*HerdNode::run*) and provide a custom function pointer to the protected function *runExec*. Or by overriding the *nodeThread* function. The former version gives the possibility to start a second thread to run the *nodeThread* function, for handling incoming commands while the custom function handles the main thread loop. If two threads are not needed than the best way is the latter modification and incorporate the command handling functionality.

4.2.1.3 DataPacket

The data packet is the base component for exchanging messages and data in general between nodes. It had been kept with minimal functionalities providing get and set functions for most common primitive data types that exist across several programming languages. Albeit depending on the language some data-types may have a different notation or lack of it, in those cases (e.g. Java primitive type conversion in code 4.4) it is often possible to *upgrade* the primitive type to a bigger type that is capable of representing the same value.

```
public int getUShort() {
 return m_dataBuffer.getShort() & 0xFFFF;
}

public void addFieldUShort(short data) {
 int i = data & 0xFFFF;
 m_dataBuffer.putShort((short) i);
}
```

Code 4.4: Java unsigned short (2 bytes) conversion to a (signed) int (4 bytes).

In figure 4.8 the class with all the functions are shown. Upon creating a data packet, it has three different constructors, the default, second with a size parameter indicating that the buffer

should directly allocate a certain amount in memory and third the copy constructor for cloning the object. The internal data buffer is split into header and body, where the header contains basic information on the body part. The functions *encodeHeader* and *decodeHeader* are used to write dedicated variables within the *HerdDataPacket* object into the header of the data buffer and vice versa. Several functions provide access to the data, first of the *addField* and *get* functions that provide write and read functions for the primitive data types, other means of access are by *char* pointers to the buffer. These are *getData* which points directly to the beginning of the buffer, *getBody* points to the first byte after the header and thus the beginning of the body part and the *getRead-/writePosition* which are dynamic pointers. The *getReadPosition* pointer increases every-time a *get* function is called, e.g. *getUShort* increases the offset with 2, as a ushort is 2 bytes. The same applies to the *getWritePosition* which increases with every *addField* function. Other functions provided offer additional data to the packet, such as the Id of a packet, which can refer to a user or session Id. The *toSend* function sets a boolean marking it for sending, which actually also indicates the direction of the packet. Since the packet is given to a node by the *handlePacket* function, it does not know the intended direction, this will be more clarified in section 4.2.3.1.

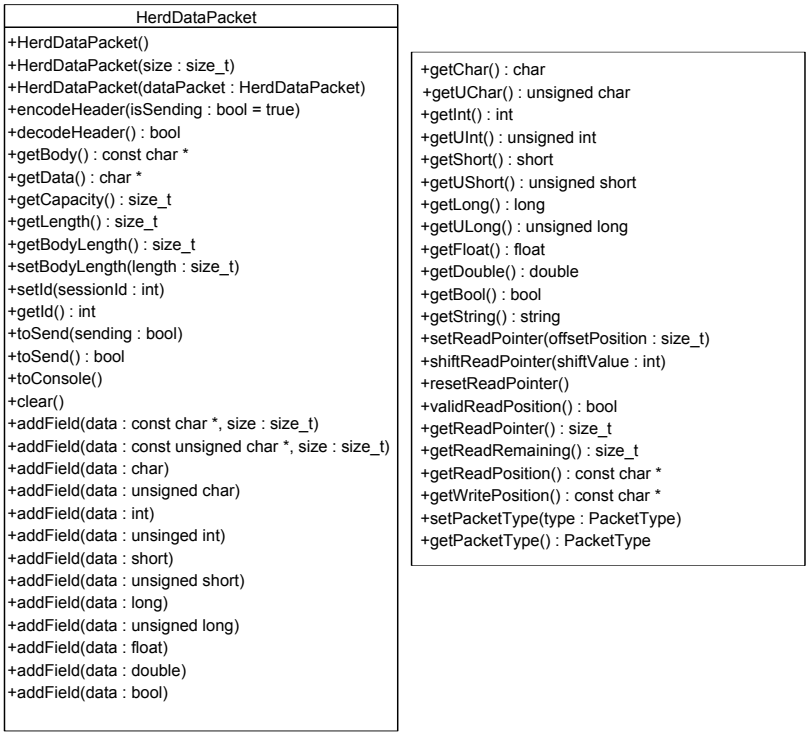


Figure 4.8: DataPacket class.

The implementation is intended to be flexible and open, as there is direct access to the data buffer, which however implies that it is the programmers responsibility to handle this with care. There are *guidelines* on its intended usage for example the use of the read and a write pointer index. Whenever a *get-* function is called the read pointer is shifted according to the amount of bytes read. Same applies to the write pointer upon executing an *add-* function. This however is purely for convenience and can be overruled if needed, for example the buffer allocation can be set to a certain amount, the direct pointer to the data taken and used for direct *mem copy*

Table 4.1: HerdDataPacket structure

Header		Body
Size (4 byte)	Type (1 byte)	Data (max 4294967295 bytes)

Table 4.2: HerdDataPacket supported data types

Bytes	Min	Max	C++	C#	Java
1	−128	127	char	sbyte	byte
1	0	255	unsigned char	Byte	short
1	*false*	*true*	bool	Bool	boolean
2	−32768	32767	short	Short	short
2	0	65535	unsigned short	Ushort	int
4	−2147483648	2147483647	long	Int	int
4	0	4294967295	unsigned long	Uint	long
4	$1.175494351E-38$	$3.402823466E+38$	float	Float	float
8	$2.2250738585072014E-308$	$1.7976931348623158E+308$	double	Double	double
*	0	Max packet	char*/String	string	String

operations. As long as the data packet header is set correctly it is handled as a valid packet by *nodes*.

As given by table 4.1 the header is in total five bytes and the body itself can be arbitrarily long (up to a theoretical 4GiB, as it is the maximum value for an unsigned integer data type). The packet type is set to a *system protocol identifier*, which is one of four possible values, *System, Node, Client, Merged* type. Initially they represent no true meaning, but are intended for basic event filtering, where *System* packets are handled internally by the service node, the *Node* event for a group, the *Client* for an individual and *Merged* indicates that the package is containing multiple packages that have to be unpacked first. These types of events are based on the way the TCP node was constructed and how service nodes in concatenation are developed (see section 4.2.3.1).

In table 4.2 are the different supported data types listed with their values and given for several programming languages. Notice that Java does not support unsigned values and therefore the return type is a primitive data type that can encapsulate the full range (4.4).

In code snippet 4.5 an example is given on how a data packet is created. In the example the packet is instantiated and is followed by a *Clear* function call. This is not needed for a newly created data packet and serves as illustration. The clearing of data does not resize the buffer, merely sets the written volume to zero. An unsigned short (two bytes) is added and a string with a length of fifteen bytes. Then the *Encodeheader* is called, to write into the data packet header the size of the body, which is seventeen bytes $(2 + 15)$. After which it is given to another node into the *handlePacket* function (for example an TCP node which will send the packet to all its registered sessions.

```
DataPacketPtr packet = HerdDataPacket::create();
packet.Clear();
unsigned short packetType = UserProtocol::CUSTOM_PACKET_ID;
packet->AddField(packetType);
packet->AddField("Adding a string");
packet->EncodeHeader();
m_serverNode->handlePacket(packet);
```

Code 4.5: Creating a data packet.

This makes the use of the *HerdDataPacket* extremely flexible in how the developer wants to keep data.

4.2.1.4 Graphical Manager

On top of the kernel manager there is the graphical version of the manager, providing a visual tool in managing, loading and setting attributes to node services. We briefly show the usage of this interface as it merely is the higher level functionality for maintaining services. In figure 4.9 the main screens are shown, the *log*, *add node*, *libraries* and the *main* -screen. The log is a reroute of the standard output stream. It is divided into two parts, the upper for normal messages and the lower for error messages. The *add node* screen shows the loaded libraries and the nodes they contain. Upon selection a node its attributes are read, however the node itself is not instantiated. Attributes can be freely changed and are remembered during the time the *graphical manager* is active. The node will be added to the *main* screen when the *OK* button is clicked. The *libraries* screen shows the library nodes found in the default directory. The icon to the left of each library indicates if it is loaded or not. Details can be displayed in the text area on the right when selecting a library (version number, node names etc.). The *main* screen shows which nodes are loaded (instantiated objects) and the icon to the left of each nodes shows it's current state (running, stopped, pause). Since nodes can have parent-child relations the nodes are shown as a tree when applicable. Note that it is up to the developer to go along these lines as nodes can be created without being registered by the kernel or set at being a child/parent member of another node. Through this interface nodes can be easily added, removed, started, stopped, paused and reconfigured by its attributes.

The implementation of this interface uses the Qt SDK in support for desktop platform independent compilations.

4.2.1.5 Process launcher

The *HerdNode* has the internal ability to spawn a new process of itself, in which its intended service will be executed. In order to do this we use the Qt's *QProcess* for calling a small application that will spawn the desired node. The standard input and output from the application are linked with the *QProcess* and provides our main communication method with the node. Through the standard input/output streams we use the *HerdDataPackets* to send over data. To initiate a node to run in it's own process the boolean attribute *Process detach* is set to true. Upon running the node it checks the boolean and creates a *QProcess* object. Which will run the *NodeProcess*

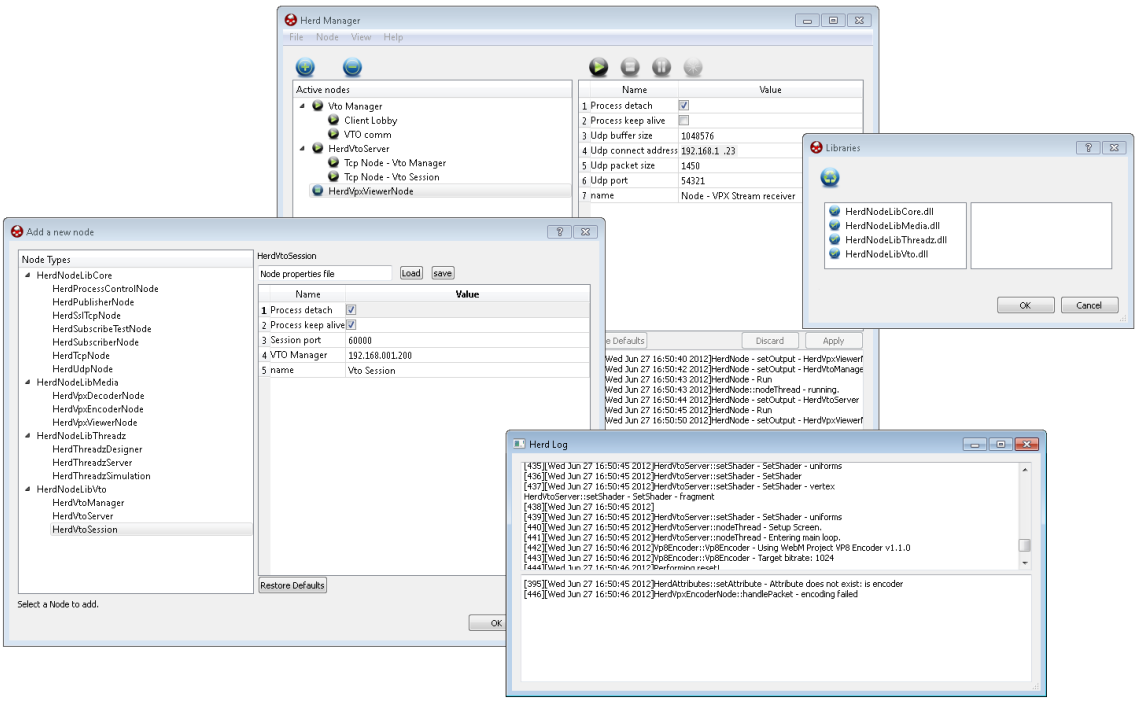


Figure 4.9: Graphical interface for managing the Herd framework.

executable. The *NodeProcess* will start reading from the standard input (thus from the *QProcess* object). Incoming data is treated as *HerdDataPacket* organized data, by first reading the header and decoding the header for determining the expected packet body size. Than read for the amount of data of the body. When the specified amount of data is read the *HerdDataPacket* is complete and read for instructions. These are predefined by a protocol, create node, quit process and new command. A packet with the *create node* identifier contains all additional information for the creation of the intended node. These are library name, node name, the full version identifier and the set of attributes for configuring the node. The *quit process* identifier speaks for itself, any active node will be terminated and the process will exit. The last identifier, *new command*, provides commands to the node by packing the actual data packet. A new packet is created from the data and given to the intended node through the *handlePacket* function.

4.2.2 Plugin Libraries

The *plugin libraries* are the previously mentioned *node libraries* as given in figure 4.1. In this section we take a deeper look into each of these libraries and the given nodes. From code snippet 4.2 each library is constructed in the same way, the plugins are however platform dependent. For each platform the library has to be compiled in order to be loaded properly for the given platform. The implementation supports this by using development tools that are not specifically dependent on platform implementations and providing a unified approach to loading the plugins. The core basis is developed with C++ (hence compiling platform specific), however partial implementations are also provided with different programming languages, such as managed code languages as C# and Java. These are mostly used on the client side.

4.2.3 Core Plugin Library

The core plugin library contains several nodes that are used by most of the applications and therefore form the basic building blocks for it. These include the network communication and memory sharing nodes. An abstract overview of the two network communication nodes (TCP and UDP)are shown in figure 4.10.

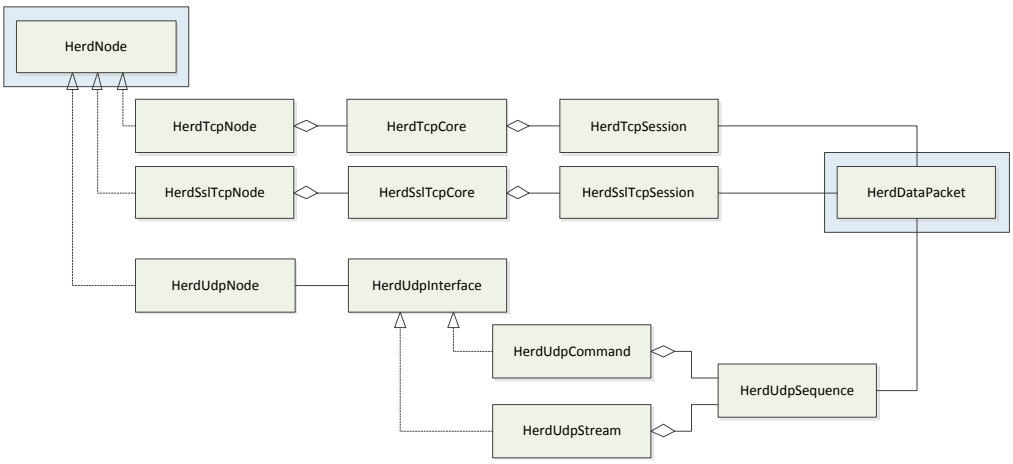


Figure 4.10: TCP, SSLTCP and the UDP communication nodes diagram.

Utilizing the communication nodes for sharing data between services/nodes provides only a very generic way of accessing shared data and is not tightly coupled in terms of updates since everything has to be explicitly done by the service. Which is fine for general data transmissions such as input events and streamed data, but not for data models where the state of the model have to be synchronized. For this we introduce a sharing methodology based on the publish/subscribe. An abstract overview is given in figure 4.11, showing two nodes a publisher and subscriber. Both of these maintain a set data, which is atomized into *MemCaches* and *MemObjects*, each of these can be updated individually. The benefit of this approach is that it can be easily coupled with an existing data model. Either by using the cache structure as a shadow copy of the data, or a direct extension on the data model as will be shown in 4.2.3.3.

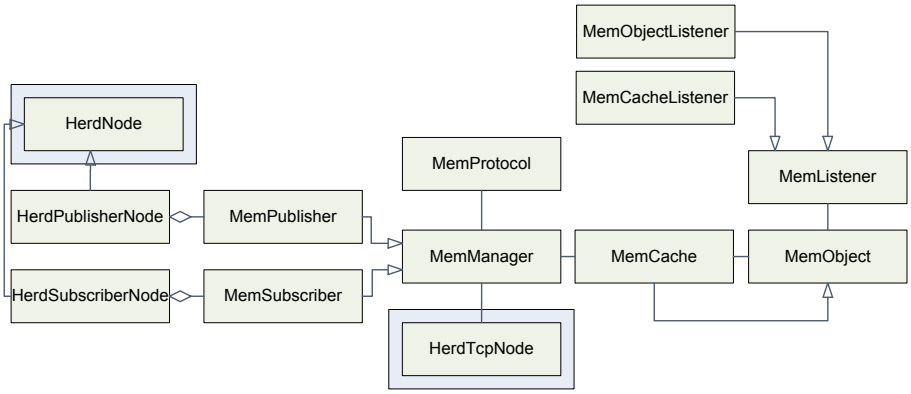


Figure 4.11: The publish subscribe components.

Albeit the figure indicates that the publish/subscribe service explicitly uses a *HerdTcpNode*, internally it is referred to as a *HerdNode* and therefore could be replaced by any other node.

4.2.3.1 Transmission Control Protocol node

For most of the services the TCP connection is used for data exchange over network or as interprocess communication. During the design and development stage of the *HerdTcpNode* it laid down the foundation for how the *HerdNode* itself operates. The node encapsulates the functionality that a TCP socket offers, but provides additional functionality for managing sessions and event control through the use of *HerdDataPackets*. It therefore also solely expects *HerdDataPackets* as input, this means it will not recognize any other structured data. For example HTTP or file transfer protocol (FTP) should be encapsulated into their own respective nodes, ultimately also using raw sockets for communication.

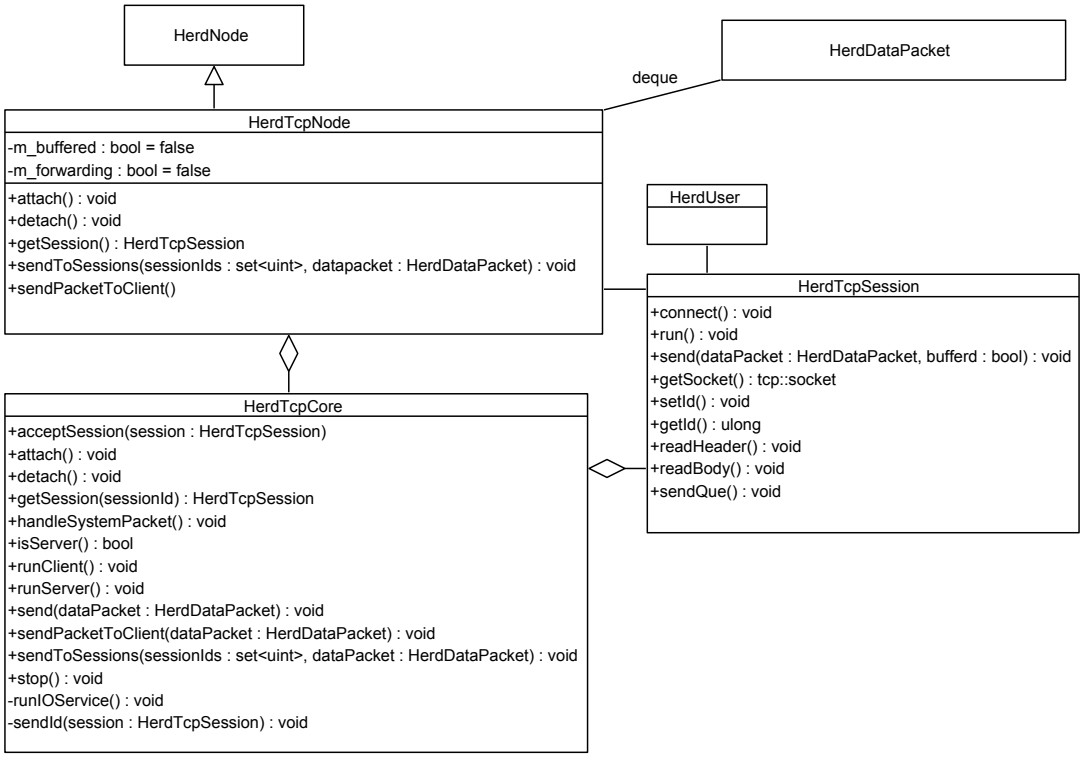


Figure 4.12: Abstract Tcp class diagram.

Internally the node generates *HerdDataPackets* objects from the data incoming from the network-stream, which are then forwarded to the *handlePacket* function for further processing, these can be either forwarded, buffered or provided to a callback function. It is the same *handlePacket* that also handles the packets coming from the outside (e.g. parent nodes). Which can be handled for the same purpose, but mostly for sending over the network to connected clients (TCP sessions). In figure 4.12 the core classes are shown with functions, where the *HerdTcpNode* is the main access point. Note that in most cases after creation of the TCP node, it is being accessed by its base class *HerdNode*. Data is send by using a *HerdDataPacket* object and provide it to the *handlePacket* function as illustrated by the code in 4.5.

In order to use the node, several attributes have to be set. An example is shown in code 4.6, where initially the node is created (line 2), followed by a set of *setAttribute* calls. The TCP socket port number is set to 8080. The destination is in this case an IPv4 address *127.0.0.1*, which is the *local host*. Forwarding is set to false, which means that events received are handled by the node, otherwise if set to true, the node acts as a bridge and forwards the packet to all other connected sessions without processing it itself. The attribute connect indicates the direction of how to establish a session, in this case it is this node that is going to connect to another TCP instance using the local host and port number *8080*. If set to false, the node will not initiate any connection directly but waits for other TCP instances to connect upon which a session is created.

```
1 Herd::HerdKernel kernel;
2 auto node = kernel.createNode("HerdNodeLibCore", "HerdTcpNode" );
3 node->setAttribute( "port","8080" );
4 node->setAttribute( "destination","127.0.0.1" );
5 node->setAttribute( "forwarding","false" );
6 node->setAttribute( "connect","true" );
7 node->setAttribute( "name","Herd Demo client" );
8 node->validateAttributes();
9 node->setPacketCallback(node, &wrapperHandlePackets);
```

Code 4.6: Creating a TCP node, set attributes and provide a callback function for receiving packets.

The creation of sessions is basic TCP handling, but in addition for higher level sessions we provide more information and have an additional *handshake*, providing the connecting session with an identifier number. At the same time on the receiver side that is creating the session, it generates a new *HerdDataPacket* that indicates that a new session has been created. The connection flow is shown in figure 4.13. The actor (higher level application) runs the node which will run internally a thread in which it will try to connect using the previously assigned attributes. Upon connecting to the server side, a new session will be created and a first data packet is send to the client containing the identifier for the session. On both sides the *HerdTcpSession* will create a packet and provide it to the *HerdTcpNode* which is then either being buffered or provided to the callback to be handled by the application.

The client now can start sending and receiving data, as shown in figure 4.13 as point 7. A user packet is provided to the *HerdTcpNode* and propagated to the receiver, the server, where the byte stream is read and put into a new *HerdDataPacket* object, which is then given to the HerdTcpNode for further processing. Aside from the low-level packet type protocol (in the five byte header) we use an additional protocol for specifying the purpose of the packet. As seen in the code 4.5 the first field in the data packet is a two byte unsigned short type (in the example UserProtocol::CUSTOM_PACKET_ID). We define a base protocol and on top of that a user/application protocol. The base protocol, code snipped 4.7, lists the most common used identifiers.

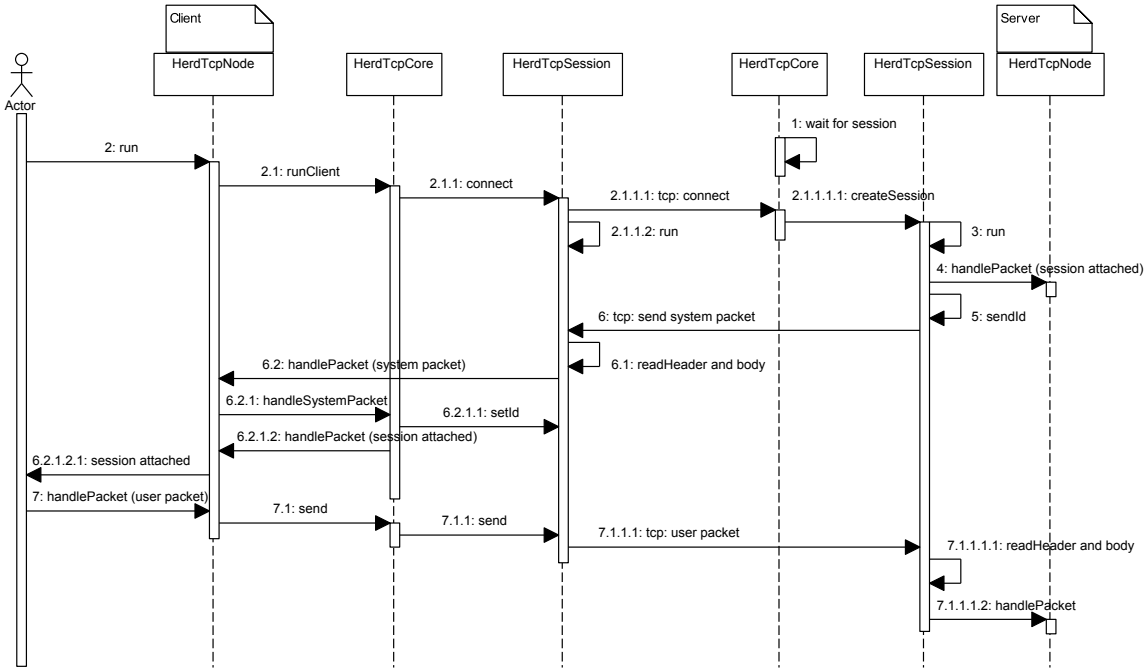


Figure 4.13: Establish connection using the HerdTcpNode.

```
SESSION, USER_EVENT,
SESSION_ATTACH, SESSION_DETACH, PROFILE,
FRAME,FRAME_RATE_UP, FRAME_RATE_DOWN, FRAME_RATE, RESET_FRAME,
MOUSE_DOWN, MOUSE_UP, MOUSE_MOVE,
KEY_DOWN, KEY_UP,
BASE_PROTOCOL_END
```

Code 4.7: Base protocol containing system and basic event identifiers.

The first couple of identifiers are used during the initialization of the session, where *SESSION_ATTACH/DETACH* are created internally by the *HerdTcpNode* as illustrated before. Thereafter are basic adaptation identifiers, e.g. for sending video frames., and last basic input events for mouse and keyboard. The protocol ends with an identifier *BASE_PROTOCOL_END* which is not used directly but is used by the application-level protocol, which may start from this identifier. Thus for example the identifier from code 4.5 *CUSTOM_PACKET_ID* may be defined as "$CUSTOM\_PACKET\_ID = BASE\_PROTOCOL\_END + 1$"

4.2.3.2 User Datagram Protocol node

The benefit of using UDP is speed, which is favourable for services where guaranteed transfer of data is of lower importance. Since UDPis connectionless it does not serve well for session based services, however with a small extension and without reimplementing the whole TCP stack of functionalities it is preferable using it for input events and video streaming. For video streaming then sender side compresses a frame and sends it to the other side, which decompresses the frame again, this however entails that the codec used should be able to handle *lost frames*. As for input events similar rules are applied, for example mouse movement is a stream of screen coordinates of the mouse pointer, losing a couple of these events is not critical. However when-

ever there is a *mouse button press* it is a more important event. We therefore introduce two types of UDP based streaming, command and stream, as can be seen in the class diagram figure 4.14. These implementation are solely used for the framework presented and alike to the *HerdTcpNode* serve its purpose for the presented use-cases. Other protocols based on UDP such as RUDP, DHCP, RTP etc. should be encapsulated into their own respective *HerdNode*.

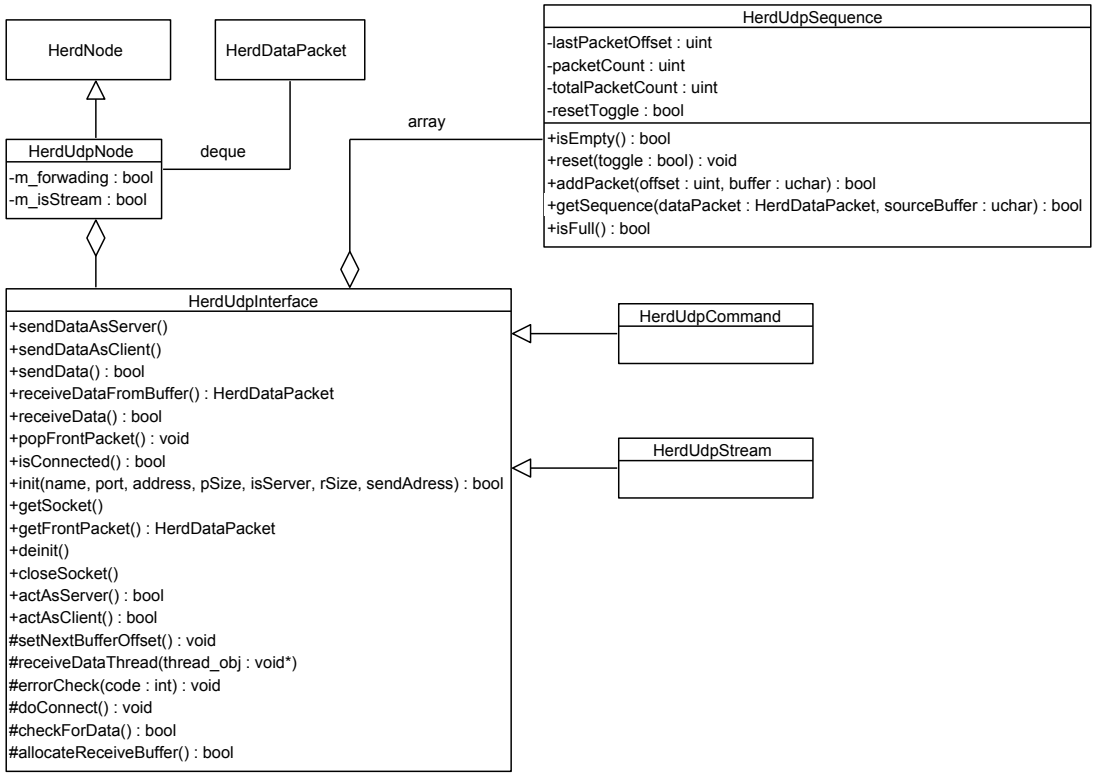


Figure 4.14: Udp functionality overview.

Both types extend on the *HerdUdpInterface* which is similar to the *HerdTcpCore*, handling connections, sending and receiving data. Since there are no sessions it is left to the implementation of the service/application on how to handle this if needed. This might be done by using per *application session* a different socket, or provide additional information in the data packet. The *actAsServer* starts listening to a given socket and handles any data that it receives on it, it does not know anything about sending data at this point. Where the *actAsClient* is given an address to where it has to send(connect), it does not check if it exists as it will *fire* packets into the net unknowingly if it ever arrives. Whenever the server receives datagram packets it will use the senders address as its new reply address. In practice this works fairly well, and in fact most networked games work use UDP for player movements. The *HerdUdpInterface* keeps a buffer for the received datagrams, using *HerdUdpSequence* objects. A datagram is limited in size (theoretical limit of 65507 [4], but commonly a size of around 1400 bytes or less is used) and therefore the transfer of data can be split into several datagrams. Furthermore the datagrams can arrive in an unordered fashion, meaning that for example sending two *HerdDataPacket* objects can be split

[4] 65535 minus the Internet protocol (IP) header and UDP header itself

Table 4.3: UDP buffer cell structure

Cell Header	Packet header			Body
Parent packet offset 4 bytes	type 1 byte	sequence 1 byte	info field 2 bytes	data max (udp packet size - header size)

into several datagrams and be received in a mixed order. Depending on the network hardware and lower level implementation, the buffer for receiving packets is limited, which can lead to flooding of the buffer when receiving too many packets. To avoid this the buffer must be read constantly and must not loose time on later processing of the data and from practice this turned out to be quite tricky. To handle UDP quickly and correctly (incorporating the possibility of fragmentation) we encapsulate the *HerdDataPacket* into a new packet and use a fixed buffer for placing the incoming data without any further processing. The method we came up with proofed to be quite fast and foremost easy to use. The fixed buffer is a one time memory pool allocation, which can contain an X amount of datagrams. Each datagram has an additional header of 4 bytes and the data within the datagram also contains a 4 byte header for the data, this is shown in table 4.3. The fields within a cell are:

- *Parent packet offset*, during writing a cell an offset to its parent cell with the same sequence is written here and the new latest offset is kept in the sequence object.
- *Type*, indicates the packet state. Possible values are
  - Value 1 Indicating the first packet of a sequence of packets.
  - Value 2 An in between follow-up packet or so called sequence packet.
  - Value 4 The final packet for a sequence of packets.
  - Value 8 A singular packet that is the first and final packet (simply a single packet).
- *Sequence*, UDP streaming can keep track of maximum 256 different packet sequences the sequence ID is the index number
- *Info*, The packet number (maximum packet numbers 65535)

Assume the maximum size for a datagrams is set to 1450 bytes and the buffer is set to 1 MiB, for each datagram stored 4 bytes of additional data is used, therefore 721 datagrams can be stored into it, which we call cells. Even if a datagram is not using its full capacity, it will be registered as a full cell in the buffer. Upon receiving a datagram, a single *memory copy* is used to place it into the buffer at the current buffer offset, which thereafter increments. The offset resets to the beginning of the buffer at the last cell, creating a ring buffer. A secondary buffer is used for managing and registering the incoming packets, which is an array of 256 *HerdUdpSequence* objects. For each sequence number (0-255) there is an *HerdUdpSequence* that keeps track of any incoming datagrams with the given identifier. This procedure is illustrated in figure 4.15.

Note that data received is directly written into the buffer and a pointer is used for indicating the start of the cell in the buffer. At the end there is a check if the active sequence is complete,

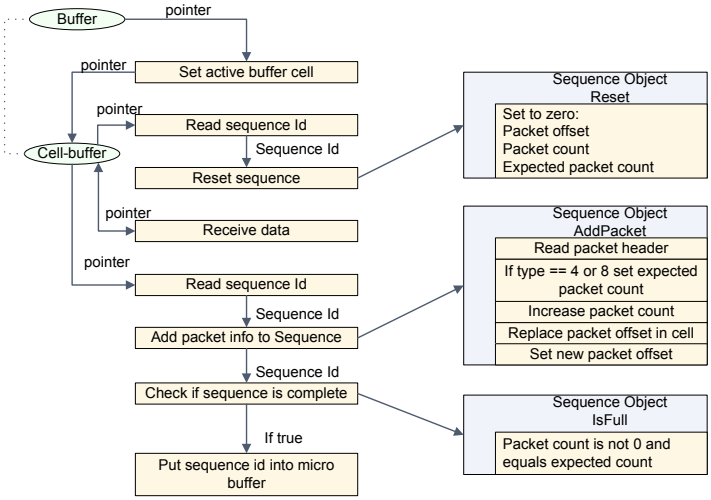


Figure 4.15: Reading UDP packets.

meaning it either has a single packet of type 8 or the internal count of packets is equal to the expected count of packet (which is the packet number from packet type 4). The identifier of the sequence is placed into a *micro buffer*, which in case of streaming has a limited amount (e.g. 4 identifiers). It also sorts the identifiers with a relaxing sort at the end of the buffer as we want sequences with a lower identifier handled first, but at the end it can occur that sequence 255 comes a bit late and sequence 0 has been placed already, then we still want 255 to be handled first, but still after 254. The placements of the identifiers into the micro buffer is thread-safe and from another thread the available identifiers are retrieved. This reduces the amount of operations to be done within the actual networking thread and minimized the chance of packet flooding. Packets are still being dropped if the micro-buffer is full, but it will push out the older ones first. Whenever an identifier is retrieved in the other thread, the sequence object is accessed and a the function *getSequence* is called. A new *HerdDataPacket* object is created and the needed amount of memory is allocated.

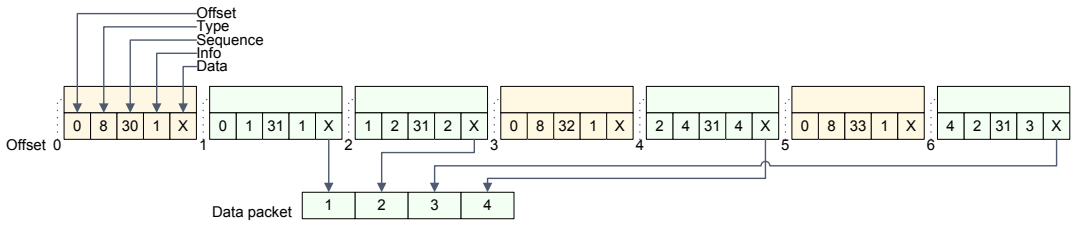


Figure 4.16: Retrieving a data packet from the buffer.

Shown in figure 4.16 is the buffer with several cells illustrated. The first cell has 0 for offset, 8 for type meaning it is a single packet in this cell, sequence id 30, info set to 1 as it is packet number 1 and the data is marked *X* as this can be the maximum size of the remaining cell space. In this case we want to retrieve sequence 31 which is composed out of several cells and spread out with higher sequence numbers in between and incorrect order. The latest received data contained

packet number 3, therefore the *HerdUdpSequence* object is pointing to offset 6 internally, it will take the data and place it directly into the *HerdDataPacket* at position 3, then it will read the offset from the cell header, which is pointing to 4, thus the next cell at offset 4 is read and placed. This procedure is continued until the *HerdDataPacket* is filled. Afterwards the *HerdDataPacket* object is forwarded to the *handlePacket* function where it is either buffered or redirected to the callback function.

4.2.3.3 Publish-subscribe data sharing node

The *published-subscribe* data sharing allows for easy synchronization between data models. Although the focus is on remote rendering and adapting towards the client, it became clear that there is more data other than the rendering that can benefit the adaptation. For example for localized rendering mixed together with remote rendering providing a hybrid solution. In order to make it generic, we simply looked at what other data we have and how this is useful for sharing. For collaboration it becomes very important where the users are working together virtually and how where physically this virtual workspace is hosted, either at a central system, and every view is streamed to each end-user or locally and modifications are propagated to each other. This type of synchronization however is dependent on the type of connection either one user since his modification to all other users (peer-to-peer) or to a central server that then takes care of updating the other users (client-server). Our approach is capable of handling both situations by breaking down the relation in generic terms of subscriber and publisher. A client for example can be a subscriber and publisher. This however involves some management of data-transfer between the subscriber and publisher within the client application. We can define three spaces where data is residing, outside somewhere on the network, internally as a generic data model and specialised data models optimized for specific routines that use the data. This is illustrated in figure 4.17. Where the publisher is on the outside and communicates directly with the subscriber which is internal and upon the subscriber we have listeners that translate the data into the specialized data models.

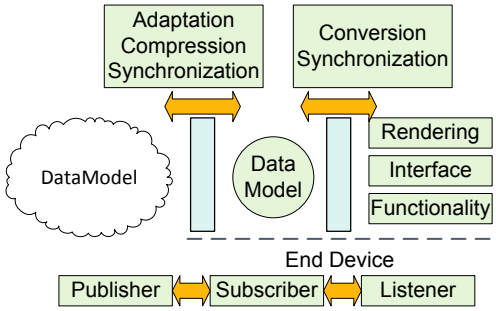


Figure 4.17: Flow of data from the cloud to the local data model and to application data. The cloud data model is the Publisher, the local data model the Subscriber and the application data the listener.

The *publisher* and *subscriber* are similar in functionality, as both maintain a tree-based data structure of caches and objects, which is being kept synchronised. The data stored on both sides may not be identical due to the adaptation rules being set for the given session. From figure 4.11 we

can extract three important components, the *MemManager* from which both the publisher and subscriber inherit, the *MemObject* which will contain the data and *MemCache* will inherit and third the *MemListener*. In addition there is a protocol extending the base protocol (code: 4.7) that provides the necessary identifiers for handling the communication between the internal *MemManager* and external *MemManager*, but also the internal communication from *MemManager* to the listener.

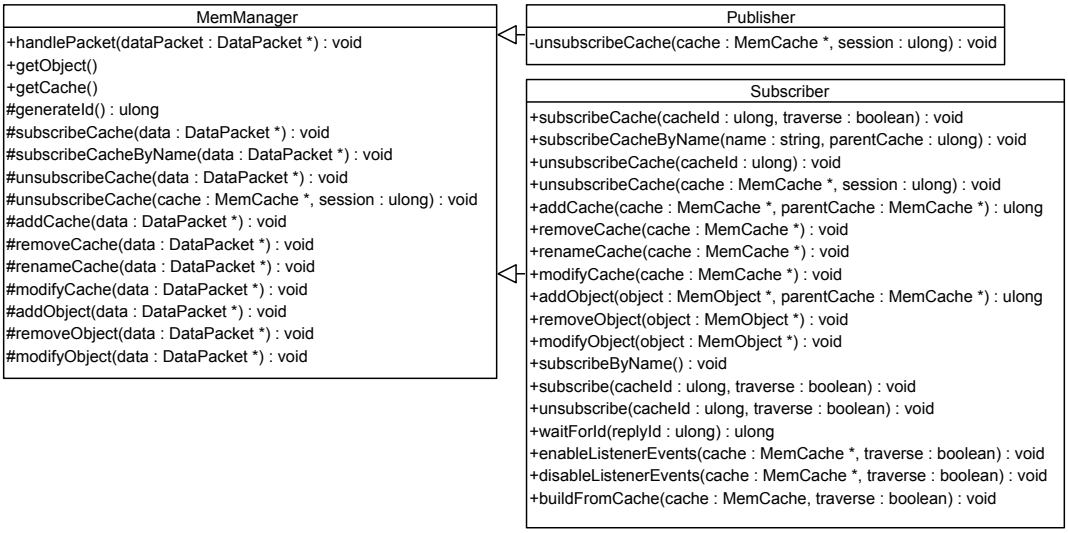


Figure 4.18: Abstract class diagram of the MemManager, Publisher and Subscriber.

The *publisher* only handles the *HerdDataPacket* objects, where the *subscriber* extends on this by also providing similar function for direct access as shown in figure 4.18. Which is also reflected in the encapsulating node classes. An example of creating a subscriber and adding data is given in code 4.8.

```
1 auto node = kernel->createNode("HerdNodeLibCore", "HerdSubscriberNode");
2 subscriber = static_cast<HerdSubscriberNode *>(node);
3 subscriber->run();
4 subscriber->subscribe(0);
5 auto parentCache = m_subscriber->getCache(0);
6
7 MemCachePtr cache(new MemCache("Test cache"));
8 subscriber->addCache(cache, parentCache);
9
10 MemObjectPtr memObject( new MemObject());
11 memObject->data()->addField("This is some test data.");
12 memObject->data()->addField("Another string.");
13 memObject->data()->addField(42);
14 memObject->data()->addField(1024.0f);
15 subscriber->addObject(memObject, cache);
```

Code 4.8: Creating the subscriber and add data.

First through the *kernel* the required node is created. In this example we assume that there is already a publisher node active on the system and the subscriber is able to connect directly to it upon calling the *run* function. The subscriber *subscribes* to the *root*, which always has identifier 0.

We then can get the *root cache* which is used as the parent cache for a new cache to be created, at line 7. Using the *addCache* function we tell the subscriber to add the cache as a child to the root cache. After that a new object is created, at line 10, which is child of the previously created cache. By calling the *data* function we get access to the underlying *HerdDataPacket* and can add various data. Using the *addObject* function the publisher is informed about the new object and is added to the data tree. Any other subsciber is directly notified of these changes, but only those that are subscribed to the *cache*. Figure 4.19 shows that a *MemCache* has a set of subscribers (which is a list of TCP session identifiers and not the instance or pointer of *MemSubscriber*) and a set of *MemObject* instances. The *MemCache* itself is also of base type *MemObject* and can therefore, aside from a string typed name also contain various data.

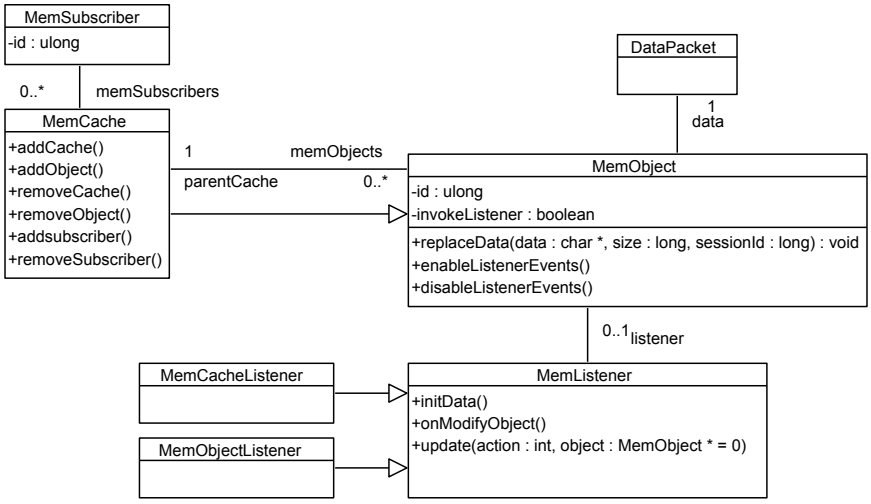


Figure 4.19: Abstract class diagram of the MemObject, MemCache and MemListener.

The whole concept of this approach is to avoid complexity and keeping it minimal, having a singular connection publish/subscribe, a data tree of containers called *MemCache* and leaves called *MemObject*. Without diving into constraints and type limitations which is often seen in other data formats. This however does bring limitations in accessing the data. Each object does have an identifier, and subscribing to a cache is best done by its identifier. But a subscriber that just connects is not aware of what is already existing. It can therefore subscribe to a cache without traversing the whole tree and the same applies for unsubscribe. Also the subscriber provides a function *subscribeCacheByName*, however this will make a subscription to the first cache with the given name within the list of children of the given parent cache, since it is possible to simply add caches with the same name. This is currently left for the developer to take care of. A default approach is to first always subscribe to the root, as this is a mandatory cache, without traversing it. From there look at its children cache names and then decide whether to create a new cache or subscribe to an existing one.

Until now we only discussed the direct connection between subscriber and publisher, but not the relation to its listeners and what these listeners exactly are. Whenever the publisher is pushing an update to its subscribers, the subscribers will update their data model accordingly, however the higher level application has no direct notion of this happening, since it happens in its own thread

and is therefore separated. Although it is possible to directly link the data from the *MemObject* by pointer and therefore any update is given, albeit not really thread safe, but in some cases it can be used. For example we can store the mouse cursor position of each user in a single cache and the client application is directly linked to the data object and uses the data as coordinates for drawing cursors for each user. Whenever the data is more sensitive and does not allow for errors to occur or when there is a conversion needed a more sophisticated method is needed, which is provided by the listeners. For example a listener is needed when we want to share a 3D model's vertex data, to render this data we can use vertex buffer objects (VBOs). Since we only need to update the VBO whenever there is an actual update from the publisher, we need to be informed about this, upon which we can copy the data from main memory into the graphic cards memory using OpenGL commands [5] and possibly some pre-processing might be needed (e.g. calculating new normal and tangent vectors). The listeners are providing these functions. Depending on the listener it either contains a *MemCache* or *MemObject* and is used at a higher level for inheritance. A class inherits from a listener type (cache or object) and can overwrite certain event functions which are shown in figure 4.20

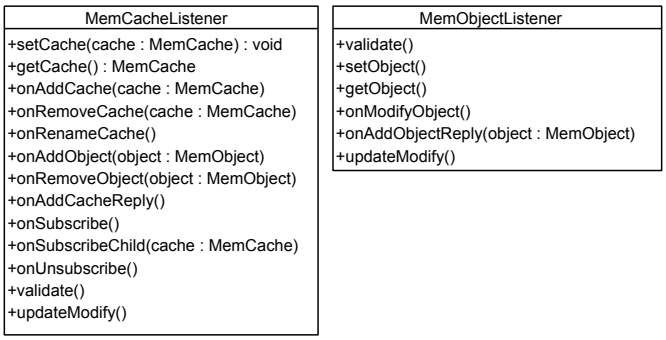


Figure 4.20: Abstract class diagram of MemObjectListener and MemCacheListener.

This provides the communication from the subscriber to the listener, but not from listener to subscriber. Modifying any of the data objects doesn't invoke any updates towards the publisher. For this to happen the *updateModify* function is provided. This function can also be overwritten since conversion of data might be needed again (just in reversed direction). Ultimately the *updateModify* function in the *MemSubscriber* class is called which takes the full data object (thus the *HerdDataPacket*) and places this into a new *HerdDataPacket*, this is shown in code 4.9

[5] VBO information `http://www.opengl.org/wiki/Vertex_Specification#Vertex_Buffer_Object`

```
if (!_object->id()) {
  std::cerr << "MemSubscriber::modifyObject - Unable to modify publisher object, local
   object has no valid id" << std::endl;
  return;
}
auto packet = HerdDataPacket::create();
packet->addField(static_cast<unsigned short>(MemProtocol::MODIFY_OBJECT));
packet->addField(_object->id());
packet->addField(_object->getData(), _object->getSize()); //the data
packet->encodeHeader();
m_node->handlePacket(packet);
}
```

Code 4.9: inform the publisher to update an object.

So far we can build a tree of data, subscribe to specific parts of the tree, get informed about any actions performed on what we have subscribed to. For each action we can implement a custom routine for handling and invoking new operations. It is possible to modify the data and then have it updated to the publisher. There is however one obstacle left to solve, which is the initial synchronizing of the data model with dependencies. In the tree of data each object is its own with the only relation of having a parent cache. However on the application level the data model may have complex dependencies on the data elements, where for example data element *A* can only exist if data element *B* exists or has a certain value. This is no problem if the application already has a valid data model that has to be published, since it will just be a conversion of each data element to a data object and the relation can be stored in the data by using the identifier of the object. However if the application is linking to a publisher which only pushes a set of data elements, then the application must first wait until the subscription is finished synchronizing. After this it can invoke the *buildFromCache* function which will traverse from a given cache through all child caches and objects, for each of these a *invokeUpdate* update is called with *add_cache* or *add_object* as action. This in turn will call the listeners *onAddCache/Object* functions, which are then used for building the appropriate data model.

4.2.4 Media Plugin Library

We can define two types of data, generic data and image data. Generic data is basically any kind of data that we want to transmit, this data should arrive exactly the same as it was send. The image data on the other hand can be compressed with a degree of loss in image quality and furthermore can be treated as a single image or a stream of images (video). Therefore three different types of compression are provided, lossless and data type independent compression is done by zip compression (for development zlib was used [6]), a single image lossy compression using JPEG (libjpeg [7]) and the third compression codec used for continues streaming rendered images the video codec VP8 (WebM[8]).

[6] "zlib was written by Jean-loup Gailly (compression) and Mark Adler (decompression)" `http://www.zlib.net/`

[7] "Developed by Tom Lane and the Independent JPEG Group (IJG) during the 1990's and it is now maintained by several developers" `http://libjpeg.sourceforge.net/`

[8] Developed by On2 Technologies `http://www.webmproject.org/code/`

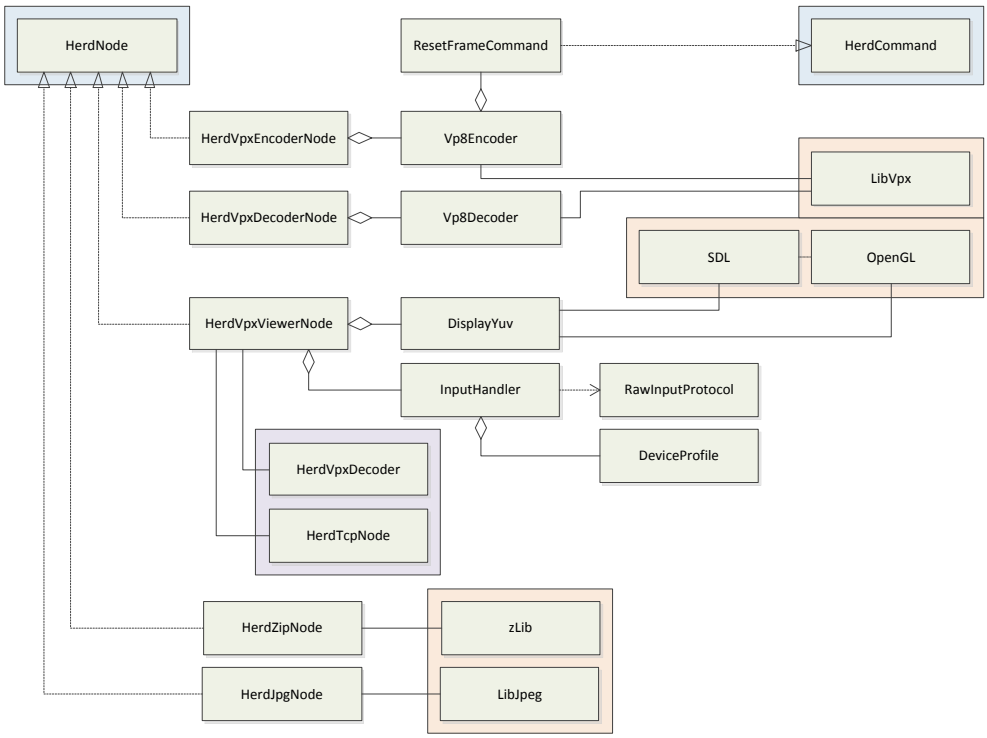


Figure 4.21: Vpx (encoding, decoding), Jpeg and Zip media nodes diagram.

In figure 4.21 the encapsulation of these compression libraries are given with additionally a viewer for the VP8 codec. The *Vp8Encoder* and *Vp8Decoder* encapsulate the external library *LibVpx* and provide a specific interface for using it (encoding or decoding).

4.2.4.1 Encoding and decoding

For encoding the nodes *HerdVpxEncoderNode*, *HerdZipNode* and *HerdJpgNode* work in a similar manner. A *HerdDataPacket* is provided to the *handlePacket* function in the encapsulating node where it is being buffered into a thread safe array. In a separate thread the array is checked and for any new data it is being compressed. After compression the data is placed into a new *HerdDataPacket* object and is forwarded to its child nodes (e.g. a *HerdTcpNode*). The *HerdZipNode* and *HerdJpgNode* are set to encoding through the use of node attributes, where the VP8 is provided in separate encoding and decoding nodes. Both the Jpg and Vp8 encoders assume that the data provided is in RGB format. In general the procedure is as follows: The *Vp8Encoder* initializes itself for encoding, providing pre-set variables such as the target bit-rate, the key frame interval, width and height. Then within running thread upon receiving data the encode function is called and the dimensions are checked between given frame and the encoder's initialized dimensions. The frame data is then extracted from the *HerdDataPacket*, converted into YUV format[9] and copied into the buffer for encoding. The output of the given frame is provided into two parts, first the header, which is 12 bytes and then the body. The data is copied into a *HerdDataPacket*,

[9]YUV is a colour space with reduced bandwidth for chrominance components `http://en.wikipedia.org/wiki/YUV`

additionally with the base protocol identifier *frame* (code: 4.7).

The decoding works similar to the encoder, with the difference that the decoded data is handled again by the node, and then either being placed into a buffer or send to a given callback function. The decoding output of VP8 frames however is handled in a more special way. The data from the *HerdDataPacket* object is decoded into a YUV frame but is not directly converted into RGB, instead separate buffers for each channel (Y,U,V) are directly accessible from the outside. The *HerdVpxViewer* takes advantage of this by converting the data itself according to the capabilities of the hardware and platform it is running on. Offering three different conversions, the straightforward single threaded CPU conversion which is the most intensive but can run on any system, a more optimized threaded version which can give more benefit with larger image dimensions and a GPU version which uses an OpenGL fragment shader for the conversion.

4.2.4.2 Remote rendering client viewer

The *HerdVpxViewer* is used as a general client to any VP8 video streaming service and supports the streaming over TCP and UDP. Upon executing the service it creates a single window in which the incoming frames are displayed. The rendering is using the Simple DirectMedia Library [10] and can use either software renderer or accelerated renderer using OpenGL (SDL also supports Direct3D however no rendering path for this has been implemented). Several checks are used to determine the appropriate renderer to use. If accelerated is supported (in this case we only use OpenGL) a second test is used to determine if shaders are supported and the non-square texture support [11]. If the test fails, the CPU method for YUV to RGB conversion is used and the result is rendered using OpenGL's command *glDrawPixels*. Otherwise the shader is used and the process of converting YUV to RGB is transferred to the GPU. The shader for this is given in code 4.10.

[10] Simple DirectMedia Library (SDL) `www.libsdl.org`

[11] GL_ARB_texture_rectangle enable non-power of two dimensions texture targets `http://www.opengl.org/registry/specs/ARB/texture_rectangle.txt`

```
#extension GL_ARB_texture_rectangle : enable
uniform sampler2DRect Ytex, Utex, Vtex;

void main(void) {
  float r,g,b, y,u,v;
  y = texture2DRect(Ytex,vec2(gl_TexCoord[0].x, gl_TexCoord[0].y)).r;
  float nx = gl_TexCoord[0].x * 0.5;
  float ny = gl_TexCoord[0].y * 0.5;
  u = texture2DRect(Utex, vec2(nx, ny)).r;
  v = texture2DRect(Vtex, vec2(nx, ny)).r;
  y = 1.1643 * (y - 0.0625);
  u = u - 0.5;
  v = v - 0.5;
  r = y + 1.5958  * v;
  g = y - 0.39173 * u - 0.81290 * v;
  b = y + 2.017   * u;
  gl_FragColor = vec4(r,s g, b, 1.0);
};
```

Code 4.10: the GLSL fragment program for converting an YUV image to RGB.

From the code it is shown that for each channel a separate texture is used (*uniform sampler2DRect*) and corresponds to the channels provided by the decoder. In code 4.11 the rendering and update of the textures procedure is shown. For each channel the corresponding texture unit is activated and updated using the *glTexSubImage2D* command, which streams the buffer from the main memory to the GPU memory. After updating each channel a full-screen quad polygon is rendered which in combination with the active fragment shader visualizes the final image.

```
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_RECTANGLE_NV,1);
glTexSubImage2D(GL_TEXTURE_RECTANGLE_NV, 0, 0, 0, m_width>>1, m_height>>1, GL_LUMINANCE,
   GL_UNSIGNED_BYTE, m_uBuffer);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_RECTANGLE_NV,2);
glTexSubImage2D(GL_TEXTURE_RECTANGLE_NV, 0, 0, 0, m_width>>1, m_height>>1, GL_LUMINANCE,
   GL_UNSIGNED_BYTE, m_vBuffer);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_RECTANGLE_NV,3);
glTexSubImage2D(GL_TEXTURE_RECTANGLE_NV, 0, 0, 0, m_width, m_height,GL_LUMINANCE,
   GL_UNSIGNED_BYTE, m_yBuffer);
drawQuad();
SDL_GL_SwapBuffers();
```

Code 4.11: Updating the Y, U and V textures.

4.2.5 From Desing to Implementation and back

In this section we reiterate on the presented design in the previous chapter and look at how the implemented architecture relates to it. We look at the adaptive rendering overall architecture as presented in figure 3.2 and elaborate on each of the several layers in a bottom-up fashion. At the lowest level of the architecture we find the *Network layer*, the physical layer. On this part

we have not much influence other than creating supportive software on top of it. Albeit one might argue that changing structures in the physical network in order to adapt to the need of the framework would greatly improve its usefulness (e.g. packet prioritizing, better multicast support etc.). This is however out of the scope of the current research, as we mainly look at the structures on top of this layer and how these can be more optimally structured and deployed. The *Communication abstraction layer* containing the *event manager* and the *communication manager* can be found back in the form of the basic *HerdNode* and the implementations of the protocols inheriting it. We find these in the *Core libraries* (section 4.2.3), supporting TCP and UDP. The data packets (*HerdDataPacket*) are basically the events containing an identifier and other data. As was illustrated in table 4.1, a data packet contains a minimal header and does not force the use of an identifier. The header, containing *packet size* and *system protocol identifier*, has the only five bytes that are mandatory. The use of an *extra* identifier is presented here as a good practice and is used in all application scenarios as user protocols. Where the first two bytes *after* the *system protocol identifier* reflect the type of packet, represented in an *unsigned short* (hence two bytes and thus a theoretical limit of *65536* identifiers are possible). The communication node processes incoming and outgoing packets and performs minimal event management. Albeit the presented TCP does register sessions and creates internally packets to inform the higher level event management. This is given through either a callback function, pushing upwards to a parent node or by buffering and externally fetching packets from the buffer. The usual method is using the callback functionality, it prevents unwanted handling of packets in the parent node (it also gives a more clear direction of the packet flow, thus parent to child). Fetching from the buffer is useful if the packet has to be handled within a certain thread (e.g. rendering thread or specific GUI architectures in which multi-threading makes it overly complicated). In general *Event Manager* is the function that reads just the first two bytes and then determines where the packet has to go for further processing. This function is simply a switch-case structure, an example is shown in code 4.12.

```
1 void ClassName::handlePacket(DataPacketPtr packet)
2 {
3   unsigned short eventId = packet->getUShort();
4   switch(eventId)
5   {
6     case CommunicationProtocol::MOUSE_DOWN:
7       handleMouseDown(packet);
8       break;
9     case CommunicationProtocol::FRAME_RATE:
10      handleFrameTime(packet);
11      break;
12    default:
13      std::cout<<"ClassName::handlePacket Unknown event ID: " << eventId << std::endl;
14  }
15 }
```

Code 4.12: Example of handling an incoming a data packet.

The example has been shortened for visibility, showing two cases from the user protocol, where

the first case *MOUSE_DOWN* is further processed in function *handleMouseDown* and is shown in code snippet 4.13. The function *handleMouseDown* implements the actual unmarshalling of the packet. It shows that in sequence the data is taken from the packet and fed to the function that acts on the given data. In this case it is a viewer instance from OSG where we force a mouse button press event. Which is then handled by the OSG instance and further application logic that responds to it.

```
1 void ClassName::handleMouseDown(Herd::DataPacketPtr _dataPacket)
2 {
3   float x = _dataPacket->getFloat();
4   float y = _dataPacket->getFloat();
5   unsigned int b =_dataPacket->getUChar();
6   m_guiInstance->getEventQueue()->mouseButtonPress(x,y,b);
7 }
```

Code 4.13: Process a mouse button down (event)data packet.

If we look back at the overall architecture then the single box for *Event Manager* is represented in the implementation a dispersed set of functions, that either run in sequence or are operating independently in a paralleled fashion with the only connection a shared buffer. This makes it hard to concretely lay down the handling of event as we can only describe it at a very low level. Using any 3rd party library often already introduces their own event management system and therefore changes the implementation. The benefit of the presented framework is that it is able to adapt to these circumstances in that it offers a uniform low level connection, easy enough for using custom packet handling or 3rd party conversion/adaptations.

At the next layer the focus lies on the adaptation, containing the modules *Context Manager, Adaptation Manager* and *Data Controller*. Most of the functionality needed for these modules are harboured in the *Media* library, containing several nodes for compress, decompress and visualization. The *Context Manager* is best represented by the viewer, *HerdVpxViewerNode*[12], as it monitors the rate at which it can process the data and sends back information on the optimal data rate. This is continuously updated in order to adapt to changes in the either the network or on the device itself. However the viewer, which uses SDL (OpenGL), is not utilized in all scenario applications. Mainly due to different requirements on the actual streaming. In most cases where the 3D content is streamed the viewer can be directly used as a node attached within the client program logic, but in some cases different means are used. Best example is scenario *The Collaborative MRI Segmentation*, which requests new frames, instead of a steady stream of frames. The viewer also has been partially reimplemented in Java and C# and in combination with other visualization libraries such as Qt and OSG. Therefore the *Context Manager* cannot be pinpointed to a single part in the framework, it does however provide a reference implementation given by the viewer. The same applies to the subcomponent *Resource Manager*, as different 3rd party dependencies change the way on how to handle data, out framework does provide uniform methods on how to generalize the procedure. The *HerdNode* provides a single entry for data packets, these can be processed directly or buffered. In case of buffering the node is usually running its

[12]The name HerdVpxViewerNode is a bit misleading as it also can visualize JPG and direct raw images.

own thread and checks the buffer for new packets. Another method is the *publish-subscribe data sharing* (section 4.2.3.3), which offers a full sharing and data coupling solution between server, client and the client internal data structures (figure 4.17). This is linked with the *Data Controller* as it offers a structured way on how to control the data. It does not enforce what specifically should be done with the data, as this is part of the overall program logic (thus the application functionality). It is also linked with the layer on top of it, the *Presentation layer*, which focuses on the actual rendering, compression and state of the program. The procedure of the 3D rendering and presentation information exchange to the 2D rendering is best given in the first application scenario *Remote Rendering for Low-end Devices* in section 4.3. Where the rendering is performed by some graphics library (e.g. Used OpenGL based libraries such as SDL, Qt and OSG), frames being grabbed and forwarded to a compression node and after send to connected sessions. The *Dynamic state management* refers more to the publish-subscribe internal data structures which are event driven by any changes that are made to the data, either coming from the *Data Controller* (e.g. from the network) or from the top layer, the *Application Layer*. The *Device Input Management* is translated into client side functions that translate incoming application events into event data packets with a identifier from the *user protocol*. During the implementation of the scenarios (described in the sections hereafter) several adaptations for handling the input management have been invented. Mostly combined with the interface adaptation and representation methods. The last layer on top of it all is the *Application Layer* which contains most of the application functionality itself. Meaning that all the layers underneath it are there to enable and extended user experience by facilitating collaboration, unburden the client and/or enabling clients to present visualizations of which they are inherently not capable. One major aspect, which is not truly visible in the presented *context-aware adaptation rendering system* architecture, is the scalability itself. The framework does not decide what has to be a server or a client (as the way it is presented in figure 3.2), it simply offers the flexibility with the presented node structure, and full-fills the requirement.

4.3 Remote Rendering for Low-end Devices

The implementation of the first scenario as described in section 3.5 is setup as a typical server/-client. Where the server is a service on the network and the client can be utilized on a variety of hardware. Following the layers presented in figure 3.9 we can already see that many of the functionality is provided by the previously described core- and media-library plugin. The server is utilizing the OSG rendering engine and can load 3D scenes from file. In the test setup (see section 5.2) a static 3d model is used and interaction is limited to rotation of the model.

In order to capture a frame within OSG, a post camera callback has to be added to the active camera within a *viewport* (figure 4.22). The callback contains a pixel buffer, which is activated and a fast copy of the front buffer (since we don't need the back buffer, double buffering is disabled by default) into the pixel buffer is performed. Thereafter the mapping is set to the pixel buffer, through which we can access directly the data and copy the pixel information into a local buffer. This provides a great performance in contrast to directly reading from the *front buffer*

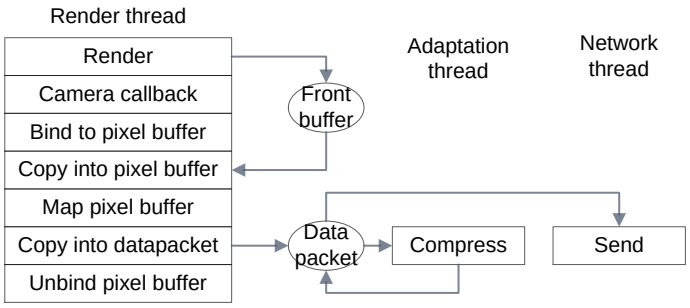


Figure 4.22: Rendering and reading the buffer from the GPU to local memory, in order to be processed by the adaptation and finally send over the network.

using for example *glReadPixels*. The local buffer, *Data packet*, is pushed into the adaptation thread where the data is being compressed. For compression the previously presented media library is used and can switch depending on the device between the *JPEG* and *VP8* codec. The output packet is given to the network thread to be send to the end-device for display. For grabbing the frame in OSG the code presented in snippet 4.14 and 4.15. The code has been condensed to the specific frame grab functionality only, constructors and helper functions are removed as well as additional functionalities for adaptation, such as per user session frame per second limit.

```
1 struct RenderThread : public osg::Camera::DrawCallback {
2 public:
3  virtual void operator () (osg::RenderInfo& renderInfo) const {
4   glReadBuffer(_readBuffer);
5   auto extension = osg::GLBufferObject::getExtensions(_gc->getState()->getContextID(),true);
6   if (ext->isPBOSupported()) {
7    grabFrame(extension);
8   }
9
10  void grabFrame(osg::GLBufferObject::Extensions* ext);
11 }};
```

Code 4.14: Example code for grabbing a frame in OSG, internal struct ContextData

The snippets show the main function for grabbing a frame, since this is specific code using OSG some limitations were applied such as the use of a *osg::Camera::DrawCallback* struct. However the callback can be easily attached to any camera given in OSG and the given view of the virtual camera is streamed. The *struct* is called *RenderThread* which implies that it is a thread, this naming is related to the threaded nature of OSG which heavily relies on threading. But mostly this name was chosen as to distinguish between render thread, compression thread and networking thread which it originally started with.

```
1 void RenderThread::grabFrame(osg::GLBufferObject::Extensions* ext) {
2  if (_pboBuffer==0) {
3   ext->glGenBuffers(1, &_pboBuffer);
4   ext->glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, _pboBuffer);
5   ext->glBufferData(GL_PIXEL_PACK_BUFFER_ARB, m_totalSizeInBytes, 0, GL_STREAM_READ);
6  } else {
7   ext->glBindBuffer(GL_PIXEL_PACK_BUFFER_ARB, _pboBuffer);
8  }
9  glReadPixels(m_user->m_offsetX, 0, m_user->m_width, m_user->m_height, _pixelFormat, _type, 0);
10    GLubyte* src = (GLubyte*)ext->glMapBuffer(GL_PIXEL_PACK_BUFFER_ARB, GL_READ_ONLY_ARB);
11    if( src ) {
12      m_dataPacket->setId(m_user->_user_index);
13      memcpy((void*)m_dataPacket->getReadPosition(), src, m_totalSizeInBytes);
14      m_user->m_encoder->handlePacket(m_dataPacket);
15      ext->glUnmapBuffer(GL_PIXEL_PACK_BUFFER_ARB);
16 }}
```

Code 4.15: Example code for grabbing a frame in OSG

What happens is as stated before we first set the buffer from which we want to get the pixel data, line 4 in snippet 4.14. After this since we utilize an intermediate copy of the buffer to a off-screen buffer the specific extension for this is being accessed and checked if supported. If the extension is supported the grabFrame function is being called. Through the use of the extension we set the appropriate settings and bindings. Snippet 4.15 can be read into two parts, first the check if the off-screen buffer was created (line 4, *_pboBuffer*. If not, then the buffer is allocated with the appropriate memory size. And otherwise the existing buffer is simple activated, called binding (line 7). The second part is the actual reading of the buffer. The command *glReadPizels* is used, but its usage now has changed slightly. Because of the binding of an *GL_PIXEL_PACK_BUFFER* the last parameter is no longer used as data output but as a byte offset within the bound buffer. Therefore the pixel data is directly copied into the buffer which then can be mapped and a local pointer access is given. Using a *memcpy* command the whole buffer is copied into a data packet which is then further handled by the encoder. Figure 4.23 shows a sample application using our proposed system.

Figure 4.23: Remote rendering for Laptop and PDA.

4.4 Remote Rendering with Augmented Reality

As described in the section 3.6 two different approaches to the given scenario are given. There is no direct interaction with the virtual camera other than physically moving the web-cam and/or the fiducial markers.

The first design, figure 3.10, directly sends the web-cam image to the server and performs no others task aside from rendering the incoming frames from the server. We utilized on the UMPC an attached web-cam and a small program called WebcamXP[13] which streams images from the client to the AR server. Once the imaged has arrived at the server, we use the ARToolKit library to track our camera according to the marker. The resulting transform then controls our virtual camera. We then augment our VTO frame with the web-cam image.

JPEG compression is performed on the image before sending it back to the client. A raw $320x240$ RGB8 image is 225KiB large. At 15 frames per second (fps) this would account for 3.3MiB, which approaches the limits or our UMPC's Wireless Fidelity (Wi-Fi) capabilities. The JPEG compression brings it down to about 12 to 14 KiB. We added compression by using Zlib[113], which reduced the size with an additional 2 to 4KiB, which does gain some additional performance even with the increased compression and decompression times. On the client side the images are decompressed and drawn into the frame buffer using a glDrawPixels call. The major advantage of this setup is that, because the server has the final augmented result it can be distributed using the networking middleware to any number of clients, thereby allowing the user to share their results in real-time. The problem with this setup lies in the used video streaming. AR-

[13]WebcamXP web-link: `www.webcamxp.com`

Toolkit expects to connect to a web-cam, which necessitated the use of WebcamXP to create an image stream and a virtual web-cam on the server side to connect to this stream. This drives up CPU utilization on the server side, and combined with the buffering that happens in the stream we get an additional lag of the final result. To eliminate the use of camera stream from client to server a second setup was established (figure 3.11) in which the camera tracking is performed on the client side. In stead of ARToolKit we used ARToolkitPlus [14]. This is an edition that is less heavy on the CPU and made with mobile devices in mind. All we transfer to the server is the transformation matrix that is the result from our tracking. This leads to another necessary change. Now the final augmentation is no longer performed on the server side, but by the client itself. The server only renders an image using the transformation matrix in the VTO. Augmenting this result onto the camera frame can be done in either of two ways. By employing a simple colour key blending scheme we can compress the image in a similar way as in the first setup. We found however that the JPEG compression would result in some unwanted artefacts around the border of our virtual fashion model. An alpha-blending approach provides results of a higher quality, although this comes at the cost of an extra alpha layer in addition to the RGB data. In the case of alpha blending we used *Zlib* compression only, although we could have combined it with a PNG format. Overall performance did not significantly differ between these two approaches. The added benefit of having an alpha-layer is the option to include shadowing information or even a level of transparency in garments made of certain translucent fabrics.

The advantage of this setup over the first one is the increase in performance. By sending only the transformation matrix the amount of data sent to the server is minimal. The limiting factor in this scenario turned out to be the capture speed of the used webcam, which was 15 fps for a decent quality. There are some disadvantages as well. We do currently not store the image used for tracking on the client. So when the resulting VTO frame comes back from the server it gets blended with a new frame from the camera. This causes some visible mismatch, although not nearly as significant or disturbing as the lag introduced by the first scenario. Furthermore the blending operations, although simple, do require some graphical power on the UMPC. The combination of glBlend and glDrawPixels calls currently do keep up with the web-cam speed, but at higher resolutions or frame rates this might turn out to be problematic. One of the advantages of the first scenario disappears as well. The server no longer has the final augmented frame, so distribution to other viewer clients is not as easy and natural as in the first scenario.

4.5 Adaptive Rendering with Dynamic Device Switching

The implementation of this application scenarios was done in two application setups. The first with the focus on switching between devices without losing the user session and the second on the sharing of data between devices. Both setups are depended on 3rd party provided technologies and will be more elaborated on in the respective sections.

[14] ARToolkitPlus web-link: `http://studierstube.icg.tugraz.at/handheld_ar/artoolkitplus.php`

4.5.1 Real-time switching device

In figure 4.24 the components are given for the two instances, the server side and client side. The server or *Content Server* contains the 3D content, rendering and adaptation, for security the *License Manager*, and the *Home Agent* for the Mobile IP (MIP). On the user side we see several counter parts, such as the *Personal Agent* and *PA Manager* which are part of the MIP, for security the *DRM client* module and last for the adaptive rendering the *Presentation Manager* and *Interface Adaptation Manager*. On the client side we have furthermore the localization module *Localization Management* which is connected to its external system the *Context Manager*. For specific implementations on the 3rd party modules see the Intermedia project for references as we can only go into detail on our framework and the integration (see section D.1).

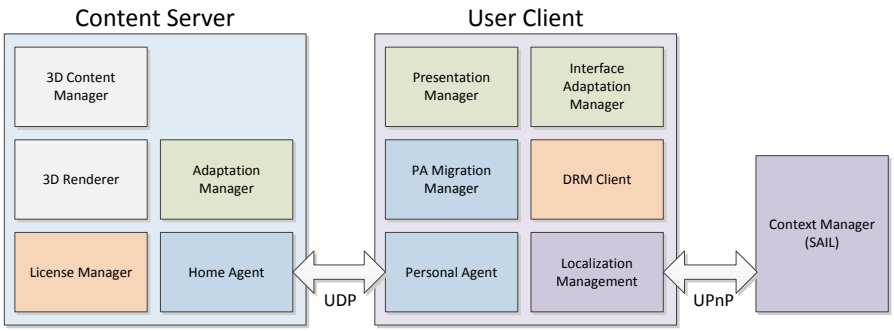


Figure 4.24: System setup for Adaptive Rendering with Dynamic Device Switchingscenario

The *Content Server* exists out of several servers (figure 4.25), in order to support Mobile IP two servers are employed, the security module is run separately as well as the *Context Manager(SAIL)*. The Mobile IP is ubiquitous towards the server and partially to the client application. Only need minimal implementation on the client side. Specific drivers need to be installed which did however limit deployment of the client implementation, as the Windows operating system was required and only UDP was supported for network communication.

For integrating the MIP the client has to use the provided library and include a single header, as shown in code 4.16. The header provides three functions that have to be used, *start*, *stop* and *check_registration*.

```
1 #pragma comment(lib, "MobileSession.lib")
2 #include "mobile_session.hpp"
```

Code 4.16: Adding the Mobile IP framework.

After the inclusion the MIP has to be started, meaning that we have to tell the system that we want to use the IP provided by the MIP framework. This will add the session address to the device and starts the Mobile IP registration for it. As shown in code snippet 4.17 the *start* function takes two parameters, the *session_address* and *home_agent_address*, which must be obtained in some way, either from command line, reading them from a configuration file, etc. The *start* function returns true on success or false if cannot use the *Personal Agent* to connect to the *Home Agent*. The *session_address* is the IP that we want to use and the *home_agent_address* corresponds to the

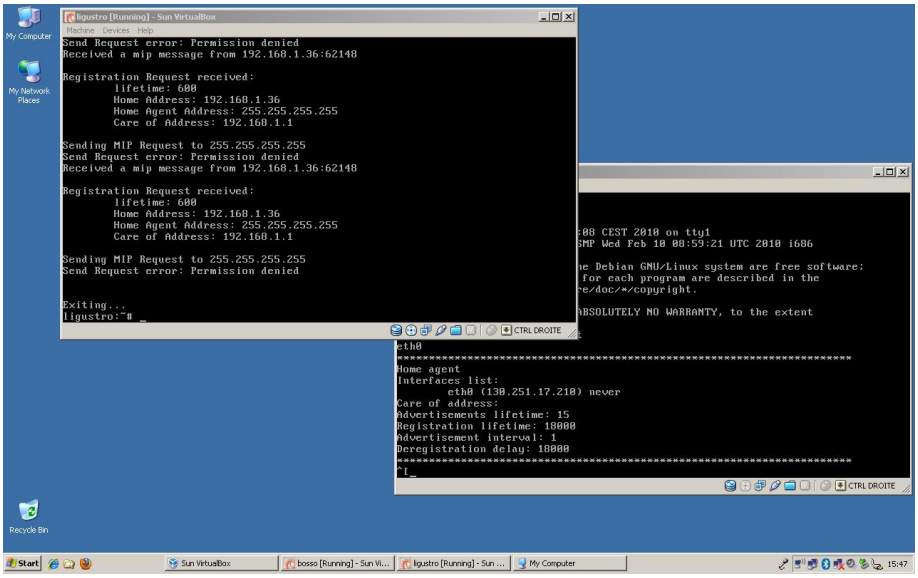


Figure 4.25: The Mobile IP servers running inside virtual machines, with a Linux distribution as operating system.

server that will register the session address.

```
1 std::cout << "sessionAddress:" <<sessionAddress.c_str()<< " homeAgentAddress:"<<
      homeAgentAddress.c_str() << std::endl;
2 if (!start(sessionAddress.c_str(), homeAgentAddress.c_str()))
3 {
4  std::cerr << "Error communicating with Mobile Session Service." << std::endl;
5  return false;
6 }
7 if(!checkRegistrationMobileSession())
8 {
9  std::cerr << "checkRegistrationMobileSession returned false." << std::endl;
10  return false;
11 }
12 return true;
```

Code 4.17: Starting the Mobile IP on the client, connecting to the Home Agent.

With the session address or more correct the *Personal Address* tells the system to use this instead of its host IP address. The registration of the *Personal Address* is shown in code snippet 4.18. The registration process takes some time for completion, however the function returns immediately (to avoid blocking application), therefore a time-out structure is used.

```
std::cout << "Probe Mobile Ip registration" << std::endl;
bool registered = false;
while(!timeOut)
{
 registered = check_registration() == 0 ? false : true;
 if (registered)
  break;
 Sleep(500);
}
if (!registered)
 std::cerr << "Mobile IP registration error." << std::endl;
return registered;
```

Code 4.18: The Run function within the Node class.

Where the start function tells the client device to use a certain IP address, the registration tells the *Home Agent* to have the given IP address corresponding to the underlying media access control (MAC) address. To unregister the session address and basically remove it from the device, a single call to the *stop* function is required.

In this setup we have the client with a specific Personal Address that we can take from device to device. This is simply done by having another client starting and registering, which will change the registered MAC address at the *Home Agent* and therefore the server streaming is automatically redirected. This implementation however supports only one client, in fact the server simply wait for an incoming UDP packet at the right *port* and will start sending out UDP packets containing the 3D rendering output to the address it got from the incoming packet.

In summary switching between devices is a client executing the register function. When and how this happens is done using the *Localization Management* or the *SAIL* system. At each device there is a Zigbee module, and we identify the device by its Zigbee module. Each module has an identifier and all together construct a *mesh network* based on the least power consumption between modules. The user carries a Zigbee module, which will automatically connect to the nearest Zigbee module. By looking at the connecting modules, we can extract which device should be active. This has to be communicated to the client application as otherwise no registration will happen. Albeit a bit crude, the solution to this was using a *http* server publishing the active module by its identifier. The client application polls every half a second the http server to check if it is still the active device. In case the identifier changed, it will call the stop function and therefore stop receiving the streaming from the server. Another device that then corresponds to the changed identifier will start and register, continuing the visualization of the stream.

The third aspect of this scenario was the security. A digital rights management (DRM) licensing system was used to authenticate the device to receive the stream. It did not however perform encryption on the streaming itself. Figure 4.26 shows how a session take over is performed.

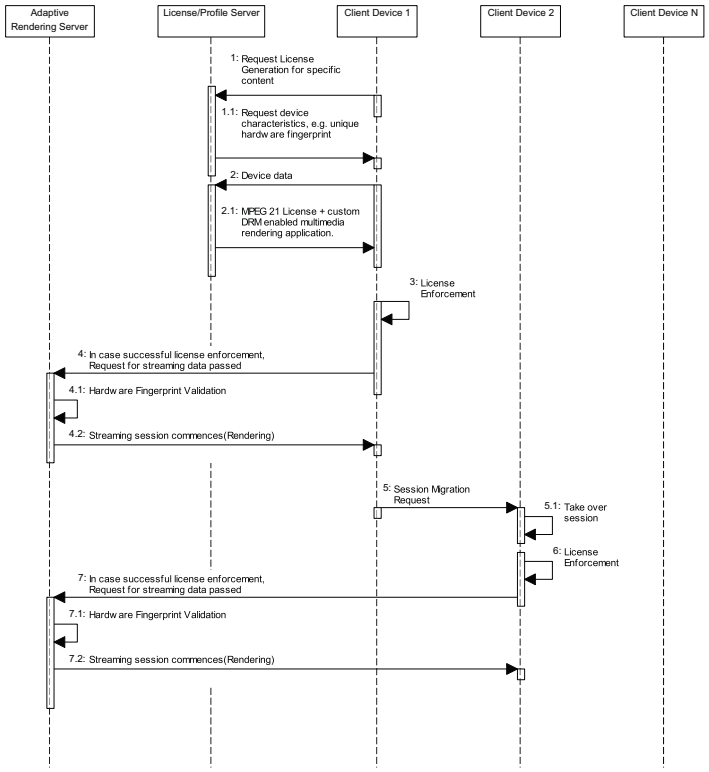


Figure 4.26: Switching between devices using security enforcement and session takeover.

4.5.1.1 Interface Adaptation

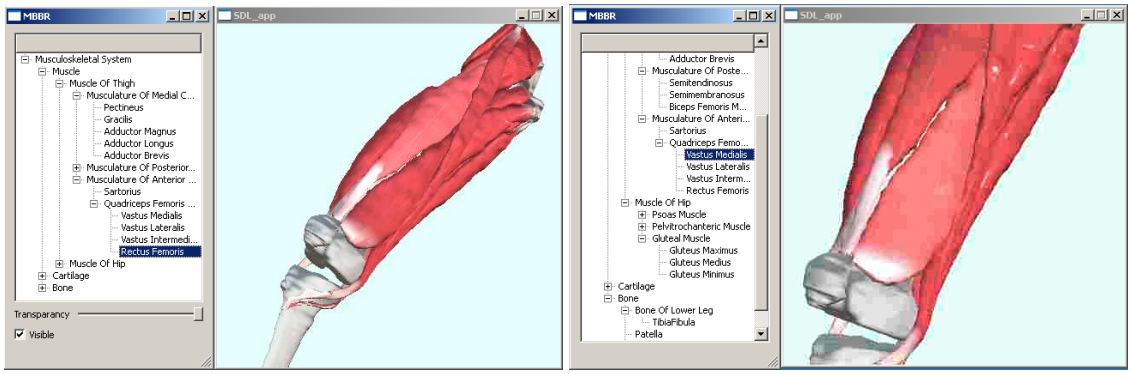
In order to support interface adaptation for different devices, we utilized the Qt framework for the rendering of the interface, and used the interface language for describing the interface. The content for rendering was provided by the 3DAH project (section D.2) and contains a 3D bone and muscle model of the leg with an ontology. For designing the interface the Qt Designer was used, which then saves the interface in an XML formatted file. A minimal example is shown in code 4.19.

```
1 ...
2  <widget class="QWidget" name="centralwidget">
3   <widget class="QTreeView" name="ontologyView">
4    <property name="geometry">
5     <rect>
6      <x>0</x>
7      <y>0</y>
8      <width>256</width>
9      <height>192</height>
10    </rect>
11   </property>
12  </widget>
13 ...
```

Code 4.19: Qt Interface language.

When a device sends a *profile* event to the server, which basically contains an identifier upon

which the server sends back the associated interface file to the client. This is illustrated in figure 4.27a and 4.27b where the laptop for example has control over the transparency of the muscles and the pda does not. In this case the interface changes are minimal but serve as an example. It should be noted that the server does not depend on Qt itself, rather it just sends the requested data to the client.



(a) Laptop interface **(b)** PDA interface

Figure 4.27: The server provides a slightly different interface depending on the device profile.

The client reconstructs the interface from the received data using the *QUiLoader*. However this does not have any functionality since there are no bindings to functions etc. and furthermore the interface description does not describe what this functionality will be. We must therefore intercept the events from the widgets and send them back to the server, where the actual application is running. The *Event Manager* on the server will then execute the functionality bound to it. This is shown in code snippet 4.20

```
1  inline void GuiLoader::load(unsigned char * interfaceData, unsigned long size){
2   QByteArray ba((const char*)interfaceData, size);
3   QBuffer buffer(&ba);
4   delete formWidget; //delete old interface
5   formWidget = loader.load(&buffer, 0);
6
7   QList<QPushButton *> qButtons = formWidget->findChildren<QPushButton *>();
8   foreach (QPushButton *qButton, qButtons)
9    eButtons.push_back(new EPushButton(qButton));
10
11  QList<QTreeWidget *> qTrees = formWidget->findChildren<QTreeWidget *>();
12  foreach (QTreeWidget *qTree, qTrees)
13   eTrees.push_back(new ETreeWidget(qTree));
14
15  QList<QSlider *> qSliders = formWidget->findChildren<QSlider *>();
16  foreach (QSlider *qSlider, qSliders)
17   eSliders.push_back(new ESlider(qSlider));
18 }
```

Code 4.20: Extract basic interface elements and overrule them with custom implementations.

Upon receiving the interface description the load function is called. There using the default *QUiLoader* the interface is constructed. After the construction we traverse the interface and filter

out all widgets and create our own widgets wrapping around each. In the presented code can be seen that only *push buttons*, *tree widgets* and *sliders* are supported. Normally every Qt offered widget has to be wrapped around in similar manner in order to intercept its events and inform the server. Code 4.21 shows an example *wrapping* implementation.

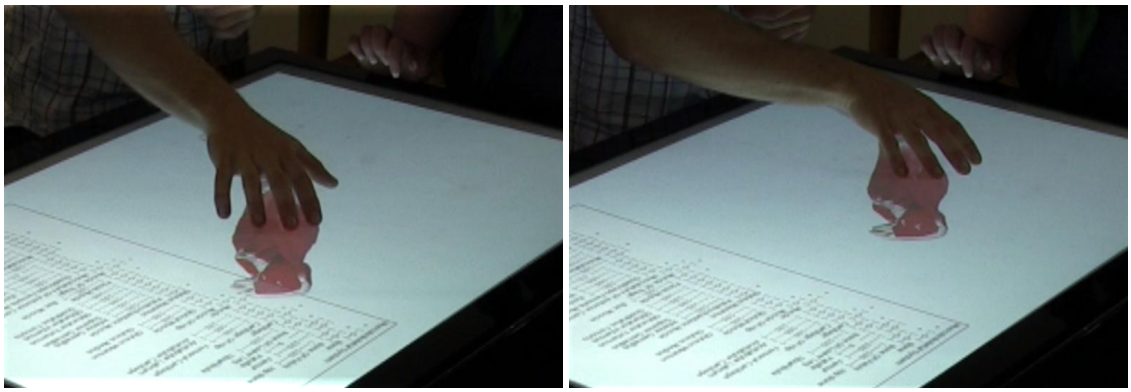
```
1 ESlider::ESlider(QSlider *slider): m_slider(slider){
2  connect(m_slider, SIGNAL(sliderMoved(int)), this, SLOT(SliderMoved(int)));
3 }
4
5 void ESlider::SliderMoved(int sliderValue){
6  DataPacketPtr dataPacket = HerdDataPacket::create();
7  dataPacket.addField(UserProtocol::GUIEVENT)
8  dataPacket.addField(m_slider->objectName().toStdString());
9  dataPacket.addField(sliderValue));
10  m_networkNode->handlePacket(dataPacket);
11 }
```

Code 4.21: Wrapping a slider widget and connect its event to a custom function.

The base slider is kept by pointer value and a Qt connect function enables the communication of events to the desired function. In this case, whenever the slider moves the *ESlider::SliderMoved* is called. A packet is then created, providing first the user protocol identifier, followed by the name of the widget (in Qt all widgets have an unique internal name) and last the new value. This is then send to the server which will handle the rest and the client application will see the response within the 3D rendering.

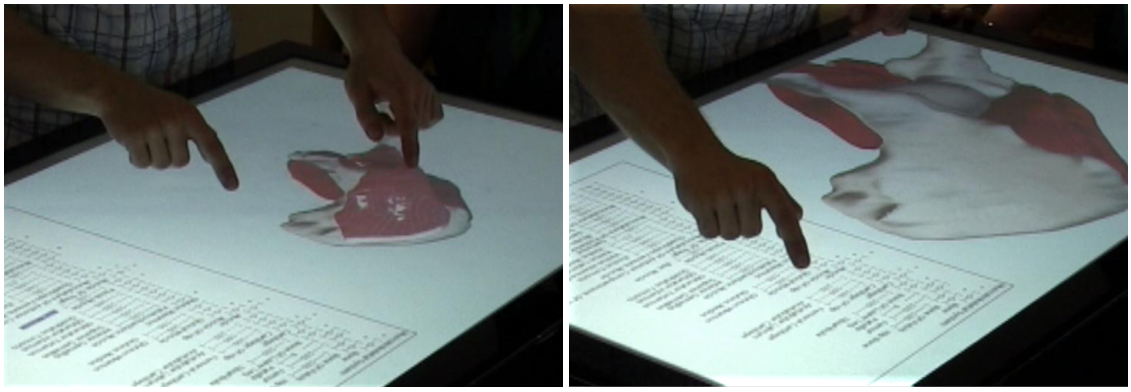
4.5.2 Data sharing among switching devices

In this section we illustrate how the three components of the system, interactive table, mobile phones and 3D graphics, have been implemented to deliver the integrated interactive experience for medical students. The integral component of the system is the multi-touch interactive table which is basically a composition of various technologies (infra-red light, infra-red camera, and projector) integrated together in one box and coordinated through computer-vision software [114].



(a) Moving - five fingers. **(b)** Rotating - four fingers.

Figure 4.28: Interactive table moving and rotate.



(a) Zooming - two fingers. **(b)** Selecting - one finger.

Figure 4.29: Interactive table zooming and selecting.

The table uses the Windows operating system and inherently supports single-pointer events. Many solutions have been introduced to overcome this limitation and the TUIO framework is one example. It defines a protocol and an API for tangible multi-touch interfaces which have been integrated into OSG to translate multiple touches on the table surface to corresponding single-pointer events that it handles. In this case, moving five fingers along the surface of the table triggers the *move* action of the 3D model as in figure 4.28a. Moving four fingers will rotate the model in the appropriate rotation angle, figure 4.28b right. And dragging two fingers towards each other zooms out the model and dragging them apart zooms in, figure 4.29a.

A Java ME[15] application has been implemented to handle the detection of and the connection to the server. Both NFC and Bluetooth[16] have been utilized to accomplish this. NFC is a short-range high-frequency wireless technology which utilizes touch as a way of interaction between two phone terminals, or between a phone and an NFC tag. The system here adopts the second case where a student touches an NFC tag on the table with her NFC phone to identify the Bluetooth address of the server. Then the mobile application initiates the authorized Bluetooth connection and sends the 3D model across in XML format which is then parsed and rendered by the 3D viewer.

The transferred model is a 3D model of the upper leg muscle and bone construction and it is part of the *3D Anatomical Human* project[17] whose purpose is to study the anatomical and functional aspect of the musculoskeletal system. A medical student can interact with the model's different parts by touching them directly on the table. She can tap the muscle or bone of interest on the 3D model and see its corresponding name on a side menu that is displayed along with the model on the table. In addition, the user can disable parts of the 3D model in order to help her closely diagnose specific bones or muscles. And she can even control the transparency of any of its parts to allow a see-through experience. Both these actions are triggered when one finger touches the surface of the table as figure 4.29b above shows.

[15]Java micro edition for mobile devices `http://www.oracle.com/technetwork/java/javame`
[16]Bluetooth wireless communication system `http://www.bluetooth.com/`
[17]Project 3DAH section D.2

4.6 Service Distribution and Render Context Switching

In this section we discuss the methods and design for modifying the existing application in order to support the goals as stated in the first section.

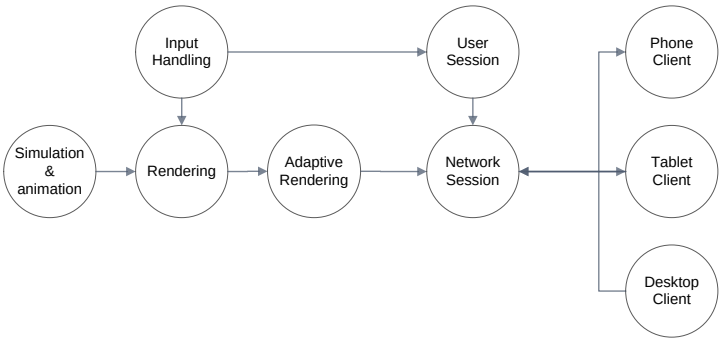


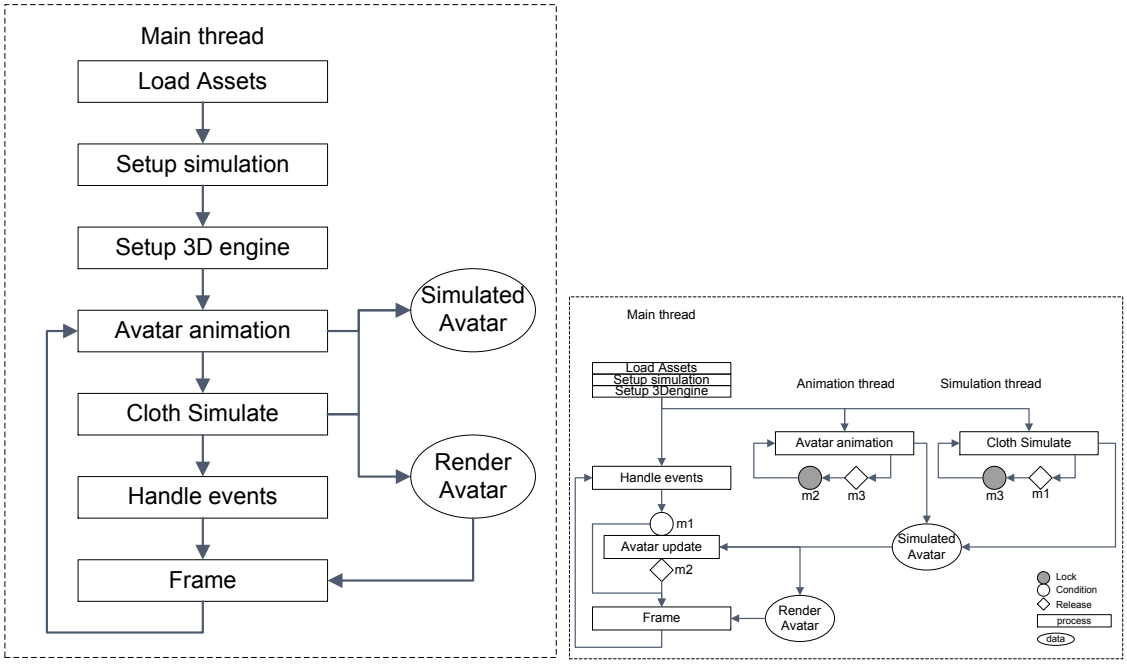
Figure 4.30: Abstract system overview.

The VTO is represented by the *Simulation & animation*, *Rendering* and *Input handling* in figure 4.30. It has been separated into several nodes as will be shown later that each of these parts are handled in separate threads, whereas the initial instalment is a straight forward single threaded desktop application (aside from the inherit threading capabilities from OSG and Qt). The *User session*, *Adaptive rendering* and *Network session* are partly application logic and are based on parts of the Herd framework. The user session contains the state of each user connected, this includes session identification, device-hardware profile, interaction-input capabilities. Thus all information needed for the rendering for a specific end-device, the optimal parameters for the adaptation, and identification of the networking session and connection. The adaptive rendering provides a couple of functionalities, first of, the *grabbing* of what is being rendered, a frame, as this has to be send to the end-client through the network. Secondly, the compression of the frame into a more optimized data size for sending over the network and to be handled by the end-device. Figure 4.30 shows three different clients (phone, tablet and desktop), where each of these offer a different experience in terms of input, screen size and mobility. In the following sections we analyse the current application and discuss the conversion into a multi-user service.

4.6.1 A multi user approach for a collaborative view

The base implementation of the VTO application is a single window application, as presented in figure 4.33, rendering in an OpenGL context and accepting solely mouse pointer interaction. All direct interaction with the avatar customization is provided through Qt widgets, which are visualized within the 3D environment (OSG). Aside from interaction with the widgets, any other mouse input is handled by the OSG manipulator for the virtual camera. Meaning when dragging the mouse anywhere else than on a widget makes the camera rotate or pan for example.

The performance from a single threaded application is the accumulation of processing time for each of its components and often is held back by some of these, which in this case is the VTO



(a) Normal sequential application flow. **(b)** Separation of the simulation and rendering.

Figure 4.31: From sequential to parallel, separating the simulation and rendering.

simulation. Therefore in order to be able to handle multiple users we need to decouple the simulation of the avatar and garments from the rendering and input handling. And while we are at it, the avatar animating component is taken apart also, since the animation and simulation have the dependency to be executed in sequence. This is shown in figure 4.31a and 4.31b, where on the left the application is limited to about 25 to 30 frames per second depending on the complexity of the garment used by the simulation. For a single user application this might be acceptable, however serving the same simulation and rendering to a multitude of users is going to increase the cycle time significantly which results in a lower response and overall interaction rate. On the right the simulation and animation are both placed into their own respective thread, and are slightly executed in sequence, as first the animation thread is setting the new pose (the next animation frame for the underlying skeleton) for the avatar, then the simulation thread handles the deformation of the garments. The rendering is continuously as the interactive widgets are also rendered into the same 3D context, therefore the updating of the avatar for rendering is based on an update condition, which is set after a simulation cycle. If this condition is not met, the current unmodified avatar will be rendered again.

The next step is to be able to render for multiple users, which is presented in figure 4.32. This we do by providing multiple virtual cameras and at each rendering cycle looping through the active users, set for each user the proper context (virtual camera, interactive widgets). The OSG toolkit offers functionality to create multiple views (composite viewer), and therefore being able to initialize for each user a unique view, while still sharing the data among each OpenGL context.

Each view has its own event queue which takes the input given by the user and is handled within

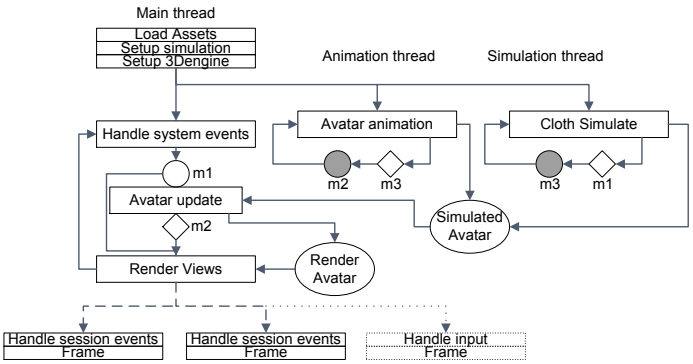


Figure 4.32: For each user a separate frame needs to be rendered, with also the separation of the events between sessions.

the given context. However normally each view is still rendered locally, meaning that the input comes from directly connected devices at the computer running the application. In our scenario the input is provided, marshalled into events, from each client through the network. Therefore we need to handle in each view the events linked to the given user session and convert them into the proper OSG events. Furthermore higher level events still need to be handled at the main thread, such as accepting new sessions and attach a newly created view.

The client implementation is more straightforward and is closely designed to the MVC model. Where there are two controllers, the application logic for handling user input and global interaction and the second controller handles the network events, render updates and profiling. The network part handles incoming stream packets, into a buffer, which is read by the decoder. The decoder flags for update after each decoded frame. The rendering measures the time between each frame and how long it takes for processing a frame. This value is given back to the server to adjust the frame rate. The server uses a relaxation in order to filter out spike frame rate adjustments. This for example happens if the Wi-Fi connection is lost for a second, or the application stops responding, but the network is still buffering frames. Aside from the client frame adjustment, the server also tracks the frames it had send and can fill up a micro-buffer (3 to 4 frames), if the micro-buffer is full the server will stop rendering for the given view port. This kind of "hard frame drops" prevents the encoder and decoder from producing artefacts, since no frame was lost between them.

The tablet and desktop client are programmed using C++ language and uses SDL software and hardware acceleration (based on the OpenGL API for rendering). Supporting both VP8 and MJPEG as input from the server. Albeit VP8 is preferred and can be decoded using hardware acceleration and without for older hardware. The smart-phone client device is a Microsoft Windows Phone 7 (WP7), and therefore limited to use the specialized WP7 software development kit (SDK) support the C# programming language since no native SDK is provided, and this limits us to use MJPEG for streaming as JPG decompression is supported while VP8 is not. Utilizing Microsoft's XNA framework (based on the Direct3D rendering API which in turn is part of DirectX) the JPEG image is decoded and streamed into a texture. This texture is then rendered as a full screen 2D sprite onto the phone's display.

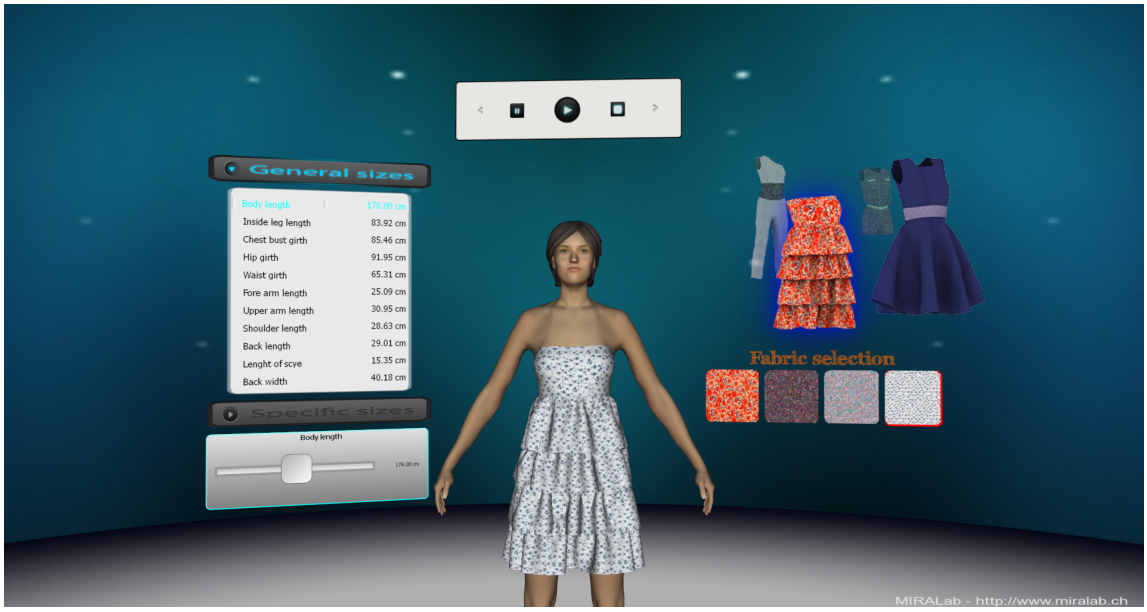


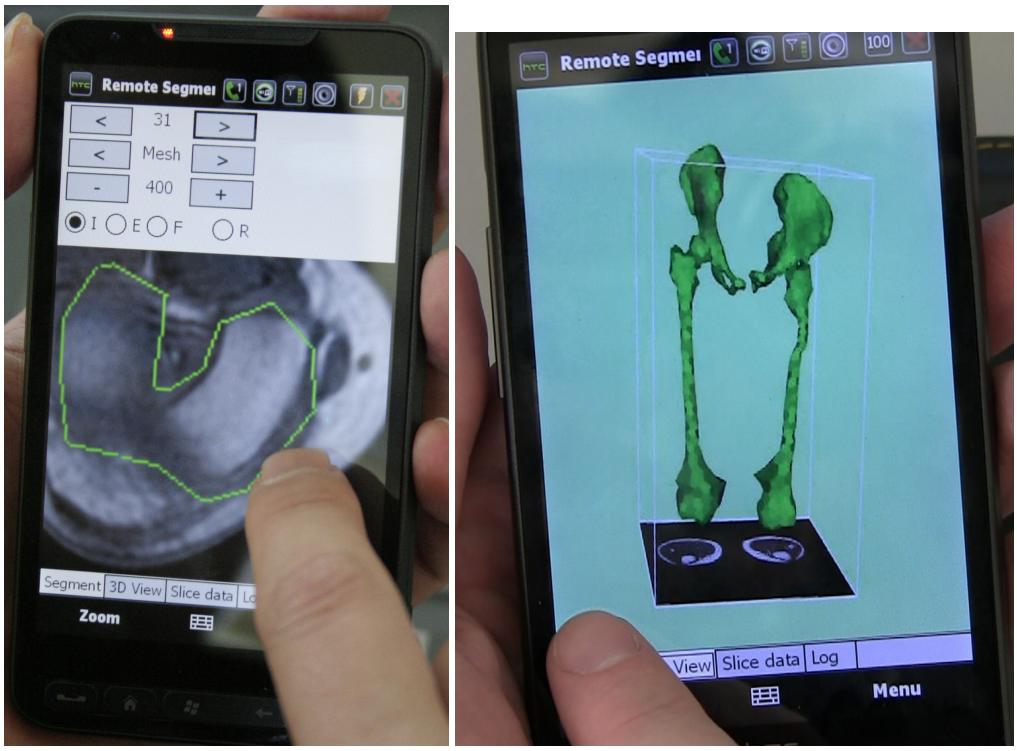
Figure 4.33: The VTO example showing a screenshot from the client-program.

4.7 The Collaborative MRI Segmentation

The server holds the semantics and constitutes the 3D processing modules in Medical semantic layer (figure 3.20), while subscribers share and edit the polymorphic presentations. It provides a 3D rendering view of the surface model and a 2D view on a per user base. In this current implementation, the 3D rendering is only for illustrative purposes and not used in the collaborative editing. The views can be synchronized which means that all the users observe the same slice. This particular mode is appropriate in the *teacher-students* scenario described in section 5.6.1, in which the teacher first segments while the students only observe as shown in figure 5.11. A second mode consists in letting the users to view a slice independently to the other users in the same session. This option is essential for the second case scenario, in which experts can segment different parts of the same dataset. The 2D viewer consists of the standard image manipulations controls - LUT change, B/C, scaling, and annotation, in addition to be able to change image slices (moving up and down in the image stack). Each subscriber is assigned to a unique annotation colour and can either add or remove constraint points on the image (via mouse selection) as a segmentation parameter. Per iteration cycle, the publisher compiles all the parameters and executes the control based on time-stamp (first-come, first-serve) and event management in Interaction substrate component. Multiple manipulations are possible, i.e., LUT change and segmentation change, in a single iteration cycle.

A C# implementation of the framework was used to enable the MRI segmentation on a mobile device (hardware (xi). Due to limitation of previous methods on interface adaptation, the whole client was build from ground up with specific functionality supporting touch interaction and switching between MRI segmentation and viewing the 3D model. This segmentation is illustrated in figure 4.34a and on top of the interface is a tabbed widget where the user can switch

over to the 3D view, figure 4.34b.



(a) Mobile MRI segmentation display. **(b)** Mobile 3D rendering display.

Figure 4.34: Collaborative segmentation on a mobile device.

For desktop PC and the UMPC (mobile devices) two separate client applications were build. An application dedicated to the MRI segmentation itself, and the other to the visualization of the 3D content, which is based is identical to the previous presented application showing a 3D rendering. The only minor difference is the actual rendering, where previously OSG was used, here the VTK libraries were used. In end effect however it didn't change much as both are OpenGL based and therefore the grabbing of the rendering frame stays the same. Figure 4.35 illustrates both applications on the UMPC.

The implementation are very basic in terms of user interface on the UMPC and desktop and require a keyboard and mouse for interactions. For rendering SDL was used and the transmission of MRI images were provided in non-compressed 8-bit grey values, since the original images are 16 bit grey several controls were provided to the user to set a *window* on the source image. The reason for the downcast to 8-bit was mainly because most applications working with MRI data are doing this. Thus as a requirement from the users doing the segmentation (also most screens are only capable of showing 8 bit colour depth). VTK provides functionality to do this easily, as well as for providing *cut through* of the mesh, and showing it contour on a specific slice. A separate server application was built for providing the actual MRI slice, offloading this process from the simulation application. In general it concerns the exact same application, but with mutual exclusion of functionality (simulation and MRI slice generating).

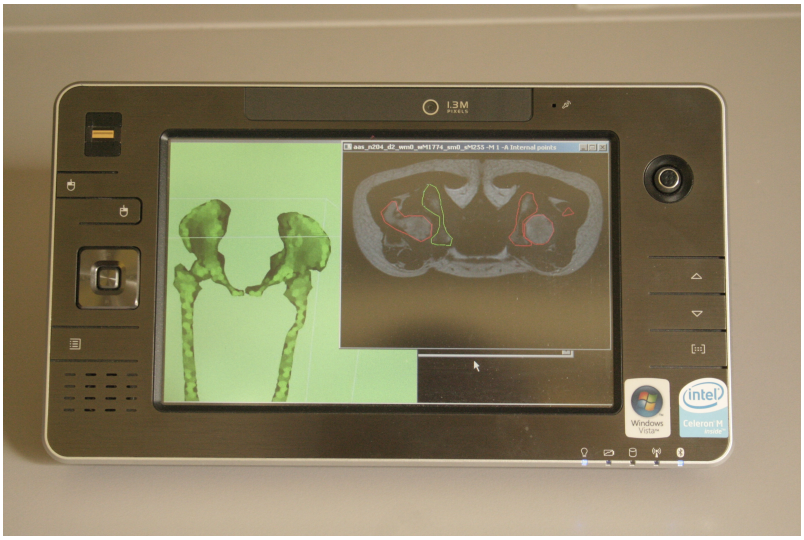


Figure 4.35: Interface for the collaborative segmentation.

The rendering on the client is provided into two layers, the background layer showing the MRI image and the overlay of the annotation data. Both received in separate threads, as two connections are required (one MRI slice and the other the simulation). Furthermore the annotation layers keeps a state of the user session, such as slice index, active mesh, contour of the meshes and the interaction points. The contour and the interaction points are plotted as points on top of the MRI image in the render cycle. The interaction points are depending on their function drawn with a different colour as well as the active mesh and non-active meshes (figure 4.35).

Although the experiment shown in section 5.6 is based on the application just described, we find it noteworthy to mention the continuation of this scenario and present a more sophisticate version and continuation on the concept of the collaborative segmentation, which is shown in figure 4.36.

The concept was well received and the benefit of having collaborative segmentation tooling was evident. We therefore looked a more complex interaction and overhaul of the primitive interface. Instead of points we look at a sequence of points that offer a relation between each other. From a simple linear line, curved as well as custom defined point to point interpolation. Furthermore the previous application was a specific tool to influence the simulation, in this case, there can still be a simulation driven interaction, however, as a request it is possible to leave out the simulation and have a manual segmentation instead. The implementation uses the publish-subscribe data sharing method as described in section 4.2.3.3. Which furthermore enhances the flexibility of the system, as other services can simply subscribe to the publisher and perform interactions. In this case there are four server applications, the *simulation*, the *MRI slice provider*, the *mesh contour provider* and the *publisher*. On the client side a single application is provided for the segmentation (as shown in figure 4.36) and a separate 3D viewer.

The simulation is currently still required (although it doesn't have to do anything aside from loading the initial meshes), and will connect to the publisher providing the mesh data data. Any

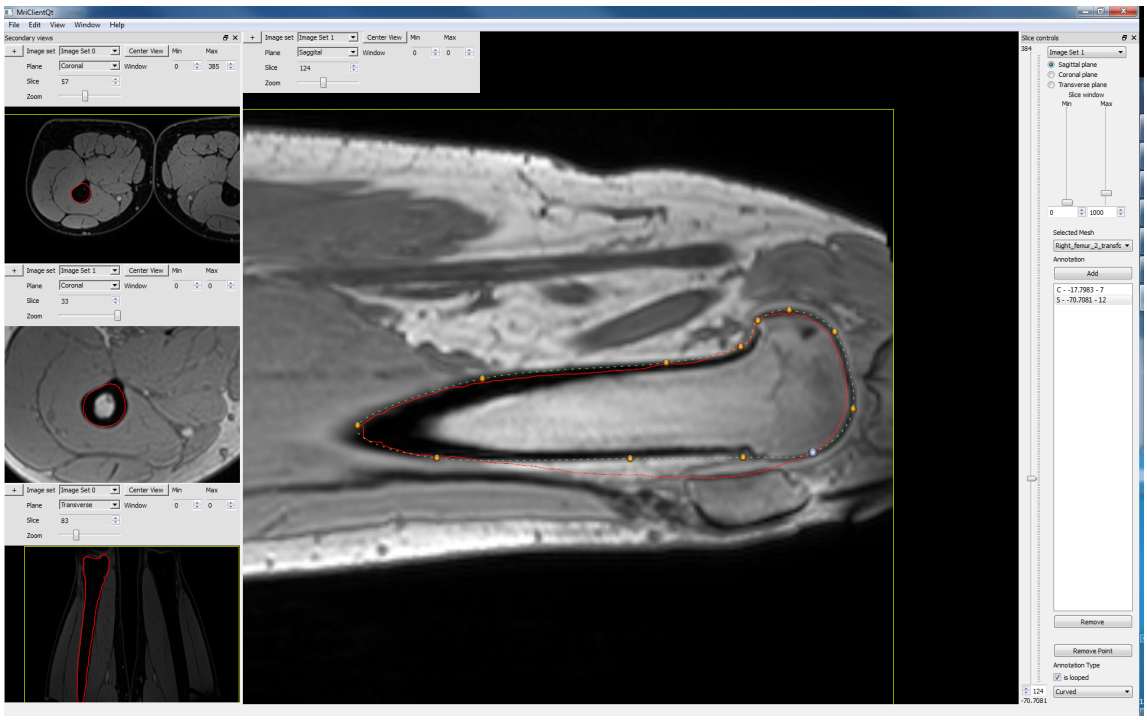


Figure 4.36: Interface for the collaborative segmentation.

modification to the mesh are pushed to the publisher. The MRI slice provider does not connect to the publisher and clients connect directly to it. It is identical in function to its *ancestor* but this time rebuild from scratch containing only the functionality to provide a slice in the MRI stack. Whenever a client changes from slice it will make a request to the MRI slice provider, which in turn keeps track of the user, using the TCP session as its basis. For each user it has a VP8 node, this offers the functionality of fast traversal in the MRI stack, reduces bandwidth and increases interactivity. It is however the client that makes the request for a *video frame* or *raw frame*. This depends on the rate of interaction, if the interaction rate per second becomes lower then a certain threshold it will request a *raw frame*. This shows a different approach on the presented design as discussed in section 3.2.3. For the mesh contour, which is now provided by the *mesh contour provider*, the approach is similar, as the client directly connects to it. The *mesh contour provider* keeps track of the active users as well, but it actively sends mesh contours whenever the mesh changes in the publisher. Meaning the *mesh contour provider* connects to the publisher and subscribes to the meshes supplied by the *simulation*. Currently a fixed refresh rate is used of five frames per second. The contour provided is a list of points which are rendered as a line loop. The client rendering is a mix between OpenGL and Qt widgets. The OpenGL rendering is adapted to be capable to handle the decoded *video frame* images coming from the VP8 decoding node, as well as rendering the *raw frame* and supports the use of shaders. Shaders also enable the possibility of client side customized visual enhancements e.g. sharpening and different filtering. The Qt widgets are used for the contour rendering and the annotations. Active annotations get interactive widgets enabling the user to select, move and remove points. Overall the use of Qt provide an overall better user experience as the rendering is more robust.

4.8 Collaborative Services with shared Data Models

4.8.1 Threadz Plugin Library

The *Threadz* library contains service nodes which are part of the use-case scenario as described in section 3.10. The main service revolves around the Threadz Library, created by Kevelham [75], which provides a GPU driven cloth simulation for real-time purposes. Four nodes are provided, the manager, the server, the simulation and the creator. In this use case the focus lies on inter service communication, where the manager maintains several server services, each server service is uses one or more simulation nodes. The output of the simulation is the simulated 3D model (thus an array of vertices).

4.8.1.1 Managing and spawning server services

The manager service functions the same way as the VTO manager as shown in figure 3.17, capable of maintaining a set of server services and load balance these among several physical servers. Users that connect to a manager are being redirected to an empty server service, upon which they get information for directly connecting to the free server service.

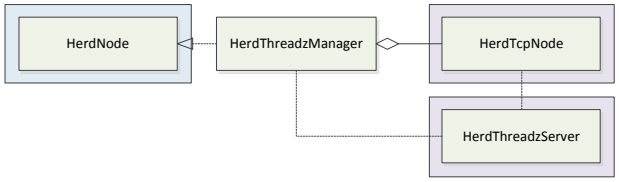


Figure 4.37: Threadz Manager diagram.

The service itself therefore is quite small and all functionality is contained in a single class *HerdThreadzManager* (figure 4.37). The manager creates two additional *HerdTcpNode* nodes, which are child nodes. One node functions as the communication for users and the other for communicating with the servers. Although this separation is not necessary it does simplify the setup and gives a cleaner overview of the functions. A client connects to the manager on the first TCP connection, upon which the manager can simply send a "broadcast" message to the second TCP connection. Without worrying on searching through and maintaining information on each server, as each server can reply back to the manager if it is free. The first available server is taken, and the connection information to it is given to the client. The client in turn connects to the given server and provides a device profile.

4.8.1.2 The server service

The server does not directly perform the simulation itself but handles the user input and interfaces with the actual simulation by adding *HerdThreadzSimulation* nodes as child objects. Similar to previously presented protocols, the *HerdThreadzProtocol* extends the base protocol with several new event identifiers. These are represented also in figure 4.38 as *HerdCommand* objects.

The *HerdThreadzHandler* translates incoming events into *HerdCommand* objects which are then provided to the *HerdThreadzSimulation*.

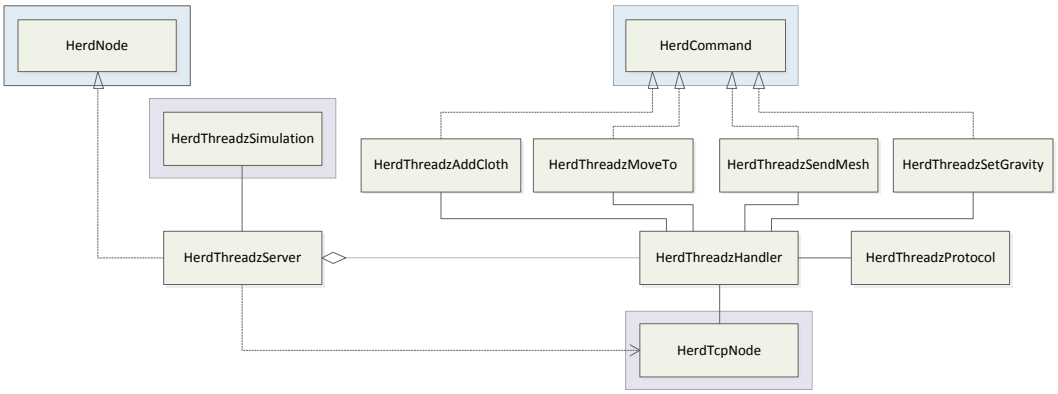


Figure 4.38: Threadz Server diagram.

The simulation service as shown in the abstract class diagram figure 4.39 maintains a list of *Cloth* which through the *ClothMechanical* are provided to the actual simulation process. The LibThreadz library focuses on the simulation of cloth material using NVidia CUDA[18] for hardware acceleration, as a fall-back it can also perform the same simulation directly on the CPU, however at a considerable drop in simulation cycles per second. In order to give a better impression of the

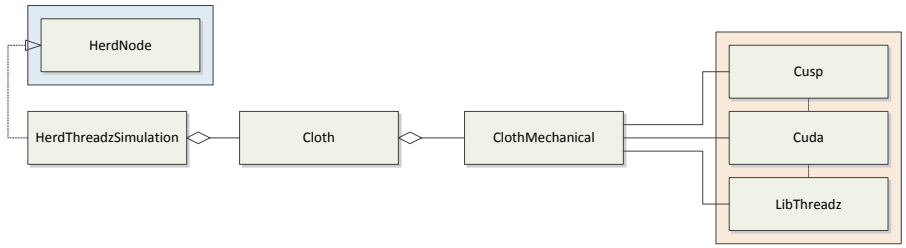


Figure 4.39: Threadz Simulation diagram.

difference between executing calculations on the CPU and GPU table 4.4 provides a comparison and figure 4.40 a visual of the actual test. The evaluation was performed by executing the code paths from the Threadz library on a workstation[19]. As previously mentioned the VTO also contains a cloth simulation, which is purely CPU based, however this is not taken into comparison since the implementation differs, e.g. using different solvers and mechanical parameters.

4.8.1.3 Cloth creator

From the scenario given in section 3.10 the *Cloth creator* service takes 2D patterns and converts this into 3D meshes. These meshes can then be given to a simulation service.

2D patterns which are then translated into 3D meshes using a Delaunay triangulation algorithm. The *Cloth creator* service provides a function to translate a 2D pattern to a 3D mesh.

[18]NVidia CUDA parallel computing platform `http://www.nvidia.com/object/cuda_home_new.html`
[19]Workstation used for *Threadz* test: Intel Core i7 X 980 CPU with a NVIDIA GeForce GTX 480. (v)

Table 4.4: Simulation timings for both the CPU and GPU back-ends for a varying number of elements.[75]

Elements	CPU Tim	GPU Time	Speed-up
651	5.57s	0.55s	10.04x
1003	8.57s	0.55s	15.44x
2519	22.20s	0.56s	39.78x
4754	41.48s	1.07s	38.67x
7704	68.36s	1.18s	58.06x
12699	108.77s	1.60s	68.16x

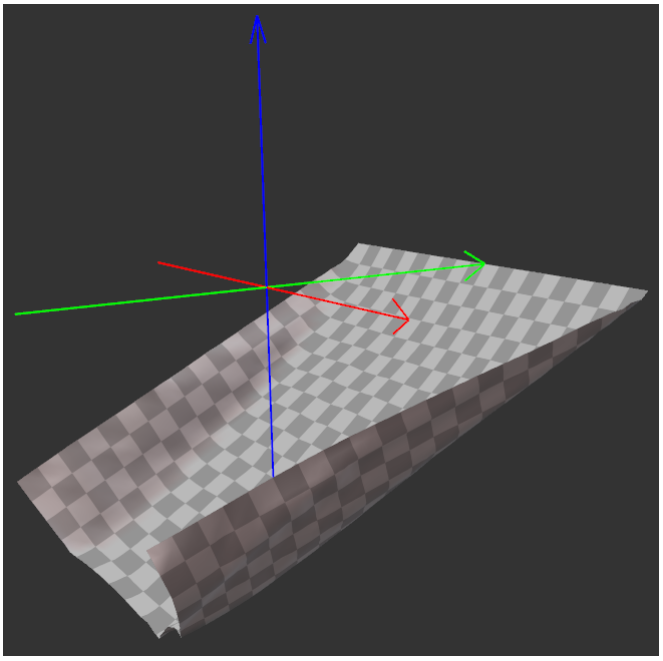


Figure 4.40: A square fabric, clamped at the top edge, falls down under its own weight due to the effect of gravity.

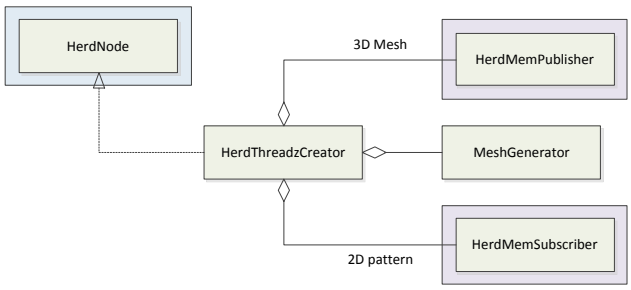
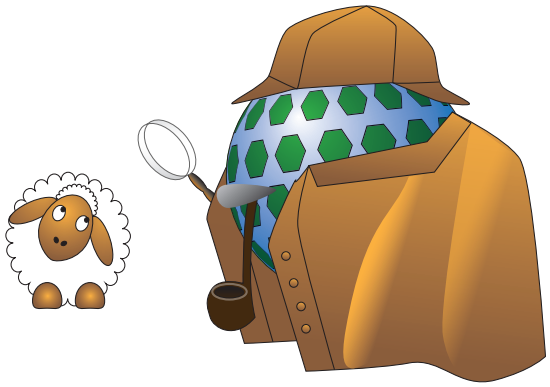


Figure 4.41: Threadz Creator diagram.

Chapter 5

Experiments and Results



5.1 Motivation

In this chapter we look at the deployment of each application scenario which were implemented using the proposed architecture. In the first application we look at one of the core aspects of the framework, which is the remote rendering for low-end devices. We measure the frame-rate and responsiveness between client and server. The second application scenario introduces the use of Augmented Reality and a *heavy* simulation used for mesh deformation, this forces the setup to be dispersed between end-device and server. We look at the performance and feasibility of two different setups. The third application scenario handles another core aspect of the architecture, flexibility for deployment and dependencies on third party technologies. With a strong focus on *User Centric Media*, here we show the feasibility of seamless session hand-over from device to device. Retaining an uninterrupted the 3D media content streaming. The fourth application shows the scalability of the proposed architecture. Introducing multiple services and maintenance of these services, with per session multiple users connected and inherently from previous application scenarios support for heterogeneous devices. With the fifth application scenario we introduce the Telemedicine domain and focus on collaboration with a more strict common goal. Two different types of collaboration were utilized and tested upon.

5.2 Remote Rendering for Low-end Devices

In this experiment we look at the deployment of the client on a low end devices with the main focus on the interaction rate and the frame rate. A traditional server client setup is used in a LAN environment and a local Wi-Fi access point.

5.2.1 Deployment

A very early on experiment showing the initial proposed algorithm for adaptive rendering is presented first. The client application has been installed on a low end mobile device UMPC(hardware (vii)) and the server application on a desktop PC (hardware (ii)). The time measured is the response time in a local LAN environment. The adaptation regulates the quality of the image encoding resulting in a faster response time, as shown in 5.1.

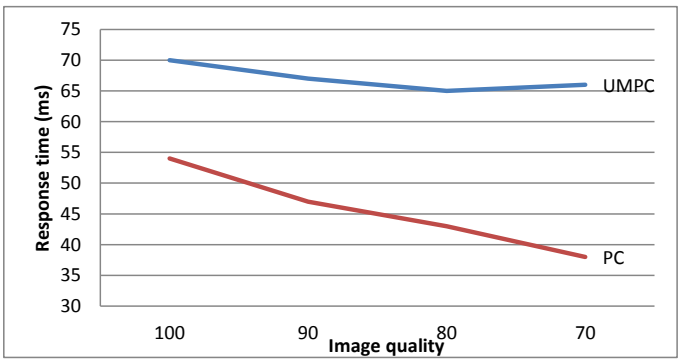


Figure 5.1: The response for the UMPC and PC clients.

As the results were done on a preliminary version of the framework a second experiment has been conducted with the same hardware. In figure 5.2 shows the setup, using the last version of the framework and the Java client implementation supporting the VP8 codec (where the previous was JPEG only) and the server running an VTO instance. The rendering performed at *640x480* and runs on the server about 20 frames per second. Limiting itself in frame rate with a little overhead over the response rate from the client. Depending on the settings the client frame rate is adapted. The client is utilizing about 30% to 60% of the CPU and has a refresh rate of about 10$\tilde{1}$5 fps. Big visual changes in the scene will make the VP8 codec make a new key frame resulting in spiking in bandwidth usage and drop in fps on the client side as it takes longer to process. The interaction rate still shows the same behaviour as presented earlier.

Figure 5.2: Adaptive Rendering, on the left is the server window shown (1080p screen) and on the right the UMPC

When removing all limiters the server will run at its maximum, which is with the VTO simulation about 30 fps and the client runs at 20 fps, 100% CPU usage. However the response rate drops down to a couple seconds and is uncontrollable as it fluctuates between 300 ms to 3 seconds. This is mainly because the server drops the frames beforehand instead of the client and is therefore flooding the client. As this is better for visual quality (no decoding errors on client because of missing frames).

5.2.2 Concluding Remarks

Interactive performance in terms of responsiveness is one of key challenging issues for interactive 3D applications. We introduced run-time presentation adaptation and dynamic interface adaptation mechanisms which aim to preserve the real-time interactive performance of 3D content, taking into account heterogeneous devices in user-centric pervasive computing environments. To support perceptual real-time interaction with 3D contents, temporal adjustment of presentation

quality adaptation is used. It dynamically adjusts the quality of presentation on client devices according to the current device context. To overcome the inevitable physical heterogeneity in display capabilities and input controls on client devices, we provided a dynamic user interface reconfiguration mechanism for interaction with 3D contents. It can change the way how the interface is presented to the user (big screen or small screen bring several design issues with it) and adaptation to the user device input capabilities. In addition, functionality of 3D contents and rendering are dynamically bound with user interfaces at runtime according to profiles.

5.3 Remote Rendering with Augmented Reality

Similar to the previous setup, however now with the added AR. During the implementation (section 4.4) preliminary tests were conducted and showed that the first setup caused a great amount of lag, therefore we mainly look here at the second setup. Also it should be noted, rephrasing from section 3.4, that this application scenario was shortly no longer supported and continued on after these tests.

5.3.1 Deployment

We have used a desktop PC (hardware (iii)) as our server system and the UMPC (hardware (vii)) again as our low end client system. The VTO running on the server acts as our rendering content that has to be augmented with the webcam images from the client. The 3D content is a template female body consisting of approximately 7500 polygons and garments are made up of about 2000 polygons. The VTO library on its own runs at an average of between 30 and 35 fps. Any real-time changes to body sizes or length changes requiring animation adaptation do not have a significant impact on this frame-rate and neither does the size adaptation of the garments. The performance is mostly affected by the geometrical complexity of the garment as more elements directly relates to more computation. As stated before the current performance of our system is limited mostly by the camera hardware running at a fixed 15 fps for the UMPC. But tests show that without waiting for the web-cam image to update we achieve a frame rate slightly below 30 fps. The system setup can be seen in action in figure 5.3 and 5.4. The UMPC at the bottom shows our VTO on top of the marker that is visible above it. On the right-hand side the workstation that serves as the server is visible, as well as the used camera. In order to have a real-time experience the rendering resolution had to be kept quite low at 320x240 pixels. On the server side we could enhance it slightly by using anti-aliasing.

5.3.2 Concluding Remarks

With this experiment we have presented a remote augmented reality and simulation solution for a mobile VTO. We have demonstrated that it is both plausible and feasible to implement such a system, while achieving interactive or even real-time performance. The development of this prototype has shown us what the most important bottlenecks are. It has given us an idea of where we should focus on for further research in order to improve the system, both in terms of

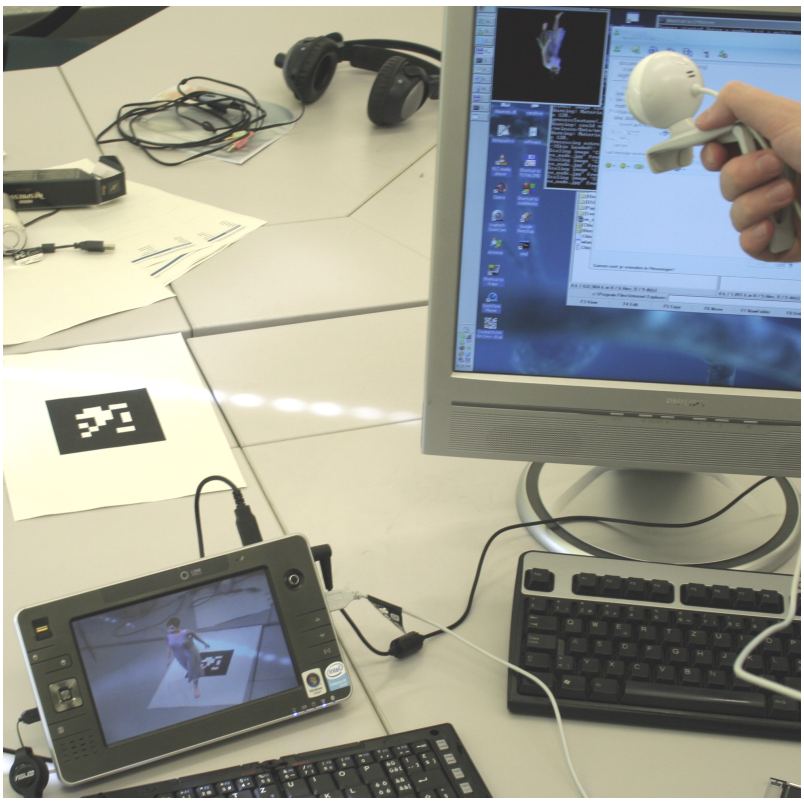


Figure 5.3: The remote AR VTO in action.

performance and usability.

Two areas that clearly could use some improvement are the increase of frame-rate and a higher resolution of the augmented result. This however is not just a matter of improved hardware. Research into better or faster compression techniques and the analysis and improvement of our networking middleware might allow us to improve the speed and quality of the results, thereby providing a more pleasant user experience. Computationally the cloth simulation has the biggest impact on application performance. It will be worthwhile to look into ways to improve its performance without degrading, or while improving, the accuracy. One possible avenue would be to implement this module, if not the entire VTO, as a highly parallelized GPU based application. Besides bringing improved performance it might also cut down hardware costs on the server side, thereby making our remote augmented reality system even more feasible. And with the advent of GPU server racks it might improve the scalability as well. But besides performance improvements it would also be interesting to investigate multiple user interaction, in order to create a mobile collaborative environment. The ideal would be to provide a remote but shared virtual world in which multiple users can interact in the same complex environment, without the requirement of complicated or expensive hardware on their side.

Figure 5.4: Image blending of the camera view with the 3D rendering.

5.4 Adaptive Rendering with Dynamic Device Switching

As this application scenario was divided into two sub scenarios, two separate experiments were conducted on the use of the presented technologies. The complex nature of these scenarios make it difficult to replicate as there is a great dependency on the 3rd party components.

5.4.1 Real-time switching device

Presented and tested at a workshop[1]. As previously mentioned, due to complexity this setup has been replicated once thereafter at the project review of Intermedia.

5.4.1.1 Deployment

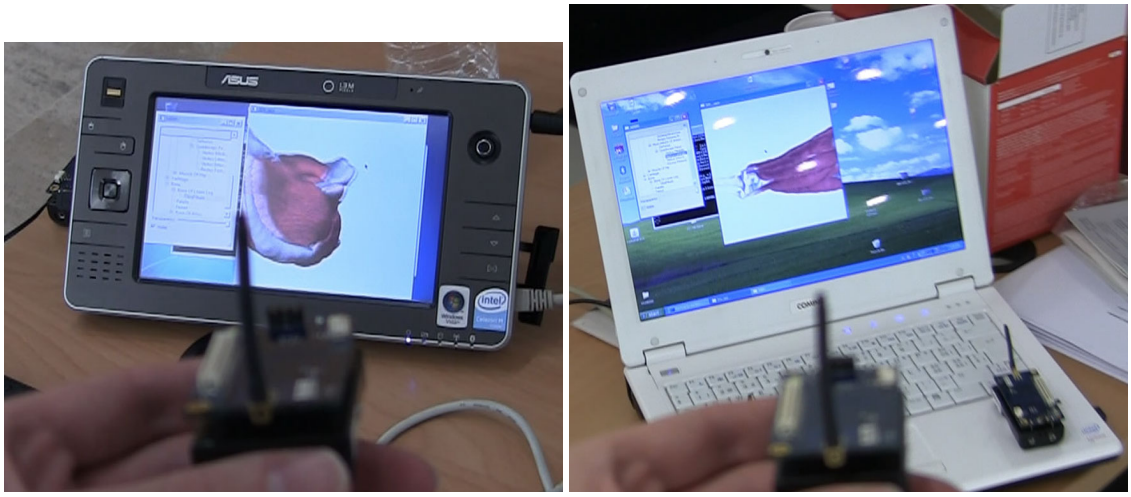
The infrastructure as was presented in section 4.5 figure 4.24 is as follows:

- A server running the Home Agent (Linux OS) for the MIP framework;
- A server running Foreign Agent (Linux OS), also used for the MIP framework;
- A server running the 3D server application (Windows OS);
- Some user clients, running the 3D client application (Windows OS), see figure 5.5a and 5.5b;
- A server monitoring the Zigbee mesh network and publish the active module using a HTTP server (Linux OS).

5.4.1.2 Results

The user carries, as shown in the figures, a Zigbee module and whenever the user gets close to a client device, the Zigbee reconfigures its network based on signal strength. By analysing the Zigbee network structure the current active client is revealed. This information is then given to the mobility framework which then tells the *new* client to change its mobile IP. The switching happens at about one meter distance and works robust. It has been tested by more then ten people, without counting the amount of times the testing person actually went forth and back between the two end-clients. The one meter distance also gives a grace time for the client to switch IP and start receiving the stream information. This takes about two to three seconds. Which after also the user interface is adapted and changed. The main results are the integration

[1]InterMedia Cafe - organized by Intracom Telecom - Athens Information Technology



(a) Moving to the UMPC. **(b)** Moving to the Laptop.

Figure 5.5: Seamless session hand-over between UMPC and laptop.

of the presented framework with the 3rd party technologies: Mobile Session Service, Context Aware Localization and Digital Rights Management

5.4.2 Data sharing among switching devices

5.4.2.1 Deployment

From the preliminary tests conducted during development and shooting a video[115] for the application presented here it was evident that interacting with 3D models on a wide scale display directly without any intermediary peripherals (e.g. mouse, keyboard, joystick, etc.) has appealed to users. They found the 2D gestures that simulate the 3D manipulations easy to apply. More dedicated user trials with medical students, and maybe doctors, are planned to be performed. This is expected to give more specialised recommendations regarding the content to be displayed and even the way of manipulating it on the table to suit the way doctors interact with anatomical material during their practices. A further work will be carried out to allow multiple users to upload content onto the table simultaneously. This could facilitate an interchange of knowledge within the formed group and the table would be sensitive to the location of those multiple users around it and thus display content in the appropriate orientation.

5.4.3 Concluding Remarks

A working test-bed was implemented and will be shown during the upcoming InterMedia event to get feedback from potential users. Such feedback will concern the responsiveness of the 3D application and the degree of satisfaction during the migration. Moreover, performance analysis is foreseen to quantify the delay during hand-overs and migrations. Through this application, we gained the opportunity of extending the use of our generic mobility framework (it has been used also in other sub-tasks, as described in the following). This has been very useful to experience with many limitations of the underlying MIP protocol, and to get important indication

Figure 5.6: Remote rendering from a collaborative interactive table to a hand-held device.

about possible extensions. Thanks to the integration of the Network Probe into this system, we had the possibility of exploiting the traffic classification engine as the trigger for data adaptation, although very simple policies have been taken into account until now. We addressed three challenging issues; preservation of real-time interactive performance of 3D contents, context-aware polymorphic presentation adaptation, and dynamic interface adaptation. To support perceptual real-time interaction with 3D contents, a micro-scale buffer based adaptation mechanism is used. It dynamically adjusts the frame rate of client device and the compression quality according to the current network situation and rendering capability.

The novelty of the presented research here lays first in the usage of NFC for establishing a connection between a mobile phone and a multi-touch table and secondly in the mapping of 2D finger gestures for interactions to 3D manipulations. This is used for integrating mobile phones and tables which can open new doors for collaborative interactions that can engage multiple users at a time in various domains such as education, entertainment and even commerce. Adding 3D graphics enhances their experience even further which helps in creating an immersive group learning experience.

5.5 Service Distribution and Render Context Switching

The experiment was conducted in three different set-ups, the first set-up utilized the Microsoft Kinect for interaction and shows a "virtual mirror" for a single user. Any other user connected to the same instance of the VTO can see the animations performed by the Kinect user. The Kinect is

directly hooked up with the server and avoids going directly through the client. Testing showed that the Kinect offers a stable 30 captures per second, which are applied to the avatar. Since the garment simulation runs most of the time slower than the amount of captures, the animation sometimes appears a bit sluggish and is therefore limited by how fast the simulation is. The rendering itself for a single user is above 200 frames per second, as the scene to be rendered is fairly light in terms of polygons, textures and deformable models.

The hardware used for the server is an Intel i7 core 950 (quad core 3 GHz), with a NVidia GTX 465 graphics card (hardware (iv). The clients range from a similar desktop system to a tablet (Asus Eee Slate EP121) with an Intel core i5 (hardware (viii) and a Windows phone 7 (hardware (xii)) which has an ARM-based single core 1 GHz CPU. These are shown in figure 5.7, connected simultaneously to the same server.

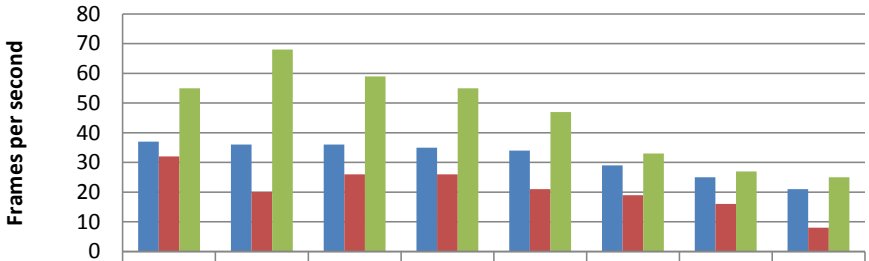
Figure 5.7: Three clients: Left tablet, middle desktop system with touch-screen and right below a smartphone.

The phone client runs significantly slower then the, obviously, more powerful tablet and desktop. This is however also because of the different kind of rendering, since the phone is limited to using MJPEG. Where, for example, VP8 sends incremental information about changes in the image, MJPEG is a full image. This is shown in the amount of data used by the network. For VP8, if it is not sending a key frame (full-frame), it ranges from a few bytes to a couple kilobytes, whereas the JPEG image has a size of 30 kilobytes per frame (800x480 resolution) at mediocre quality settings. The frame-rate on the phone varies between 4 and 8 frames per second.

5.5.1 Results

In order to test the performance we used the following set-up: One server and eight identical client machines. The server, with the same hardware as described before, creates a rendering context for the eight users with a resolution of 1280 by 800 for each(10240 by 800 pixels). This is the native resolution for the tablet. For every 5 seconds the amount of frames rendered by the server and by the active clients are logged for a fixed duration. Initially starting with one client connected, and after a one minute runtime a following client is activated. The results are presented in figure 5.9 for the performance on the clients and 5.10 for the server.

Figure 5.8: Close-up on the phone client. The client is using the Wi-Fi capabilities of the phone.



	1	2	3	4	5	6	7	8
Average	37	36	36	35	34	29	25	21
Min	32	20	26	26	21	19	16	8
Max	55	68	59	55	47	33	27	25

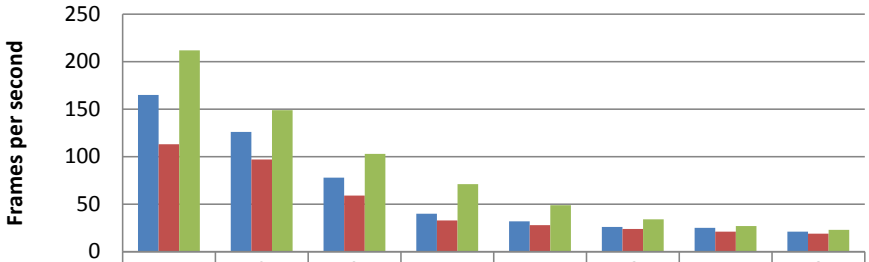
Figure 5.9: Client performance with increasingly more clients connected to the server.

Because of the throttling mechanism and the micro-buffer, the clients never run at full speed, unlike the server. This keeps the minimum and maximum frame-rate closer together and a more stable average frame-rate is achieved. At about six clients the server is unable to uphold a faster rendering speed then the clients and starts limiting the frame-rate.

With every additional client, the server not only renders another frame, but also the additional frame grabber and encoder for every active client are making the server drop rapidly in speed. Yet still with eight clients a possible interactive rate is maintained.

5.5.2 Concluding remarks

Because of the limitation of the Windows phone 7, it shows the utilizing of a different encoding mechanism depending on the device profile. Possibly on a Windows Phone 8, iPhone or Android based high-end smart-phones the VP8 codec can be deployed and should show better results. This is even more interesting for video streaming codecs with hardware acceleration support, which is dependent on the hardware-capabilities, limitations of the platform e.g. restrictions



	1	2	3	4	5	6	7	8
Average	165	126	78	40	32	26	25	21
Min	113	97	59	33	28	24	21	19
Max	212	149	103	71	49	34	27	23

Figure 5.10: Server performance with increasingly more clients connected to the server.

from the operating system and possibly support by the drivers.

The difficulty lies in the rendering on a single server where a true separation of the simulation and the rendering is not possible. Furthermore to have the interface embedded into the 3D scene simplifies the interaction mechanics on the clients side. Where in the previous application scenario a client side rendering of a device specific interface was presented on the basis of UIML, here it completely omits the client side implementation and only limits itself to the handling of user interaction or device input events. It does however burden the server side, and needs to render within the 3D context the interface. This proved quite tedious during implementation and several reimplementation were required. But this can be overcome when each user is connected to its own rendering server.

5.6 The Collaborative MRI Segmentation

5.6.1 Deployment

For collaborative segmentation, we have selected a volumetric MRI data of the lower limbs, which provides an interesting multi-user case scenario, where the segmentation is challenging due to the poor image quality and inhomogeneous intensities within its structure (imposed by hardware and protocol restrictions) [76] and thus can benefit from multiple users to concurrently aid in the segmentation process. Moreover, this data offer many different types of anatomical structures that needs to be segmented, e.g., bones and various types of muscles, thus benefiting from the collaborative segmentation of multiple structures concurrently.

In order to illustrate the concepts of the proposed collaborative telemedicine system, two scenarios were designed and evaluated.

- Teacher-students scenario A teacher shows to students the reading of a medical images and the concept of semi-automatic segmentation for Region of Interest (ROI) annotation on these images. In this scenario, both the teacher and the students have access to their own computers which have installed our proposed collaborative editing medical viewer.

A group session is started and all the students observe the segmentation evolution as the teacher initiates the segmentation and changes the viewing slice across the dataset. Then the teacher shows how to manually and locally correct the segmentation evolution by inserting various constraint points. Once this simple principle is explained, students are invited to interact with the segmentation process. The teachers are assigned greater 'weight' to the constraint points than the students, thus, enabling the teacher to override students' constraints.

- Expert-expert scenario Two (or more) experts are segmenting the same dataset. Each of them can monitor and modify the segmentation in different parts of the dataset hence expediting the segmentation process. As with the "teacher-students" scenario, when experts work in the same slice, weights may be allocated based on the experience of the users, thus giving more priority on the segmentation constraints to the more experienced user.

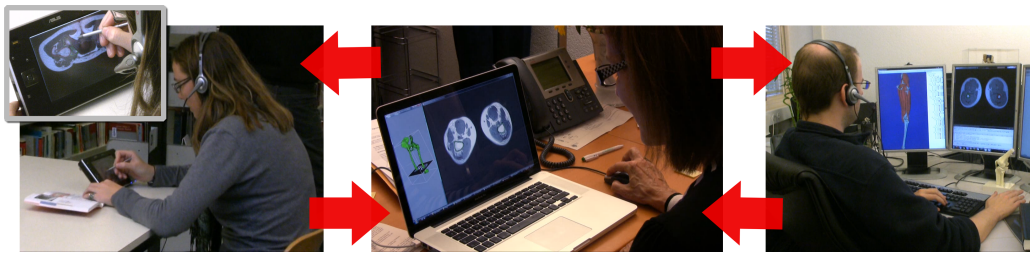


Figure 5.11: A collaborative segmentation scenario. A teacher (center) uses her laptop to monitor and guide the two students in a collaborative segmentation session, where one student is using a portable device (left) and the other using a conventional workstation (right).

5.6.2 Performance Analysis

We evaluated the interactive performance of our proposed collaborative segmentation mechanism with two example scenarios as in previous section, respectively of teacher-student and expert-expert scenarios. In the first example, the expert-expert scenario was considered i.e., two users with their own computer was requested to concurrently segment 4 different bone models on an volumetric MRI (dimensions of $483 * 358 * 270$ pixels). In the second example, teacher-students scenario was simulated, where 20 users concurrently joined in a collaborative session where one user operated on an image volume ($260 * 511 * 242$) consisting of 21 models of the muscles while the other users observed. We configured subscribers and a publisher with a desktop PC (hardware (i)) connected to LAN. For each experiment, we measured the fps on the publisher and the subscriber sides. We averaged the fps over 100 segmentation processes. In the first simulation we had 4.6 fps on the publisher, the averaging of the subscribers resulted in 4.2 fps. On the second example, 3.8 fps on the publisher and 3.2 fps on the subscribers (averaged). Simulation results show that our system does not expose any significant degradation in system responsiveness and preserves interactive performance of users in varying conditions and was satisfactory for collaborative editing (4 fps). Performance was mostly bound by the segmentation iteration on the publisher. A subscriber used less than 1 MiB and less than 1% of CPU consumption. The complexity of the segmentation, in terms of memory consumption

and execution speed was mostly related to the size of the segmented image and the number of models that were simultaneously segmented. These factors were expected to also have an impact on the system as the image data and the models contours (overlaid on the slice at the subscriber side) need to be sent over the network.

5.6.3 User Case Study

In order to measure the efficiency of our collaborative system in terms of usability, we ran a controlled experiment over 10 participants (5 males and 5 females, with ages between 20 to 36 years old). The participants consisted of 2 experts and 8 students, where an expert was considered to be a user experienced with knowledge in medical image analysis.

5.6.3.1 Experiment Design

Based on the two scenarios, repeated measures experiments were conducted. Only one independent user feedback variable, the type of collaborative editing algorithm, was manipulated. Two dependent variables were measured as follows.

- Task Completion Time: time taken to complete the segmentation on a single image.
- Segmentation Error: errors between segmented and the reference segmentation.

We also measured user satisfaction using questionnaire as follows.

- User Satisfaction: which type of algorithm fulfilled the best requirements for the specific scenario? We devised a questionnaire based on a well-known IBM Post-Study questionnaire [116]. 10 questions corresponding to the system usefulness was designed where each question used 5 point graphic scales, anchored at the end points with the terms 'Strongly agree' for 1 and 'Strongly disagree' for 5. Some space was left at the end of the questionnaire for comments.

5.6.3.2 Tasks and Procedures

Participants were asked to learn our system with a simple introductory manual which exemplifies the usage of the system. All participants had no prior exposure to the application. After participants were given only a basic introduction and a few minutes to play with the application, they were further instructed on the use of the application by participating in the 'student-expert' scenario. Participants were given two tasks: each participant had to collaborate with another user in a segmentation task on an image using firstly the strict locking algorithm and then our proposed collaborative mechanism. The segmentation consisted in segmenting the 4 bones on a MRI (single slice image) as accurately as possible within the shortest time possible. Bone models were initialized sufficiently close to the structures to be segmented so that the users could drive the meshes towards the structures through applying the three types of CPs (as described in section 3.9.4). The ground truth segmentation was a priori computed by experts for use as a benchmark in calculating the segmentation error. For a given mesh contour resulting from a user

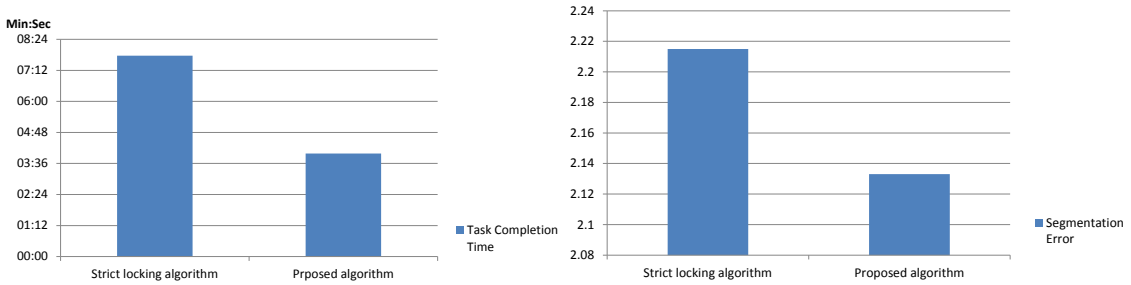
segmentation, an error $e1$ was computed as the Euclidean distance between each point of the contour and its closer point on the reference contour. The same was done by inverting roles of user and reference contours to get an error $e2$. The final segmentation error for a mesh was thus the sum $e1 + e2$. By averaging the errors for all bones, we got the so-called average symmetric distance error in mm. Pairs of participants were randomly chosen. After completing the tasks, participants filled in a questionnaire.

5.6.4 Results

We calculated discrete statistics for every dependent variable and user satisfaction.

- Task Completion Time

  Figure 5.12a shows the average time taken by non-expert participants to complete their task. Using the proposed system, participants finished their task earlier compared to the strict locking based system because participants could update their inputs without waiting the other user to release the lock. Another reason is that each participant could work on different model contours at the same time. In some cases, participant did not release the lock preventing the other participant to improve the segmentation even though she/he had a better idea to complete the task.



(a) Task completion time. **(b)** Segmentation error.

Figure 5.12: Completion time and segmentation error results.

- Segmentation Error The obtained results revealed that participants' segmentation error using strict locking was 4 percent greater on average than using the proposed system, as shown in figure 5.12b. From our observation, this was attributed to the fact that the participants were able to correct each other's errors, where the agreements from both the participants were more accurate than from an individual user.

- User Satisfaction From the questionnaire, a qualitative user satisfaction was derived. Most people were strongly or moderately satisfied with the collaborative algorithm as shown in figure 5.13. No participant strongly disapproved the use of both the proposed algorithm and the strict locking algorithm. Similarly, participants reported that they could effectively complete their tasks both with the proposed algorithm and strict locking algorithm. However, participants felt more comfortable to collaborate with others with the proposed algorithm because more freedom was given and it was not necessary to wait for the lock to

be released.

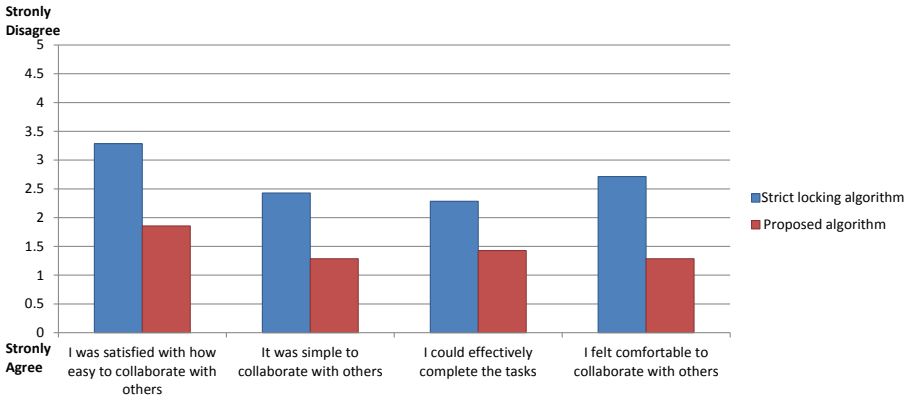


Figure 5.13: Average of the selected questions in questionnaire.

Participants were asked for comments or suggestions in the questionnaire. Two participants reported that strict locking algorithm needed a floor control mechanism requesting a lock or at least other communication facilities, such as chat, to negotiate their turn because they felt uncomfortable to wait until lock is released. Two participants reported that they felt a little-bit uncomfortable to interact with others because the simulation responsiveness was relatively low in both cases because their inputs (constraint points) progressively affected the segmentation instead of having an instantaneous effect. As mentioned in section 5.6.2, a publisher run in a more powerful workstation or the use of segmentation optimizations would yield a faster simulation. However, as explained in section 3.9.3, this progressive change of the segmentation evolution is essential to be able to detect and correct errors sufficiently early. Finally, in the context of our specific physically-based segmentation, creating large "brutal" changes may create instabilities in the simulation and thus ultimately affect the quality of the segmentation.

5.6.5 Concluding Remarks

In this experiment we tested a collaborative telemedicine system for real-time and interactive segmentation of volumetric medical images and demonstrated its usages using two typical case scenarios: teacher-students and expert-expert collaboration. User evaluation was conducted which measured the enhancements with our approach in comparison to the conventional strict locking collaborative system that is often found in medical systems. Our system performance results indicated that, prior to code optimization, it can support large number of users for online collaborative editing of 3D volumetric medical images. View generation speed was heavily dependent on the computational resources (CPU, memory size, video card capability) of the producer side rather than the subscriber side or network condition because subscribers used less than *one* percent of CPU consumption. Our user study results revealed that our proposed system can be useful in teleradiology context and has many potential clinical applications. As our system is designed to be modular, the system can be integrated to other, more complete, medical image viewers, such as the popular open source image viewers built using Insight Segmentation and

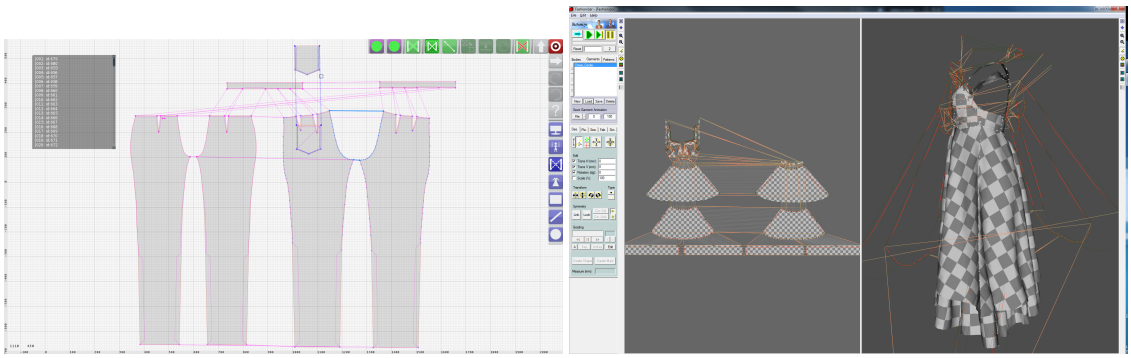
Registration Toolkit (ITK) [2]. Our system is also not restricted to the use of the deformable model segmentation algorithm as presented in this study. This algorithm was chosen for its inherent characteristics to evolve from rough to optimal segmentation results via an iterative and intuitive visual feedback. The advantage of visual feedback is in the ability for the users to understand the segmentation process as it iterates to the final result. We conducted some preliminary tests on a mobile device (an UMPC) over a wireless network, and similar performance was observed. Future work will mostly focus on dynamic polymorphic presentation (view and interface) adaptation and context-aware network adaptation according to the current device capability in order to support nomadic users to collaborate with each other exploiting diverse devices.

5.7 Collaborative Services with shared Data Models

We look at two aspects of the implementation which are the *collaborative pattern designer* and the *GPU based cloth simulation*.

5.7.1 Collaborative pattern designer

The pattern designer was deployed on several devices, a desktop pc (hardware (v)) and several tablets (hardware (viii) & (xv)) and was evaluated by a professional. In comparison the professional software for designing garments and the implementation are shown in figure 5.14



(a) Collaborative Pattern Designer. **(b)** Fashionizer.

Figure 5.14: Designing a pattern.

Based on the professional's remarks the following can be concluded. Mostly about the interface and workflow which is quite different from Fashionizer. Although it should be remarked that Fashionizer is a "not easy to use" application. But she was interested in the collaborative features and would certainly use it as a discussion tool for discussing it with colleagues (figure 5.15). The ability to both be able to edit is certainly a plus, but lack of functionality makes it not practical. This is already evident from the screenshot 5.14, where a fully simulated dress is shown as well.

[2]Insight Segmentation and Registration Toolkit web-link: `http://www.itk.org/`

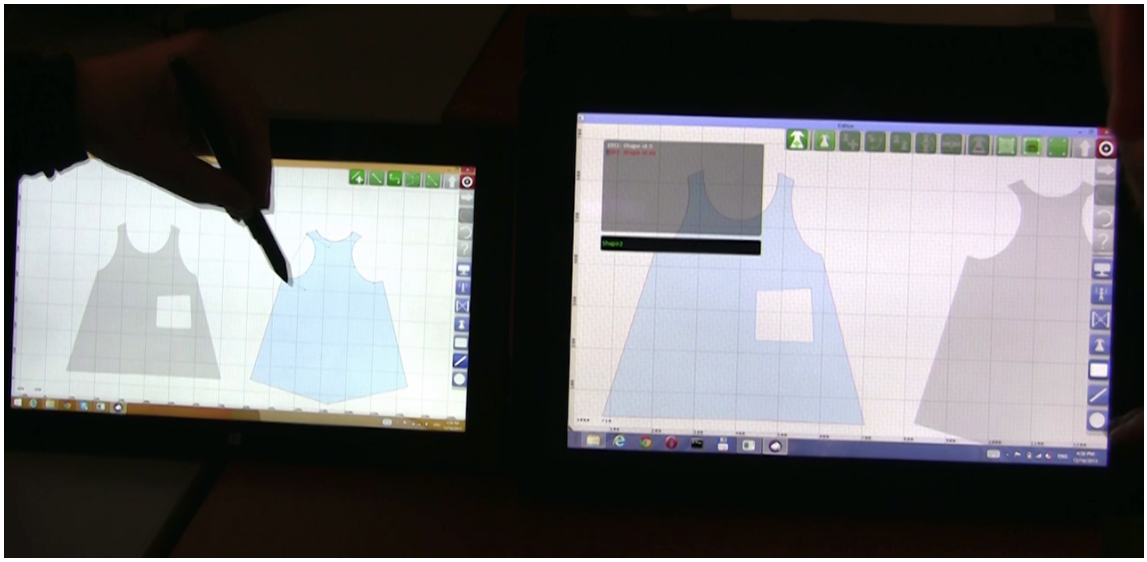


Figure 5.15: Collaborative pattern designer used by two users.

5.7.2 GPU based cloth simulation

The server running the cloth simulation was deployed on a desktop system (harwdare (v)). The results for the simulation are shown in table 4.4. The deformed mesh is send over LAN to end clients (hardware (iv)). The simulation update rate can be set and can update at maximum of the simulation cycle speed. As for the client there is a minimal impact on the performance, as the data can easily be handled by the network and streaming it directly to the rendering is done through updating *VertexBuffers*. The client runs depending on the size of the mesh at 200+ fps and renders independently from the updates coming from the server.

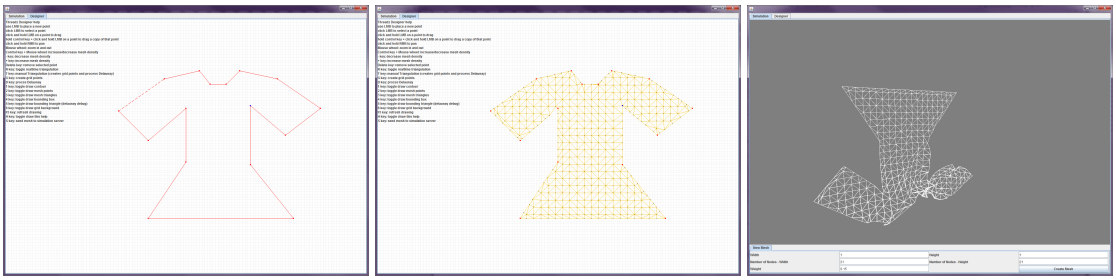


Figure 5.16: Using the Threadz simulation through a Java client.

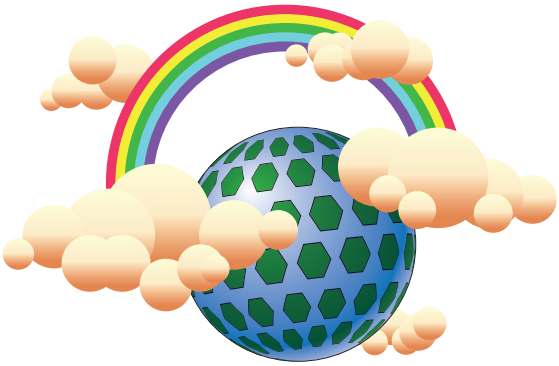
The Java client which is used for rendering was used. A simplified version of the pattern designer was used to have a pattern send to the server, which is then directly simulated. Interaction with the pattern is possible by mouse interaction. The user can select a mesh and upon dragging the mesh is moved, by moving the *fixed* vertices. This gives the illusion of moving cloth and due to stretching the rest of the cloth pulled after the *fixed* vertices. This was tested with multiple users and no performance drop was detected. This is because the interaction on the mesh is treated exactly the same as a deformation and therefore has no impact.

5.7.3 Concluding Remarks

Albeit the applications usage was found to be minimal due to lack of completion, it did however display the possibilities which were enabled by the presented framework. From a developers points of view the publish subscribe data sharing allows it to implement these kind of collaboration on complex data structures that require real-time updates and propagation relatively quick. But it would require a more full fledged application to truly test if it is applicable in professional tooling. So far it has shown promise in fashion design as well as MRI Segmentation.

Chapter 6

Conclusion



6.1 Motivation

In previous chapters we have described a framework architecture that enables 3D content to be viewed and interacted with on a variety of platforms, and facilitates the creation of scalable CVEs. We conducted several experiments each with a specific focus and deployments domain. As for each experiment a conclusion was given, here we summarize the outcome of our research and provide possible extensions in future work.

6.2 Achievements

We proposed a context-aware adaptive rendering architecture which visualizes 3D content and an user interface on top of a CVE, while dynamically adapting to the current context such as device availability in the current location and their capabilities. The design of the architecture was laid out into several layers in which several components reside, each with their specific function. The components communicate through a public/subscribe methodology. Albeit the main implementation was done in C++ other languages were utilized to demonstrate that the given design of the system can be easily adapted. This offers the flexibility to deploy mainly the end-client applications on a greater range of devices. By using pre-defined device profiles we can identify beforehand what the optimal system parameters are for the given service and at runtime a limited set of attributes is used for adaptation during operation. Predefined parameters include the hardware capabilities which result in a custom provided interface for the given device and rendering streaming type. At runtime the streaming and interaction rate is adapted, mostly in favour of a higher interactive rate rather than quality. Furthermore hybrid solutions are provided by either sending 3D content and 2D content independently of each other and are composed on the client side. The architecture provides a methodology for connecting existing data models (graphics, application logic or other) easily and enable a synchronization based on publish and subscribe. This offers a classic client/server setup, peer to peer or a hybrid of these two.

We have deployed this framework in several application domains, each with a specific focus. First we have the user-centric media, where we focused on the adaptive rendering and providing the visuals to a multitude of users. In this domain the individual user is the most important, where the user can choose its device and connect to a service. We have shown how to create a service around a Virtual Environment and how the end-client is capable of viewing and interacting with it. With the focus shifted towards *cloud* like services, where we show the flexibility in deploying the architecture among several servers and have a multitude of users connecting to the service, using a single point of entrance. This introduced the need for interest management and load balancing of the functionalities, as well as dynamic launching of additional services. The last domain, Telemedicine, brought collaboration and the possibility of medical applications to mobile devices. We showed how we can receive different types of data from several services and have these combined in the end application and how interactions are influencing simulation services that operate independently on the same shared dataset.

From the results it shows that an architecture as presented makes it feasible to create a very dynamic service that can be deployed for small scale services as well on a larger scale. It offers bindings to existing applications to enhance them into cloud-based services and detach them from local execution on the client side. This not only results in less dependencies on the client side, but also a better maintenance on the provider side.

6.2.1 Remote Rendering for Low-end Devices

This application scenario used the presented framework in its most basic setup and delivers an interactive view on 3D content. In its bare form it shows the potential of having streamed applications instead of relying on the client device. The biggest drawback for this technology however is the connectivity. Small transmission breaks can be accounted for, but larger gaps and interaction and visual updates are not responsive. This partially can be overcome by having a local rendering of the 3D environment, this however puts the burden on the client again. The local copy however could be a dumbed down version and interactions, aside from basic operations such as camera movement, can be buffered and send upon the re-establishment of the connection. Within a local LAN environment this technology is a very good solution for heavy 3D tasks and directly reminds of the terminal approach from the 80's, in a renewed way. As for WAN the presented methodologies for streaming 3D content becomes more and more feasible, as has been shown by commercially deployed applications (e.g. gaming cloud based services Onlive and Gaikai). These however are very specialised and are very costly [26]. For example ISP are being paid to prioritize their data streams, around the world multiple server centres are needed as otherwise the distance is too great and the lag becomes too noticeable. The biggest benefits from an approach like this is the rapid development and deployment for multi-platform systems. since a single service has to be build and maintained instead of *porting* it to all kinds of system and operating system (OS) flavours. On the client side there still needs to be a receiving application, but its complexity is far less than the actual service and therefore development time and maintenance of these clients are greatly reduced. The presented architecture provides methods for stabilizing the streaming and offers functionality for switching between different compression methods.

6.2.2 Remote Rendering with Augmented Reality

The addition of AR still gives a lot of hurdles when it comes to combine it with high quality simulations. The decision to have the rendering locally on the device or server side have both their advantages and disadvantages. Where locally the rendering is more responsive but less detailed than rendered remotely due to limited capabilities. The simulation as a condition for the scenario had to be run on a server and the camera image from the mobile device. Even on newer hardware such as the Nokia Lumia 820 [1] which has a dual core processor, the capturing and streaming of the camera images itself take considerable calculation power, let alone the processing of each frame for retrieving tracking information. In the last few years the rendering

[1] Nokia Lumia (xiii)

capabilities have increased rapidly, from which we can profit by using better blending functions. Yet on the other-hand the same can be said about the networking capabilities, which make it more reasonable to have the camera image send to the server and wait for a fully composed image in return. While today there are plenty of applications demonstrating both static and dynamic content in AR scenarios, their use among the general public is still fairly limited, let alone for mobile AR which is more limited to the use of platonic meta data (e.g. text and 2d images). However as technology progresses the application scenario as presented will become more feasible on a larger scale and new domains of usage will be explored.

6.2.3 Adaptive Rendering with Dynamic Device Switching

In user-centric pervasive computing environments where users can utilize diverse devices nearby any-time and anywhere, anchored subscription of 3D contents is essential because it is impractical not only to manually copy 3D contents from one device to another whenever a user moves but also to render complex 3D data locally on the resource-limited devices, such as cell phone and PDA. Three challenging issues are addressed; preservation of real-time interactive performance of 3D contents, context-aware polymorphic presentation adaptation, and dynamic interface adaptation for a context-aware adaptive rendering system which facilitates a mobile user to seamlessly manipulate 3D contents utilizing heterogeneous devices nearby. To increase interactive performance of 3D contents, a context adaptation mechanism which dynamically adjusts frame rate on a client device while and a hybrid rendering mechanism which partially caches a subset of 3D model depending on client capabilities are used. To overcome heterogeneity of device capabilities such as display size and input controls, a user interface adaptation mechanism, which dynamically customizes functionality of system and user interfaces based on current device capability, is also introduced in our system. This application scenario is the result of a cooperation of partners in different fields, researching on interdisciplinary aspects of user-centric multimedia, mobile and wearable interaction, multimedia content annotation and adaptation, and networking and DRM. Where our main contribution resulted in an user-centric experience for uninterrupted streaming of 3D content while switching between devices. Albeit quite complex as the use of several services and certain infrastructure is required to have the optimal experience, the scenario shows the potential of deploying applications in such environments where a user can simple change between devices without interrupting continuity, move to a different location and change from device again. The ubiquity of the whole system behind is the key part in achieving this user-centric experience. Deployment of our architecture was somewhat limited as the depending components where restricting certain aspects, such as the limitation to only use UDP. Therefore tracking of the user is more difficult since UDP has no guaranteed data delivery.

6.2.4 Service Distribution and Render Context Switching

As the previous and first application scenario overcomes the limitations for mobile devices and enables to have 3D media in a user centric environment, this application scenario looks to the other side of providing the service and how to build and deploy a cloud-like service environment.

As presented the architecture provides functionality that enables to have services being *spawned* and due to it modularized offer of functionality the services can be deployed across multiple servers and be managed by a master service. This is still very application specific, but as the architecture provides bare functionality, rapid development of these kind of services is made possible.

6.2.5 The Collaborative MRI Segmentation

In the previous application scenarios the term collaboration has been used, and made possible to some extend, however more elaboration on the collaboration aspect was conducted in this application scenario. Furthermore the context had been changed from a user-centric media and E-commerce domain to a Telemedicine domain. A domain which we interpreted as more serious as the intent is less commercial and a more "worthy cause". There is a lot being done in Telemedicine, however it is still in its infant state when it comes to collaboration and having the possibility of mobility, which opens up a whole new area of applications. A possible application was presented here (section 3.9) and studied in the context of collaboration. We show how different kinds of collaboration impact the usefulness, where a free independent interaction scheme clearly is preferred over turn based, while still retaining the capability to interact with modifications applied by other user. We show how the same application can be used on a mobile device with a significant smaller screen, yet still provide the same functionality and collaboration. The segmentation of MRI is still mostly done by a single professional user, however with the presented framework it is possible to provide a robust multi-user solution. Aside from the manual segmentation done by the user, the simulation that attempts at automatic segmentation is another important part, as it shows that besides real users also artificial generated interaction can be applied in a similar fashion without modifying any end-client application or collaborative aspects.

In continuation of the scenario, the application has been rebuild with the publish subscribe data sharing in mind. This currently shows a great improvement over the old implementation and is part of future research.

6.2.6 Collaborative Services with shared Data Models

To employ a full pipeline from designing a pattern to the end-client where the user can interact in real-time with the simulation proofed to be more cumbersome than anticipated. Although the framework does provide all the means to make it happen it does not ease the process of implementing *user* applications. Which need solid requirements on the expected tooling, interface and data formats. It was an attempt at showing the framework within a greater deployment but hampered at the amount of implementation needed. In retrospect the application scenario should have more focused on the multi-user and simulation interaction. Since the MRI segmentation proofed to be exemplary in showing the collaborative aspect. In its defence however, this scenario was constructed before the renewed implementation of the MRI segmentation and was implemented during establishing the publish subscribe data sharing methodology. During

which it both evolved, from concept to prototyping forth and back.

6.3 Limitations and Future research

The use-case scenarios presented here were conducted in localized areas and tested with up to 30 end-clients. Albeit the presented architecture took into consideration the ability to scale to a multi-fold of active end-clients it was never applied. Mainly due to the issues already present at the small scale deployment deemed to be more challenging than anticipated. It is therefore for future research on how to extend this to a greater area, while retaining all the same capabilities. In some sense the issue can be solved by increasing the amount of server hardware to accommodate the increase of end-clients. This however is a poor solution as it only solves the rendering and simulation demands, but the handling of user events and interaction within the VE will increase significantly.

Improvements can be achieved in having a more dedicated streaming processing software. As currently direct frames are grabbed from a buffer on the graphics card, depending on the rendering engine used this can be done in several ways. The fastest *image grab* methodology was to render to a texture buffer and have this then copied to a local buffer. Possibly faster methods can include dedicated hardware capable of capturing and direct compressing into a specific format. Albeit this was not the focus of the framework, utilizing the jpeg and VP8 codecs mainly in software mode did limit the frame-rate. Later updates to the VP8 codec offered some hardware acceleration, but there are faster compression codecs available (mostly commercial).

The adaptive user interface integration proofed to be difficult and is currently the biggest limitation. Using several methods throughout the application scenarios, the embedded interface into the 3D scene shows the easiest solution. However this requires that the interface is displayed within the rendering context (OpenGL for example). This is no problem when using Qt and building an application from ground up. But the dependency on using a library that is capable of rendering the interface in the 3D scene can be interpreted as too strict. Other solutions might be found in the use of HTML 5 and the use of JavaScript and web-sockets for custom implementations and functionality. Albeit these can be very power consuming, where a native interface performance-wise is insignificant.

In continuation of using nodes and the possibility of running them in their own respective process came from the idea of upgrading services remotely and without interruption. By having a node saving its state to disk and spawn the new version of the same node which then takes the old state. Converting it to its own state and takeover from the old node which then can be shut down. The initial implementation shows that this is quite possible but it becomes rather quickly very complex especially with platform independence in mind and having active networking connections. However we feel this is a viable direction.

Appendix A

Acronyms

3G 3rd generation mobile telecommunications

4G 4th generation mobile telecommunications

ALM Application Layer Multicast

ALMP Application Layer Multicast Protocol

AoIM Area of Interest Management

API Application Program Interface

AR Augmented Reality

CAAR Context Aware Adaptive 3D Rendering

CAVE Cave Automatic Virtual Environment

COLLADA COLLAborative Design Activity

CPU Central Processing Unit

CP Constraint Point

DRM digital rights management

CSCW Computer-Supported Cooperative Work

CVE Collaborative Virtual Environment

DHCP Dynamic Host Configuration Protocol

DLL dynamic link library

DOI Domain of Interest

DUI Distributed User Interface

DVE Distributed Virtual Environment

DVMRP Distance Vector Multicast Routing Protocol

EM Event Manager

fps frames per second

FTP file transfer protocol

GHz gigahertz

GiB gibi bytes

GPGPU general-purpose computing on graphics processing unit

GPU graphics processing unit

GUI graphical user interface

HCI human computer interaction

HDR high dynamic range

HID human interface device

HMD head mounted display

HTTP HyperText Transfer Protocol

IE Interest Expression

IM Interest Manager

IP Internet protocol

IPv4 Internet protocol version 4

IPv6 Internet protocol version 6

ISP Internet Service Provider

ITK Insight Segmentation and Registration Toolkit

JNI Java Native Interface

JPEG Joint Photographic Experts Group

KiB kibi bytes (1024 bytes)

LAN Local Area Network

LOD Level of Detail

LSVE Large Scale virtual environment

MAC media access control

MBone Multicast Backbone

MHz megahertz

MiB mebi bytes (1024 kibi bytes)

MIP Mobile IP

MMOG Massive Multi-player Online Game

MMO Massive Multi-player Online

MMVE Massively multi-user virtual environment

MOSPF Multicast Extensions to OSPF (Open Shortest Path First)

MRI Magnetic Resonance Imaging

MUD Multi User Dungeon

MVC Model View Controller

NCVE Networked Collaborative Virtual Environment

NFC near field communication

NPC Non-Playable Character

NUI natural user interface

OS operating system

OSG OpenSceneGraph

OSI Open Systems Interconnection

PA Personal Address

PC Personal Computer

PDA Personal Digital Assistant

PIM-DM Protocol Independent Multicast - Dense Mode

PIM-SM Protocol Independent Multicast - Sparse Mode

PNG Portable Network Graphics

PSSAM Presentation Semantics Split Application Model

RAM Random Access Memory

ROI Region of Interest

RTP Real-time Transport Protocol

RTT Round Trip Time

RUDP Reliable User Datagram Protocol

SAIL Sensor Abstraction and Integration Layer

SDK software development kit

SDL Simple DirectMedia Library

SERVIVE Service oriented intelligent value adding network for clothing-SMEs embarking in Mass-Customisation

SMS Shared Memory Space

SFX special effects

SO Shared Object

SSL Secure Socket Layer

TCP Transmission Control Protocol

UDP User Datagram Protocol

UIML User Interface Mark-up Language

UML Unified Modelling Language

UMPC Ultra Mobile Personal Computer

UPnP Universal Plug and Play

VBO vertex buffer object

VE Virtual Environment

VHD Virtual Human Director

VOD video on demand

VOIP voice over IP

VPN Virtual Private Network

VRML Virtual Reality Modelling Language

VTO Virtual Try-On

VW Virtual World

WAN Wide Area Network

Wi-Fi Wireless Fidelity

WLAN Wireless Local Area Network

XML eXtensible Mark-up Language

XSL eXtensible Style-sheet Language

Appendix B

Frameworks

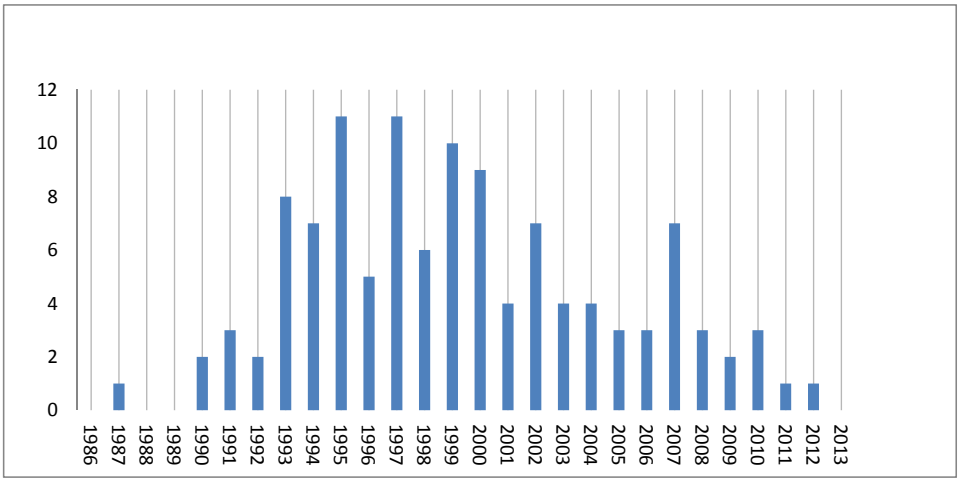


Figure B.1: Chronological overview of publications on CVEs (Appendix B)

Table B.1: CVE Architectures

Name	Year	Description	References
A3	2008	Interest Management Algorithm for Distributed Simulations of MMOGs	[45]
Advanced Distribution Simulation	1995	ADS	
ALSP	1997	"Aggregate Level Simulation Protocol describes message filtering Two classes of data: - Attributes of persistent objects Entities in VE - Interactions Non persistent events Class filter, filter on class (regardless of value) Attribute value filter (e.g. range checks) Filtering at sender and receiver Uses broadcast"	[117]
ANTS (MOVE)	2002	ANTS/MOVE Multiuser Oriented Virtual Environments	[118]
ARIVU	2010	Power-aware middleware for multiplayer mobile games	[119]
ATLAS	2002	A Scalable Network Framework for Distributed Virtual Environments	[34]
Aviary	1994	distributed obiect system.	[120]
Avocado	1999	A Distributed Virtual Reality Framework	[121], [122]
Bamboo	1998	Virtual Environment Toolkit, a portable system for dynamically extensible, real-time, networked, virtual environments	[123], [124]
Beach	2004	application model and software framework for synchronous collaboration in ubiquitous computing environments.	[125]

Continued on next page

Table B.1 – *Continued from previous page*

Name	Year	Description	References
Bricknet	1994	Based on an object sharing strategy allowing users to set-up their own private workspaces, populated by shared and private objects. Virtual worlds are not restricted to sharing an identical set of objects; a virtual world manages its own set of objects, some or all of which may be shared with the other virtual worlds on the network.	[126]
Calat	1996	Intelligent tutoring system. server side and a multimedia viewer on the client side.	[127]
Calvin	1996	Prototype of Cavern. multiple users to synchronously and asynchronously experiment with architectural room layout designs in the Cave Automatic Virtual Environment (CAVE).	[128]
Cavern	1997	A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments	[129]
CavernSoft-G2	2000	(CavernSoft G1 is Cavern)Toolkit for High Performance Tele-Immersive Collaboration.	[130]
CCTT	1995	"Close combat tactical trainer DIS compliant Interest Management – pre-defined grid and packet type based. Uses multicast per grid-cell"	[131]
Colyseus	2006	a distributed architecture for online multiplayer games	[132]
Community Place	1997	Formerly known as Cyber Passage. Community Place is a shared multi-user VRML system de- signed to work in the Internet. It consists of a VRML2.0 browser, a multi-user server architecture and an application support environment.	[133]
Continuum	2007	An architecture for user evolvable collaborative virtual environments	[134]
CoUniverse	2009	Framework for Building Self-Organizing Collaborative Environments Using Extreme-Bandwidth Media Applications	[135], [136]
Cove	2006	Collaborative object-oriented visualization environment	[137]
Coven	2001	Based on DIVE; prototype large-scale applications of CVE.	[138]
CSPray	1997	Provides different levels of information sharing, incremental updates to reduce network traffic, an intuitive floor control strategy for coordinating access to shared resources, a built in session manager to handle participants who either join late or leave early and a host of collaborative 3D visualization aids.	[139]
CVD	1999	Together with CAVE5D and tightly integrated with Cavern. Cave5D/Virtual Director, integrates the capabilities of both existing VR applications, Cave5D and Virtual Director, in order to enable the user to view and interact with the data from within the data set, visualize the data in real time and easily record the experience.	[140]
CyberWalk	2003	a web-based distributed virtual walkthrough environment.	[141]
DACIA	2000	A mobile component framework for adaptive groupware application.	[142]
DEVA3	2000	Architecture for a Large-Scale Distributed Virtual Reality System.	[31]
DEVRL	1996	Distributed extensible virtual reality laboratory	[143]
DIS	1993	Distributed Interactive Simulation, the follow up of SIMNET and focuses on military simulations.	
DISCIPLE	1999	a framework for multimodal collaboration in heterogeneous environments.	[144]
DIVE	1993	"The Distributed Interactive Virtual Environment (DIVE) is an internet-based multi-user VR system where participants navigate in 3D space and see, meet and interact with other users and applications. The DIVE software is a research prototype covered by licenses. Binaries for non-commercial use, however, are freely available for a number of platforms. The first DIVE version appeared in 1991. DIVE supports the development of virtual environments, user interfaces and applications based on shared 3D synthetic environments. DIVE is especially tuned to multi-user applications, where several networked participants interact over a network."	[145]
Dive 3	1999	Latest (and last) version DIVE 3.3x (eXperimental)	[24]
DOOVIE	1997	an architecture for networked virtual environment systems	[146]
Drone	2009	A Flexible Framework for Distributed Rendering and Display	[147]
DSG	2010	"Distributed Scene Graph to Enable Thousands of Interacting Users in a Virtual Environment"	[148]
DVS	1991		
EQUIP	2002	a software platform for distributed interactive systems	[149]
ERCIS	1998	Distributed interactive simulation for group-distance exercises on the web.	[150]
EventHeap	2000		
FlowVR	2004	A framework for distributed virtual reality applications	[151], [152]
FUSE	2002	ubiquitous collaboration within diverse mobile environments	[153]
GreenSpace	1995	a distributed virtual environment for global applications.	[154]
HLA	1995	"Higher Level Architecture; general purpose architecture for distributed computer simulation systems. Two types of filtering Class based and Value based"	[155]
HyClass	1997	Interactive Cooperative Learning System Based on Virtual Shared Space	[156]
Hydra	2007	A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer	[157]
ICOME	1999		
iRos	2002	A dedicated meeting space with large displays, wireless or multimodal devices, and seamless mobile appliance integration.	[158]

Continued on next page

Table B.1 – *Continued from previous page*

Name	Year	Description	References
JPSD	1995	"Joint Precision Strike Demo Prototype simulator Uses DIS protocol A node simulates entities Per node Interest management Entity broadcasts(publish) their interest manager to other nodes Static: priori simulation Dynamic: upon need Uses point-to-point unicast"	
LEARS	2011	lockless, relaxed atomicity state parallel game server	[159]
MACVE	2007	A Mobile Agent Based Framework for Large-Scale Collaborative Virtual Environments	[160]
MaDViWorld	2002	A software framework for massively distributed virtual worlds	[161]
MASSIVE 1	1994	Initially aimed at teleconferencing, supports multiple users, applications and worlds connected by "portals"	[22]
MASSIVE 2	1996		* [162]
MASSIVE 3	2000	flexible support for data consistency and world structuring	[162]
Matrix	2005	Adaptive Middleware for Distributed Multiplayer Games	[163]
MAVERIK	1999	A micro-kernel for large-scale virtual environments	[164]
MERCURY	2002	a scalable publish-subscribe system for internet games	[165]
Mobihoc	2007	a middleware adopt- ing VFC to support multiplayer distributed games in ad-hoc networks.	[166]
ModSAF	1995	"Modular Semi-Automated Forces Designed for US Army Initial: LISP on BBN ButterFly Progress: SIMNET SAF -> OdinSAF Final: ModSAF Used for STOW-ED-1 and NPSNET Filter by Static Spatial based, cells And filter packet on Type (class) Uses broadcast Interest management done by aggregate level (not individual) “destination based filtering”"	
MOPAR	2005	a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games.	[167]
MPACC (gaia)	2001	Model-Presentation- Adapter-Controller-Coordinator, an application model that extends the MVC pattern to Ubiquitous Computing scenarios. This new model takes into account issues such as the non-existence of a single interaction device, contextual properties associated to the user and the space where the application runs, automatic model-view data type adaptation, mobility of the view, model and controller, and applications running on behalf of a user or a space, instead of in the context of a particular device.	[168]
MPCW-REALTIME	1992		
MUVEES	2003	a Multi-User Virtual Environment for Learning.	[169]
NATIVE	1999		
NetEffect	1997	a network architecture for large-scale multi-user virtual worlds	[170]
NOMAD	2000		
NPSNET I	1991	"Navel Postgraduate School NET DIS compliant Uses Hexagonal grid, per cell multicast address AOI, radius of grid cells. Oldest active entity in grid cell -> group leader for adding, deleting members to grid cell"	[21]
NPSNET II	1993		
NPSNET III	1993		
NPSNET IV	1993		
NPSNET V	2000	"a platform for research on infrastruc- ture technology for networked virtual environments."	[171]
Octopus	2001	A cross-platform, object oriented API designed to abstract the complex details involved with CVEs including soc ket management, shared data handling, shared object control and representations of other users (avatars).	[172]
OdinSAF	1995		
OpenPing	2004	a reflective middleware for the construction of adaptive networked game applications	[173]
P2LIFE	2010	An Infrastructure for Networked Virtual Marketplace Environments	[35]
PaRADE	1997		
PAULINGSWORLD	1998	An Immersive Environment for Collaborative Exploration of Molecular Structures and Interactions	[174]
PAVR	1999		
Proximity Detection	1994		
QUICK	2000	Framework for optimizing fidelity on a per-task basis. Integrates ratings of representational Quality, scene node Importance, and ma- chine resource Cost.	[175]
RAVE	2008	Resource-aware visualization environment	[176]
Reality Built for Two	1990	Development platform for designing and implementing real-time virtual realities .	[177]
Rendezvous	1990	an architecture for synchronous multi-user applications	[17]
Repo-3D	1998	A distributed 3D graphics library	[178]

Continued on next page

Table B.1 – *Continued from previous page*

Name	Year	Description	References
RING	1995	A client-server system for multi-user virtual environments.	[179]
Rubber Rocks	1992	Interactive simulation in a multi-person virtual world	[180]
SCAPE	2003	Stereoscopic Collaboration in Augmented and Projective Environments	[181]
SCORE	2004	a scalable communication protocol for large-scale virtual environments.	[50]
SEMMO	2008	A Scalable Engine for Massively Multiplayer Online Games	[182]
SIMNET	1987	Miltary real-time distributed combat simulations. Mainly based on broadcast networking and filtering. Introduces the concept of "dead-reckoning"	[15]
SmallTool	1998	a toolkit for realizing shared virtual environments on the Internet. SmallView was realized with this toolkit.	[183]
SmartCU3D	2001	a Collaborative Virtual Environment System with Behavior Based Interaction Management	[184]
SPLINE	1996	Used for social virtual environment "Diamon Park". Support audio and video interaction.	[23]
Steve Benford Spatial model of interaction	1993	COMIC Project; A Spatial Model of Interaction in Large Virtual Environments	[185]
STOW	1994	Synthetic Theater of War	[16]
STOW ACTD	1997	"Synthetic Theater of War Advanced Concept Technology Demonstration Uses experimental HLA, and grid based, variable pre-defined grid cell sizes. Each cell with multicast"	
STOW ED-1	1995	"Synthetiic Theater of War Demo 1 Uses DIS PDU (but not fully DIS compliant) Uses multiple grid layers (fidelity) with multicast Similar to ModSAF Complete data when subscribe to all grid-layers. Agent at LAN-WAN, as long entity has IE in group, agent wil stay in multicast group."	
STOW-E	1994	"Synthetic Theater of War - Europe One of the STOW programs DIS 2.03 with custom PDU Uses grid-based and PDU type filtering AOI, grid cell set Uses broadcasting for lan-wan-lan data (node-node)"	
StudierStube	1998	An environment for collaboration in augmented reality	[186]
Three-Tiered IM	1999	Based on Bamboo, providing an extensible interest management for scalable persistent distributed virtual environments	[187]
TRADE	1997	A transatlantic research and development environment.	[188]
UCAVE	2012	Ubiquitous collaborative activity virtual environments	[189]
Urbi et Orbi	2000	An asynchronous architecture to manage communication, display, and user interaction in distributed virtual environments.	[190]
VAST	2007	Voronoi-based Adaptive Scalable Transfer; a fully-distri- buted peer-to-peer protocol, designed to handle event mes- sages in MMOGs	[191]
VELVET	2003	An Adaptive Hybrid Architecture for Very Large Virtual Environments	[192]
VEOS	1994	Virtual Environment Operating Shell. emphasizes rapid prototyping, heterogeneous distributed computing, and portability.	[193]
VERN	1991	Virtual Environment Realtime Network, object oriented testbed for interconnection of envirnments over a network of graphical workstations.	[194]
Vesuvius	2007	Interactive Atmospheric Visualization in a Virtual Environment	[195]
Virtual Society	1995	Based on Community Place,	[196], [197]
VISINET	1997	collaborative 3D visualization and VR over ATM networks	[198]
VISTEL	1993	In VISTEL, the 3D models of the participants at different sites are combined into an artificially created 3D image of a virtual space, and by displaying the 3D image of the virtual space on the 3D screen at each site, the participants will get the feeling of meeting each other in a common space.	[199]
VLNET	1995	a shared virtual life network with virtual humans that provides a natural interface for collaborative working.	[200]
Von	2006	A Scalable Peer-to-Peer Network for Virtual Environments	[33]
VPARK	1999	a Networked Virtual Environment (NVE) System, called W-VLNET (Windows Virtual Life Network) and an Attraction Building System, able to create and modify attractions used in the NVE System.	[201]
WAVES	1993	Design of architectures for low-end distributed virtual environments	[202]
Whiskey	2007	Synchronous collaborative systems for distributed virtual environments in Java	[203]
Wolverine	2005	A distributed Scene-Graph Library	[204]

Appendix C

Hardware

This section lists an overview of all the hardware mentioned in more detail.

(i) Workstation, Intel Pentium 4 CPU at 3.40GHz, 2GiB RAM, NVidia GeForce 7800GT

(ii) Workstation, 2.4GHz Intel Core2 CPU, 2GiB RAM and an NVIDIA GeForce 7800 GT GHz, Windows XP.

(iii) Workstation, 2.4GHz Intel Core2 CPU, 2GiB RAM and an NVIDIA GeForce 8800GTX GHz, Windows XP.

(iv) Workstation, Intel Core i7 950 CPU. NVIDIA GeForce GTX 480, Windows 7.

(v) Workstation, Intel Core i7 X 980 CPU. NVIDIA GeForce GTX 480, Windows 7.

(vi) Workstation, Intel Core i7 3770 CPU. NVIDIA GeForce GTX 660, Windows 8.

(vii) Ultra Mobile Personal Computers are based on the Microsoft Origami specification released in 2006 and have been developed jointly by Microsoft, Intel, and Samsung, among others. UMPCs are basically small form factor mobile PCs running Microsoft Windows XP. The UMPC used is a Asus R2H. Intel Celeron M Processor ULV : GHz 900MHz;Intel 910GML Express Chipset;DDR 533 MHz SDRAM, 256MB on board;Embedded Intel GM965.

(viii) Tablet, Asus Slate EP121, Intel Dual-Core i5 470um, DDR3, 4GB SO-DIMM, 64GB SSD, Intel HD Graphics, Windows 8.

(ix) Motion tracker, Type Mtx.a small and accurate 3DOF Orientation Tracker. It provides drift-free 3D orientation as well as kinematic data: 3D acceleration, 3D rate of turn and 3D earth-magnetic field.

(x) Mobile phone HTC Hero, 288 MB RAM, 512 MB ROM, Qualcomm MSM7200A, 528 MHz, Adreno 130 GPU, Android OS, v1.5 (Cupcake).

(xi) Mobile phone HTC HD2, 448 MB RAM, 512 MB ROM, Qualcomm QSD8250 Snapdragon, 1 GHz (Scorpion) , Adreno 200 GPU, Microsoft Windows Mobile 6.5 Professional.

(xii) Mobile phone Samsung Omnia 7, 512 MB RAM, 8GB Flash, Qualcomm QSD8250 Snapdragon, 1 GHz (Scorpion), Adreno 200 GPU, Microsoft Windows Phone 7.5.

(xiii) Mobile phone Nokia Lumia 820, 1 GB RAM, 8GB Flash, Qualcomm Snapdragon S4 MSM8960, 1.5 GHz dual-core Qualcomm Krait, Adreno 225 GPU, Microsoft Windows Phone 8.

(xiv) Head-mounted display

(xv) Surface pro 2 tablet, Microsoft Windows 8.1, 4 GB Ram, Intel Core i5-4200u CPU 1.6Ghz, Intel Graphics.

Appendix D

Projects

The work presented in this thesis was supported by several projects. Each of these projects brought their own domain, limitations and requirements to the research topic. It broadened the topic, made it more varied and diverse as a whole level of complexity in overall design of the framework was added, which made it challenging and intriguing. Each section presents a description of the project and includes a section with the contributions correlated with the work presented in this thesis. Starting with the main project Intermedia bringing the main focus on user centric media, followed by the second most important contributing project 3DAH in terms of telemedicine, and last the two supportive projects Servive and Leapfrog for e-commerce and real-world applications. The descriptions given are the official descriptions from the project websites.

D.1 InterMedia

" There have been considerable efforts to have Audio Video systems and applications converge, in particular in home environments with homes as spaces of convergence, and for nomadic users with advanced mobile devices as points of convergence. These trends are important but also have limitations that we seek to address and overcome: home-centric systems fail to account for increased mobility and the desire to provide continuous service across spatial boundaries outside the home; device-centric convergence, e.g. in 3G phones, supports nomadic use but provides a very limited user experience as no single device and interface will fit many different applications well.

In this Network of Excellence [1] [2] we seek to progress beyond home and device-centric convergence toward truly user-centric convergence of multimedia. Our vision is The User as Multimedia Central: the user as the point at which services (multimedia applications) and the means for interacting with them (devices and interfaces) converge. Key to our vision is that users are provided with a personalized interface and with personalized content independently of the particular set of physical devices they have available for interaction (on the body, or in their environment), and independently of the physical space in which they are situated. Our approach to this vision is to investigate a flexible wearable platform that supports dynamic composition of wearable devices, an ad-hoc connection to devices in the environment, a continuous access to multimedia networks, as well as adaptation of content to devices and user context.

The concept of user-centric convergence liberates a nomadic user from carrying a range of mobile devices by providing personalized access to multimedia regardless of devices and networks. It is accom-

[1] Network of Excellence web-link: `http://cordis.europa.eu/fp6/instr_noe.htm`
[2] Project Intermedia web-link: `http://cordis.europa.eu/projects/rcn/79765_en.html`

plished not only by removing spatial constraints but also by making multimedia contents adapted to diverse devices and networks in our daily activities. An overview of the three key challenging issues (dynamic distributed networking, mobile and wearable interfaces, and multimedia content adaptation and handling) is shown in D.1 with their internal components. Dynamic distributed networking layer mainly focuses on a transparent access to diverse networks for seamless multimedia session continuity which enables a user to switch among different devices and networks with minimal manual intervention from the user. We defined two frameworks: PA Framework and Sensor Abstraction and Integration Layer (SAIL). The PA framework [55] provides a cross-layer user-centric mobility framework that accounts for terminal handover and session migration. It associates network addresses to the users and their sessions. It exploits context information to automate the processes of terminal handover and session migration. SAIL [92] gathers data from heterogeneous context sources and exports context information as they came from "virtual sensors"; through high-level standard interfaces (for example, HTTP and UPnP).

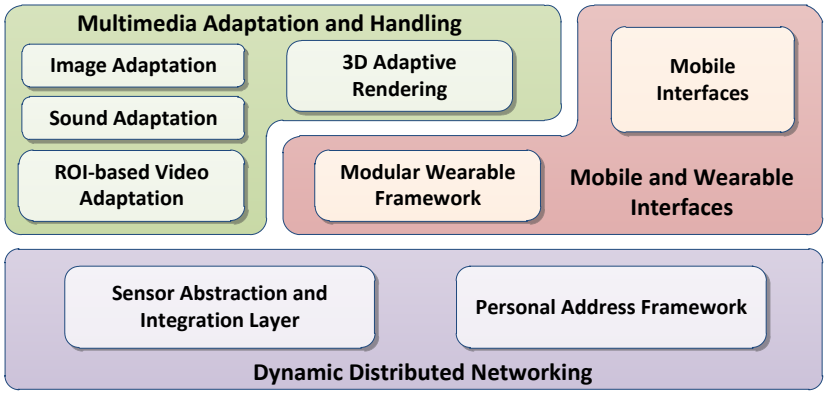


Figure D.1: InterMedia Research Challenges and Architecture[205]

The mobile and wearable interfaces layer provides various interfaces to access multimedia contents exploiting diverse devices nearby to users which make users free from using specific devices to access multimedia contents. We use a modular approach towards a wearable interface in a sense that users do not have to decide between several garments according to their fixed respective functionalities, but they can rather select and attach modules (e.g., UI, storage, localization sensors, and communication protocols, etc.) that will suit their needs. Thus, it becomes entirely personalized as it depends on the selected wearable modules, user profile, and the available surrounding devices. To overcome the limited output capabilities of mobile devices, in particular mobile phones, new mobile interfaces are also developed such as display technologies like projector phones, interactive surfaces and remote displays. Using those interfaces, multimedia can be explored and shared in a collaborative manner using new interaction techniques. Multimedia adaptation and handling layer support multimedia contents to be presented to different devices for personal manipulation which requires adaptation of multimedia to device or personal context along with seamless presentation of the multimedia for different devices. We provide diverse adaptation mechanisms according to multimedia types including video, image, sound, and 3D contents. We also provide a multimedia sharing architecture through dynamic networks with other users. ”

D.1.1 Contributions

The Intermedia project provided most of the application scenarios, which are Remote Rendering for Low-end Devices, Remote Rendering with Augmented Reality and Adaptive Rendering with Dynamic Device Switching. In the beginning of the project there was a strong focus on researching *indoor guidance systems*, which entailed a wearable mobile device or suit as described by Rigetti[206] and a head mounted display (HMD) which then augments or superimposes a 3D virtual environment on what was seen, with orientation corrections from a motion tracker[3]. The guidance system could be given commands by voice and an 3D avatar would walk in front of the user, guiding the user to its destination within the building. To give an impression figure D.2 shows an example, in which a prototype of the system is demonstrated, the user is wearing the HMD, with on top the orange Xsense motion tracker. The laptop to the right shows the 3D rendering as it is presented to the user. A basic styled rendering was chosen as it is being projected as it were over the real world. Therefore wire-frame models and bright colours were chosen.

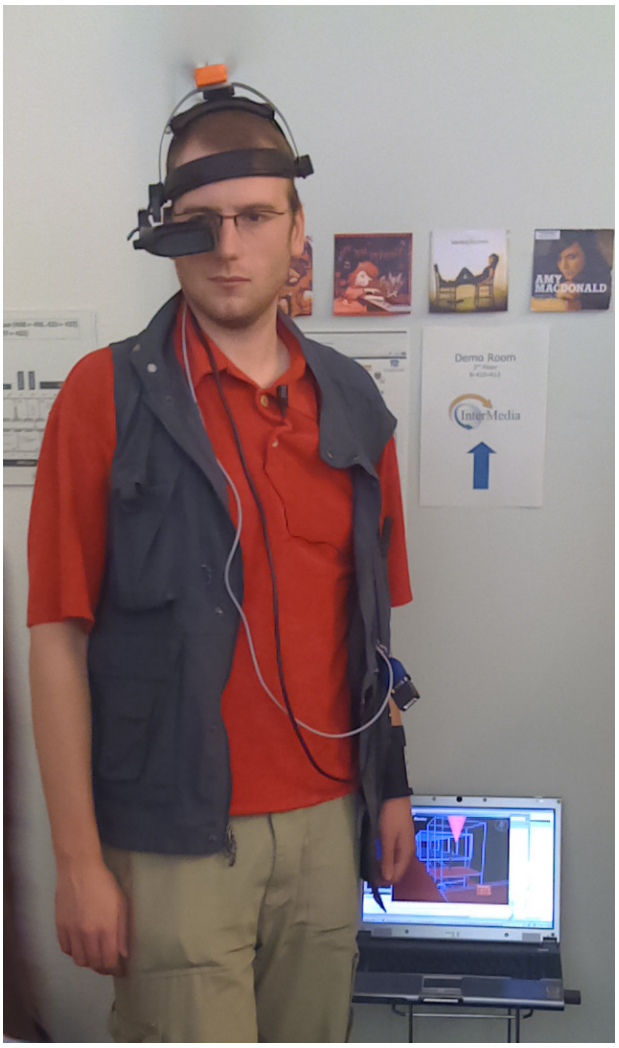


Figure D.2: Indoor guidance using Augmented Reality.

It is here where the first two scenarios are applied. The device that is with the user doesn't necessarily

[3]Xsense motion tracker see (ix)

contain all the information about the building, therefore it should be transferred. The full 3D rendering was performed on the device and displayed on the HMD[4]. As seen from the figure the HMD is a see through and one-eye display creating a superimposed image. This however demanded a lot of processing power from the mobile device, which was a UMPC[5], and therefore remote solutions looked at. At the second stage of the project the focus on augmented reality was lessened and shifted towards a stronger user-centric approach. This evolved into have still the wearable devices and have 3D content displayed on it. The 3D content however is rendered remotely and streamed to the device all the while a user session is kept, capable of switching from device to device without inturrupting the 3D content streaming. Furthermore we contributed to the collaborative aspects that were added without impeding the user-centric approach. This resulted into having uncommon devices (e.g. interactive table), portable devices and common devices (e.g. PC) running their own perspective clients that all connect and visualize a common 3D environment in which multi-user interaction is made possible.

D.2 3D Anatomical Human

“The objective of this network is to increase, by scientific exchange, the development of new technologies and knowledge around virtual representations of human body for interactive medical applications. The network has a specific goal: developing realistic functional three-dimensional models for the human musculoskeletal system, the methodology being demonstrated on the lower limb.

3DAnatomicalHuman[6] is a Marie Curie Research Training Network project within EU's Sixth Framework Program[7]. These Networks provide the means for research teams of recognized international stature to link up, in the context of a well-defined collaborative research project, in order to formulate and implement a structured training program for researchers in a particular field of research.

The objective of this project is to train a body of researchers in the various domains involving the modelling and simulation of human body for medical purposes. The network will be naturally pluri-disciplinary and pluri-institutional and will:

- Promote the culture of pluri-disciplinary research applied to concrete problems of the real world.
- Bridge complementary approaches for modelling and simulating the human musculoskeletal system through the development of modelling and simulation methods with different level of details.
- Improve the learning support for medical training.
- Increase awareness of the use of virtual reality technologies to real clinical problems.
- Demonstrate the feasibility and efficiency of virtual and augmented reality techniques to reproduce with realism not only the shape but also certain physiological processes, and provide additional information usually non-visible, like stress or temperature distribution on anatomical structures.
- Integrate Knowledge Management in functional simulation in order to provide high-level models (framework for information management) and improve the acceptance of simulation models by clinicians (medical semantics).

”

[4]head mounted display, see (xiv)
[5]UMPC mobile device, see (vii)
[6]Project 3DAH web-link: `http://cordis.europa.eu/projects/rcn/82439_en.html`
[7]Marie Curie Research Training Network web-link: `http://ec.europa.eu/research/fp6/mariecurie-actions/indexhtm_en.html`

D.2.1 Contributions

A stronger focus on collaboration was given within this project, where we used various data generated within the project itself (such as MRI scan data) and provide a view on what could be possible for Telemedicine with the researched technologies. This is shown in application scenario 3.9, which combines the automatic segmentation (Schmid and Magenenat-Thalmann [76]) of MRI data with manual segmentation into a collaborative environment. The difficulty here lies within the simultaneous interaction of the users with a simulation running in the background and to overcome device heterogeneity. The MRI which can be huge amount of data is too much to handle for a hand-held device and therefore a service provided, only provides a *view* on the data specifically for the given session running on the device. Furthermore the sharing and synchronization of data between users had to be established as well as application logic that separates interaction data and surface contour data and visualizes it. The interaction in general is provided equally, albeit on a PC a mouse is used and on a smart-phone the touch based input. The experiment in section 5.6 shows two types of collaboration, strict and relaxed locking of actions. The collaborative interaction all happens in a 2D space, as the view on the MRI data gives a "sliced" 2D-image. The segmentation output however is a 3-dimensional surface model, which is rendered at the server side, where the simulation is also running. In order to have the 3D view visualized on the client, prior streaming methods are utilized and per client interaction on the 3D scene are used for rotating and translating the virtual camera. Other contributions to this project were in close collaboration with the Intermedia project, as a collaborative learning tool in application scenario Adaptive Rendering with Dynamic Device Switching. With bringing user-centric media which in this case is the medical data into an E-Learning environment.

D.3 Servive

"The project *Service oriented intelligent value adding network for clothing-SMEs embarking in Mass-Customisation*, in short Servive, proposes the enlargement of the assortment of customizable clothing items currently on offer, the enhancement of all co-design aspects and the development and testing of a new production model based on decentralized networked SME cells[8]. The Servive network links critical Mass-Customization enabling services and adapt these services to the specific needs of well-defined target customer groups.

The main activity in the Servive project is formed by the conceptualization, design and development of a model configuration module and the Virtual-Try-On Web Service. This is a server-based and real-time garment simulation solution that presents the results of the customer's personalized garments in an appropriate form for comfort and fit evaluation based on the customer's physiology."

D.3.1 Contributions

The Servive and Leapfrog projects brought aspects of E-Commerce and are mostly expressed and elaborated in application scenario Service Distribution and Render Context Switching. Here we contributed to enable single user access to a cloud-based service that offers a 3D content rendering service. In this case clothing on a virtual avatar are being rendered, which can be customized by the user. These kind of services are not really new, but are offered in either 2D and/or pre-rendered content. The challenge here was to overcome this and provide interactive 3D content through a browser without burdening the end-device with heavy processing demands in the process. Therefore the developed architecture was

[8]Project Servive web-link: `http://cordis.europa.eu/projects/rcn/89318_en.html`

deployed together with methods for load balancing and managing the server side services. The client software was based on a partial implementation of the proposed architecture in Java, which then could be run inside a browser as an *applet* object.

D.4 Leapfrog

“The project *Leadership for European Apparel Production From Research along Original Guidelines* (LEAPFROG) main objective is to modernize and ultimately transform the clothing sector into a flexible knowledge-driven high-tech industry and to preserve its long-term prosperity and competitiveness in an enlarged Europe[9]. In order to achieve such a long-term industrial transformation the LEAPFROG initiative will focus on 3 major objectives.

- A radical re-engineering and intelligent automation of the current garment manufacturing process will create the clothing factory of the future.
- A radical move towards rapid customized manufacturing through flexible and integration of cost-effective and sustainable processes from fabric processing to customer delivery.
- A paradigm change in customer service and customer relationship management with a focus on value-adding product-services.

With these 3 sub-goals in mind, the targeted breakthrough transformation is centred on the development of a flexible automated manufacturing system for the cost-effective production of high-quality customized garments that fully address the customer requirements.”

D.4.1 Contributions

Likewise to the Servive project this project extended the scenario by introducing collaborative aspects and inclusion of utilizing other devices. From different kinds of devices multiple users are able to access the same service and interact with it. The challenge here was to have on the server side multiple sessions active in the same 3D simulation service. The interaction remained the same, thus changing the avatar body sizes and garment selection, however since there are multiple users interacting from different devices this became a challenge. An easy but short term solution could’ve been to have the interface integrated in the end-client, as it was more or less done with the Servive prototype, but this severely limits the usage and extension of such kinds of services. Therefore bringing it to a more generalized requirement a more dynamic solution was provided, where each device provides a profile and on the server side the user interface was rendered directly into the 3D scene. This offered a good way to overcome this problem, with additionally the ease of updating at the server side without having to update the client side. In later stages the Servive and Leapfrog challenges were combined and further extended into the last application scenario 3.10. Which aims at having a fully collaborative service based environment capable of exploiting the full pipeline for making garments as patterns, extract and convert them into 3D meshes and simulate these in dynamically created services.

[9]Project Leapfrog web-link: `http://cordis.europa.eu/projects/rcn/80097_en.html`

Bibliography

[1] J. Grudin, "Computer-supported cooperative work: History and focus", *Computer*, vol. 27, no. 5, pp. 19–26, 1994.

[2] P. Fröhlich, R. Simon, and L. Baillie, "Mobile Spatial Interaction", *Personal and Ubiquitous Computing*, vol. 13, no. 4, pp. 251–253, Jul. 2008.

[3] G. Anastasi, M. Conti, E. Gregori, A. Passarella, and L. Pelusi, "An energy-efficient protocol for multimedia streaming in a mobile environment", *International Journal of Pervasive Computing and Communications*, vol. 1, no. 4, pp. 301–312, 2005.

[4] A. M. Krebs, I. Marsic, and B. Dorohonceanu, "Mobile adaptive applications for ubiquitous collaboration in heterogeneous environments", in *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, vol. 1, 2002, pp. 401–407.

[5] P Eisert and P Fechteler, *Low delay streaming of computer graphics*. IEEE, Oct. 2008, pp. 2704–2707.

[6] M. Preda, P. Villegas, F. Morán, and G Lafruit, "A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering", *The Visual Computer*, pp. 881–888, 2008.

[7] K Engel, T Ertl, P Hastreiter, and B Tomandl, "Combining local and remote visualization techniques for interactive volume rendering in medical applications", *VIS '00 Proceedings of the conference on Visualization '00*, 2000.

[8] K. Weaver and D. Parkhurst, "Perceptually adaptive rendering of immersive virtual environments", in *Smart graphics: 8th international symposium*, vol. 4569, Springer Berlin / Heidelberg, 2007, pp. 224–229.

[9] M Beaudouin-Lafon and Others, *Computer supported co-operative work*. Wiley, 1999.

[10] L. Damianos, L. Hirschman, R. Kozierok, J. Kurtz, A. Greenberg, K. Walls, S. Laskowski, and J. Scholtz, "Evaluation for collaborative systems", *ACM Computing Surveys (CSUR)*, vol. 31, no. 2es, p. 15, 1999.

[11] C Joslin, "Trends in networked collaborative virtual environments", *Computer Communications*, vol. 26, no. 5, pp. 430–437, Mar. 2003.

[12] I. J. Grimstead, N. J. Avis, and D. W. Walker, "Visualization across the pond: how a wireless PDA can collaborate with million-polygon datasets via 9,000km of cable", in *Proceedings of the tenth international conference on 3D Web technology - Web3D '05*, vol. 1, New York, New York, USA: ACM Press, 2005, p. 47.

[13] J. Dyck, "A Survey of Application-Layer Networking Techniques for Real-time Distributed Groupware", University of Saskatchewan, Tech. Rep., 2006, pp. 1–38.

[14] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White, "Scalability for Virtual Worlds", in *2009 IEEE 25th International Conference on Data Engineering*, Ieee, Mar. 2009, pp. 1311–1314.

[15] J. Calvin, A. Dickens, and B. Gaines, "The SIMNET virtual world architecture", *Virtual Reality*, pp. 450–455, 1993.

[16] T. R. Tiernan, "Synthetic Theater of War - Engineering Demonstration", Naval Command, Control and Ocean Surveillance Center, RDT&E Division, Tech. Rep. June, 1996.

[17] J. F. Patterson, R. D. Hill, S. L. Rohall, and S. W. Meeks, "Rendezvous: an architecture for synchronous multi-user applications", in *Proceedings of the 1990 ACM conference on Computer-supported cooperative work - CSCW '90*, New York, New York, USA: ACM Press, 1990, pp. 317–328.

[18] C Morningstar and F. Farmer, "The Lessons of Lucasfilm's Habitat", in *Cyberspace: First Steps*, vol. 1, 1991, pp. 1–21.

[19] F. R. Farmer, "Social dimensions of Habitat's citizenry", in *True Names: and the Opening of the Cyberspace Frontier*, J. Frenkel, Ed., Tor, 2001, pp. 205–212.

[20] S. Deering, "Host extensions for IP multicasting", Internet Engineering Task Force, Tech. Rep. July, 1986, pp. 1–20.

[21] M. Zyda, M. Macedonia, D. Pratt, and P. Barham, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments", *Presence teleoperators and virtual environments*, vol. 3, no. 4, pp. 265–287, 1994.

[22] C. Greenhalgh and S. Benford, "MASSIVE: a collaborative virtual environment for teleconferencing", *ACM Transactions on Computer-Human Interaction*, vol. 2, no. 3, pp. 239–261, Sep. 1995.

[23] J. Barrus, R. Waters, and D. Anderson, "Locales: Supporting large multiuser virtual environments", *IEEE Computer Graphics and Applications*, no. November, pp. 50–57, 1996.

[24] O. Hagsand, "Interactive multiuser VEs in the DIVE system", *Multimedia, IEEE*, vol. 3, no. 1, pp. 30–39, 1996.

[25] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast", *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 15–26, 2006.

[26] S. Perlman, *The Process of Invention: OnLive Video Game Service*, 2010.

[27] L. Higgins, G. McDowell, and B. VonTobel, "Tunneling multicast traffic through non-multicast-aware networks and encryption devices", in *2001 MILCOM Proceedings Communications for Network-Centric Operations: Creating the Information Force (Cat. No.01CH37277)*, vol. 1, IEEE, 2001, pp. 296–300.

[28] J. Mogul, *Broadcasting Internet Datagrams*, 1984.

[29] J. Moy, *Multicast Extensions to OSPF*, United States, 1994.

[30] B. Haberman and D. Thaler, *Unicast-Prefix-based IPv6 Multicast Addresses*, 2002.

[31] S. Pettifer, J. Cook, J. Marsh, and A. West, "DEVA3", in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '00*, New York, New York, USA: ACM Press, 2000, p. 33.

[32] S. Hu, "Scalable peer-to-peer networked virtual environment", *of 3rd ACM SIGCOMM workshop on Network and*, pp. 129–133, 2004.

[33] J.-f. Chen and T.-h. Chen, "VON: a scalable peer-to-peer network for virtual environments", *IEEE Network*, vol. 20, no. 4, pp. 22–31, Jul. 2006.

[34] D. Lee, M. Lim, and S. Han, "ATLAS", in *Proceedings of the 4th international conference on Collaborative virtual environments*, vol. 16, New York, USA: ACM Press, Apr. 2002, pp. 47–54.

[35] H. Coltzau, "P2Life : An Infrastructure for Networked Virtual Marketplace Environments", *IJIIP: International Journal of Intelligent Information Processing*, vol. 1, no. 2, pp. 1–13, 2010.

[36] L. Fan, P. Trinder, and H. Taylor, "Design issues for Peer-to-Peer Massively Multiplayer Online Games", *International Journal of Advanced Media and Communication*, vol. 4, no. 2, p. 108, 2010.

[37] S. Banerjee, "A comparative study of application layer multicast protocols", *Network*, 2002.

[38] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games", *Electronic Library, The*, pp. 1–8, 2002.

[39] D. Van Hook, S. Rak, and J. Calvin, "Approaches to relevance filtering", in *Proc. of 11th DIS Workshop*, Citeseer, 1994, pp. 1–3.

[40] D. V. Hook, D. Cebula, and S. Rak, "Performance of stow ritn application control techniques", MIT Lincoln Laboratory, Tech. Rep., 1996.

[41] K. L. Morse, L. Bic, and M. Dillencourt, "Interest Management in Large-Scale Virtual Environments", *Presence: Teleoperators <html_ent glyph="@amp;" ascii="&"/> Virtual Environments*, vol. 9, no. 1, pp. 52–68, Feb. 2000.

[42] D. Ding and M. Zhu, "A model of dynamic interest management: interaction analysis in collaborative virtual environment", in *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, 2003, pp. 223–230.

[43] G. Morgan, F. Lu, and K. Storey, "Interest Management Middleware for Networked Games", *Exchange Organizational Behavior Teaching Journal*, vol. 1, no. 212, pp. 57–64, 2005.

[44] S. Han and D. Lee, "A deputy object based presentation semantics split application model for synchronous collaboration in ubiquitous computing environments", *Proceedings of the Third International Conference on Collaboration Technologies*, 2007.

[45] C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer, "A3: A Novel Interest Management Algorithm for Distributed Simulations of MMOGs", in *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, IEEE, Oct. 2008, pp. 35–42.

[46] D. T. Ahmed and S. Shirmohammadi, "A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs", *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pp. 27–34, Oct. 2008.

[47] B. Hariri, S. Shirmohammadi, and M. R. Pakravan, "A distributed interest management scheme for massively multi-user virtual environments", *2008 IEEE Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pp. 111–115, Jul. 2008.

[48] R. Minson and G. Theodoropoulos, "Load Skew in Cell-Based Interest Management Systems", in *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, IEEE Computer Society, Oct. 2008, pp. 43–50.

[49] K. Morse, "Interest management in large-scale distributed simulations", *Management*, pp. 1–16, 1996.

[50] E. Léty, T. Turletti, and F. Baccelli, "SCORE: a scalable communication protocol for large-scale virtual environments", *Networking, IEEE/ACM Transactions on*, vol. 12, no. 2, pp. 247–260, 2004.

[51] S. Lee and S. Ko, "Adapting content for mobile devices in heterogeneous collaboration environments", *Proceedings of the International Conference on*, pp. 0–7, 2003.

[52] G. Krasner, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system", *Journal of object oriented programming*, vol. 1, pp. 26–49, 1988.

[53] N. Sawant, J. Li, and J. Z. Wang, "Automatic image semantic interpretation using social action and tagging data", *Multimedia Tools and Applications*, vol. 51, no. 1, pp. 213–246, Nov. 2010.

[54] Y. Tian, J. Srivastava, T. Huang, and N. Contractor, "Social Multimedia Computing", *Computer*, vol. 43, no. 8, pp. 27–36, Aug. 2010.

[55] R. Bolla, S. Mangialardi, R. Rapuzzi, and M. Repetto, "Streaming Multimedia Contents to Nomadic Users in Ubiquitous Computing Environments", in *IEEE INFOCOM Workshops 2009*, IEEE, Apr. 2009, pp. 1–6.

[56] J. M. Vlissides and M. a. Linton, "Unidraw a framework for building domain-specific graphical editors", *ACM Transactions on Information Systems*, vol. 8, no. 3, pp. 237–268, Jul. 1990.

[57] I. Marsic, "Real-time collaboration in heterogeneous computing environments", in *Proceedings International Conference on Information Technology: Coding and Computing (Cat. No.PR00540)*, IEEE Comput. Soc, 2000, pp. 222–227.

[58] C. Papadopoulos, "Improving Awareness in Mobile CSCW", *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, pp. 1331–1346, Oct. 2006.

[59] R Wootton, J Craig, and V Patterson, *Introduction to telemedicine*. The Royal Society of Medicine Press Ltd, Second edition, London, 2006.

[60] J Marescaux, J Leroy, M Gagner, F Rubino, D Mutter, M Vix, S. E. Butner, and M. K. Smith, "Transatlantic robot-assisted telesurgery", *Nature*, vol. 413, pp. 379–380, 2001.

[61] V Rialle, J. B. Lamy, N Noury, and L Bajolle, "Telemonitoring of patients at home: a software agent approach", *Comput. Meth. Programs. Biomed.*, vol. 72, no. 3, pp. 257–268, 2003.

[62] J Zhang, J. N. Stahl, H. K. Huang, X Zhou, S. L. Lou, and K. S. Song, "Real-Time Teleconsultation with High-Resolution and Large-Volume Medical Images for Collaborative Healthcare", *IEEE Trans. Inform. Tech. Biomed.*, vol. 4, no. 2, pp. 178–185, 2000.

[63] S. Park, W. Kim, and I. Ihm, "Mobile collaborative medical display system.", *Computer methods and programs in biomedicine*, vol. 89, no. 3, pp. 248–60, Mar. 2008.

[64] L Constantinescu, J Kim, C Chan, and D Feng, "Automatic Mobile Device Synchronization and Remote Control System for High-Performance Medical Applications", in *IEEE Proc. Engineering in Medicine and Biology Society (EMBS)*, 2007, pp. 2799–2802.

[65] F. Simmross-Wattenberg, N. Carranza-Herrezuelo, C. Palacios-Camarero, P. Casaseca-de-la Higuera, M. A. Martín-Fernández, S. Aja-Fernández, J. Ruiz-Alzola, C.-F. Westin, and C. Alberola-López, "Group-Slicer: a collaborative extension of 3D-Slicer.", *Journal of biomedical informatics*, vol. 38, no. 6, pp. 431–42, Dec. 2005.

[66] J Schmid, N Nijdam, S Han, J Kim, and N Magnenat-Thalmann, "Interactive Segmentation of Volumetric Medical Images for Collaborative Telemedicine", in *Modelling the Physiological Human, Proc. 3D Physiological Human Workshop*, Springer, Dec. 2009, pp. 13–24.

[67] M. Culnan and P. Mchugh, "How large US companies can use Twitter and other social media to gain business value", *MIS Quarterly*, 2010.

[68] P. Eugster, "Type-based publish/subscribe: Concepts and experiences", *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 1, p. 6, 2007.

[69] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, A. Professional, Ed. 1994.

[70] M. Kasap and N. Magnenat-Thalmann, "Parameterized Human Body Model for Real-Time Applications", in *2007 International Conference on Cyberworlds (CW'07)*, IEEE, Oct. 2007, pp. 160–167.

[71] E. Lyard and N. Magnenat-Thalmann, "A simple footskate removal method for virtual reality applications", *The Visual Computer*, vol. 23, no. 9-11, pp. 689–695, Jun. 2007.

[72] ——, "Motion adaptation based on character shape", *Computer Animation and Virtual Worlds*, vol. 19, no. 3-4, pp. 189–198, 2008.

[73] N. Magnenat-thalmann, E. Lyard, M. Kasap, and P. Volino, "Adaptative Body , Motion and Cloth", in *Motion in Games - Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, 2008, pp. 63–71.

[74] P. Volino, "Accurate garment prototyping and simulation", *Computer-Aided Design and Applications*, vol. 2, no. 5, pp. 645–654, 2005.

[75] B. Kevelham and N. Magnenat-Thalmann, "Fast and accurate GPU-based simulation of virtual garments", *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry - VRCAI '12*, p. 223, 2012.

[76] J Schmid and N Magnenat-Thalmann, "MRI Bone Segmentation using Deformable Models and Shape Priors", in *MICCAI 2008, Part I. LNCS*, D Metaxas, L Axel, G Szekely, and G Fichtinger, Eds., vol. 5241, Springer-Verlag Berlin Heidelberg, Sep. 2008, pp. 119–126.

[77] P. Volino, F. Cordier, and N. Magnenat-Thalmann, "From early virtual garment simulation to interactive fashion design", *Computer-Aided Design*, vol. 37, no. 6, pp. 593–608, May 2005.

[78] N. Magnenat-thalmann, F. Dellas, C. Luible, and P. Volino, "From Roman Garments to Haute-couture Collection with the Fashionizer Platform", in *Virtual Systems and Multi Media*, OCSL Press Japan, 2004, pp. 2–12.

[79] D. Wagner and D. Schmalstieg, "Artoolkitplus for pose tracking on mobile devices", in *Computer Vision Winter Workshop*, 2007.

[80] J. Newman, G. Schall, I. Barakonyi, A. Schürzinger, and D. Schmalstieg, "Wide-Area Tracking Tools for Augmented Reality", in *Advances in Pervasive Computing 2006, Adjunct Proceedings of Pervasive*, Oesterreichische Computer-Gesellschaft, 2006, pp. 143–146.

[81] J. Hagberg, M. Thor, and M. Wiklander, "3DVN : A Mixed Reality Platform for Mobile Navigation Assistance", Chalmers University of Technology and Goteborg University, Tech. Rep., 2006, Report no. 2006–12.

[82] A. D. Cheok, S. W. Fong, K. H. Goh, X. Yang, and W. Liu, "Human Pacman: A Mobile Entertainment System with Ubiquitous Computing and Tangible Interaction over a Wide Outdoor Area", in *Mobile HCI 2004 â€" 6th International Symposium*, Glasgow, UK, 2004, pp. 209–223.

[83] S. Balcisoy, R. Torre, M. Ponder, P. Fua, and D. Thalmann, "Augmented reality for real and virtual humans", in *Proceedings Computer Graphics International 2000*, IEEE Comput. Soc, 2000, pp. 303–307.

[84] C. Hughes, C. Stapleton, D. Hughes, and E. Smith, "Mixed Reality in Education, Entertainment, and Training", *IEEE Computer Graphics and Applications*, vol. 25, no. 6, pp. 24–30, Nov. 2005.

[85] H. Tamura, H. Yamamoto, and A. Katayama, "Mixed reality: future dreams seen at the border between real and virtual worlds", *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 64–70, 2001.

[86] G. Papagiannakis, S. Schertenleib, B. O'Kennedy, M. Arevalo-Poizat, N. Magnenat-Thalmann, A. Stoddart, and D. Thalmann, "Mixing virtual and real scenes in the site of ancient Pompeii", *Computer Animation and Virtual Worlds*, vol. 16, no. 1, pp. 11–24, Feb. 2005.

[87] A. Egges, G. Papagiannakis, and N. Magnenat-Thalmann, "Presence and interaction in mixed reality environments", *The Visual Computer*, vol. 23, no. 5, pp. 317–333, Mar. 2007.

[88] M. Billinghurst, H. Kato, and I. Poupyrev, "The MagicBook - moving seamlessly between reality and virtuality", *IEEE Computer Graphics and Applications*, vol. 21, no. 3, pp. 6–8, 2001.

[89] I. Barakonyi and D. Schmalstieg, "Ubiquitous animated agents for augmented reality", in *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, IEEE, Oct. 2006, pp. 145–154.

[90] D. Wagner, M. Billinghurst, and D. Schmalstieg, "How real should virtual characters be?", in *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology - ACE '06*, New York, New York, USA: ACM Press, 2006, p. 57.

[91] A. Schmeil and W. Broll, "MARA", in *ACM SIGGRAPH 2006 Sketches on - SIGGRAPH '06*, vol. 21, New York, New York, USA: ACM Press, 2006, p. 141.

[92] M. Girolami, S. Lenzi, F. Furfari, and S. Chessa, "SAIL: A Sensor Abstraction and Integration Layer for Context Awareness", *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 374–381, Sep. 2008.

[93] A. Hang, E. Rukzio, and A. Greaves, "Projector phone", in *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services - MobileHCI '08*, New York, New York, USA: ACM Press, 2008, p. 207.

[94] K. Cheverst, A. Dix, D. Fitton, C. Kray, M. Rouncefield, C. Sas, G. Saslis-Lagoudakis, and J. G. Sheridan, "Exploring bluetooth based mobile phone interaction with the hermes photo display", in *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services - MobileHCI '05*, New York, New York, USA: ACM Press, 2005, p. 47.

[95] S Jeon, G. Kim, and M Billinghurst, "Interacting with a tabletop display using a camera equipped mobile phone", *Human-Computer Interaction. Interaction ...*, 2007.

[96] N. Nijdam, S. Han, B. Kevelham, and N. Magnenat-Thalmann, "A context-aware adaptive rendering system for user-centric pervasive computing environments", in *Melecon 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, IEEE, 2010, pp. 790–795.

[97] D. Delaney, T. Ward, and S. McLoone, "On consistency and network latency in distributed interactive applications: a survey–part I", *Presence: Teleoper. Virtual Environ.*, vol. 15, no. 2, pp. 218–234, 2006.

[98] D Delaney and T Ward, "On consistency and network latency in distributed interactive applications: A survey-Part II", *Presence: Teleoperators and Virtual*, 2006.

[99] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *distributed systems concepts and design*. Addison-Wesley, 2011.

[100] P. T. Eugster, P. a. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe", *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[101] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA Â©1999, 1999.

[102] B Gilles, L Moccozet, and N Magnenat-Thalmann, "Anatomical modelling of the musculoskeletal system from MRI", in *MICCAI 2006, LNCS*, R Larsen, M Nielsen, and J Sporring, Eds., vol. 4190, Oct. 2006, pp. 289–296.

[103] J. G. Snel, H. W. Venema, and C. a. Grimbergen, "Deformable triangular surfaces using fast 1-D radial Lagrangian dynamics–segmentation of 3-D MR and CT images of the wrist.", *IEEE transactions on medical imaging*, vol. 21, no. 8, pp. 888–903, Aug. 2002.

[104] H. Delingette, "General Object Reconstruction based on Simplex Meshes", *International Journal of Computer Vision*, vol. 32, no. 2, pp. 111–146, 1999.

[105] T Heimann, S Munzing, H Meinzer, and I Wolf, "A Shape-Guided Deformable Model with Evolutionary Algorithm Initialization for 3D Soft Tissue Segmentation", in *Proc. Int. Conf. Information Processing in Medical Imaging (IPMI)*, N Karssemeijer and B Lelieveldt, Eds., vol. 4584, Springer, 2007, pp. 1–12.

[106] D. Kainmueller, H. Lamecker, S. Zachow, and H.-C. Hege, "An articulated statistical shape model for accurate hip joint segmentation.", *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2009, pp. 6345–51, Jan. 2009.

[107] M. J. Costa, H. Delingette, S. Novellas, and N. Ayache, "Automatic Segmentation of Bladder and Prostate Using Coupled 3D Deformable Models", in *Int. Conf. on medical image computing and computer assisted intervention (MICCAI)*, vol. 4791, 2007, pp. 252–260.

[108] S. D. Olabarriaga and a. W. Smeulders, "Interaction in the segmentation of medical images: a survey.", *Medical image analysis*, vol. 5, no. 2, pp. 127–42, Jun. 2001.

[109] M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-thalmann, and D. Thalmann, "VHD++ Development Framework: Towards Extendible, Component Based VR/AR Simulation Engine Featuring Advanced Virtual Character Technologies", in *Computer Graphics International*, IEEE Publisher, 2003, pp. 96–104.

[110] J. Rohlf and J. Helman, "IRIS performer", in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*, New York, New York, USA: ACM Press, 1994, pp. 381–394.

[111] M. Lim, N. Nijdam, and N. Magnenat-Thalmann, "A general collaborative platform for mobile multi-user applications", in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, IEEE, Sep. 2008, pp. 1346–1353.

[112] M. Lim, B. Kevelham, N. Nijdam, and N. Magnenat-Thalmann, "Rapid development of distributed applications using high-level communication support", *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 172–182, Jan. 2011.

[113] J.-l. Gailly and M. Adler, *Zlib*.

[114] D. Schmidt, "Design and realization of an interactive multi-touch table", vol. 2010, 2010.

[115] F Chehimi, N Nijdam, and D Schmidt, "An interactive table supporting mobile phone interaction and 3D content", *Ubicomp 2009: Ubiquitous Computing*, 2009.

[116] J. Lewis, "IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use", *International Journal of Human Computer Interaction*, vol. 7, no. 1, pp. 57–78, Jan. 1995.

[117] A. Wilson and R. Weatherly, "The aggregate level simulation protocol: an evolving system", in *Proceedings of Winter Simulation Conference*, IEEE, 1994, pp. 781–787.

[118] P. García, O. Montalà, C. Pairot, R. Rallo, and A. G. Skarmeta, "MOVE: Component Groupware Foundations for Collaborative Virtual Environments", in *Proceedings of the 4th international conference on Collaborative virtual environments - CVE '02*, New York, New York, USA: ACM Press, 2002, pp. 55–62.

[119] B. Anand, K. Thirugnanam, L. T. Long, D.-D. Pham, A. L. Ananda, R. K. Balan, and M. C. Chan, "ARIVU: Power-aware middleware for multiplayer mobile games", in *2010 9th Annual Workshop on Network and Systems Support for Games*, IEEE, Nov. 2010, pp. 1–6.

[120] D. N. Snowdon and A. J. West, "AVIARY- Design issues for future large-scale virtual environments", *Presence - Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 288–308, 1994.

[121] H. Tramberend, "Avocado: A distributed virtual reality framework", in *IEEE Conference on Virtual Reality*, Los Alamitos, CA, 1999, pp. 14–21.

[122] H. Tramberend, "Avocado: A Distributed Virtual Environment Framework", PhD thesis, Bielefeld, 2003.

[123] K. Watsen and M. Zyda, "Bamboo-a portable system for dynamically extensible, real-time, networked, virtual environments", in *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium (Cat. No.98CB36180)*, IEEE Comput. Soc, 1998, pp. 252–259.

[124] ——, "Bamboo - Supporting Dynamic Protocols for Virtual Environments", in *The IMAGE 98 Conference*, Scottsdale, Arizona, 1998, pp. 1–9. arXiv: 10.1.1.28.2404.

[125] P. Tandler, "The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments", *Journal of Systems and Software*, vol. 69, no. 3, pp. 267–296, Jan. 2004.

[126] G. Singh, L. Serra, W. Png, a. Wong, and H. Ng, "BrickNet: sharing object behaviors on the Net", *Proceedings Virtual Reality Annual International Symposium '95*, pp. 19–25, 1995.

[127] K. Nakabayashi and M. Maruyama, "An intelligent tutoring system on the WWW supporting interactive simulation environments with a multimedia viewer control mechanism", *WebNet 96 Conference Proceedings*, p. 9, 1996.

[128] J. Leigh and A. Johnson, "CALVIN: an immersimedia design environment utilizing heterogeneous perspectives", in *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, IEEE Comput. Soc. Press, 1996, pp. 20–23.

[129] J. Leigh and A Johnson, "CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments", *Virtual Reality: Research,*, 1997.

[130] K. S. Park, Y. J. Cho, N. K. Krishnaprasad, C. Scharver, M. J. Lewis, J. Leigh, and A. E. Johnson, "CAVERNsoft G2", in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '00*, New York, New York, USA: ACM Press, 2000, p. 8.

[131] T. Mastaglio and R. Callahan, "A large-scale complex virtual environment for team training", *Computer*, vol. 28, no. 7, pp. 49–56, Jul. 1995.

[132] A. Bharambe and J. Pang, "Colyseus: a distributed architecture for online multiplayer games", in *Symposium on Networked Systems Design and Implementation*, 2006, pp. 3–6.

[133] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda, "Community Place", in *Proceedings of the second symposium on Virtual reality modeling language - VRML '97*, New York, New York, USA: ACM Press, 1997, pp. 41–50.

[134] A. Bierbaum, "Continuum", PhD thesis, Iowa State University, 2007.

[135] M. Liska and P. Holub, "CoUniverse: Framework for Building Self-Organizing Collaborative Environments Using Extreme-Bandwidth Media Applications", in *2009 Eighth International Conference on Networks*, E. César, M. Alexander, A. Streit, J. L. Träff, C. Cérin, A. Knüpfer, D. Kranzlmüller, and S. Jha, Eds., ser. Lecture Notes in Computer Science, vol. 5415, Berlin, Heidelberg: IEEE, 2009, pp. 259–265.

[136] M. Liska, "Self-organizing Collaborative Environments", PhD thesis, Masaryk Universiy, 2010.

[137] S.-H. Ryu, H.-J. Kim, J.-S. Park, Y.-w. Kwon, and C.-S. Jeong, "Collaborative object-oriented visualization environment", *Multimedia Tools and Applications*, vol. 32, no. 2, pp. 209–234, Nov. 2006.

[138] E. Frecon, G. Smith, A. Steed, M. Stenius, and O. Stahl, "An overview of the COVEN platform", *Presence: Teleoperators &*, pp. 109–127, 2001.

[139] A. Pang and C. Wittenbrink, "Collaborative 3D visualization with CSpray", *IEEE Computer Graphics and Applications*, vol. 17, no. 2, pp. 32–41, 1997.

[140] C. Lascara, G. Wheless, D. Cox, and R. Patterson, "Tele-immersive virtual environments for collaborative knowledge discovery", *Urbana*, 1999.

[141] R. Lau, "Cyberwalk: a web-based distributed virtual walkthrough environment", *IEEE Transactions on Multimedia*, vol. 5, no. 4, pp. 453–465, Dec. 2003.

[142] R. Litiu and A. Parakash, "Developing adaptive groupware applications using a mobile component framework", in *Proceedings of the 2000 ACM conference on Computer supported cooperative work - CSCW '00*, New York, New York, USA: ACM Press, 2000, pp. 107–116.

[143] M. Slater, M. Usoh, S. Benford, D. Snowdon, C. Brown, T. Rodden, G. Smith, and S. Wilbur, "Distributed extensible virtual reality laboratory (DEVRL): a project for co-operation in multiparticipant environments", in *Proceedings of the Eurographics workshop on Virtual environments and scientific visualization '96*, 1996, pp. 137 –148.

[144] I. Marsic, "DISCIPLE: a framework for multimodal collaboration in heterogeneous environments", *ACM Computing Surveys*, vol. 31, no. 2, p. 6, Jun. 1999.

[145] C. Carlsson and O. Hagsand, "DIVE A multi-user virtual reality system", in *Proceedings of IEEE Virtual Reality Annual International Symposium*, IEEE, 1993, pp. 394–400.

[146] J. Ahn, M. Lee, and H. Lee, "DOOViE: an architecture for networked virtual environment systems", in *Proceedings Computer Graphics International*, IEEE Comput. Soc. Press, 1997, pp. 113–119.

[147] M. Repplinger and D. Rubinstein, "DRONE: a flexible framework for distributed rendering and display", *Advances in Visual Computing*, vol. 5875, pp. 975–986, 2009.

[148] D. Lake, M. Bowman, and H. Liu, "Distributed scene graph to enable thousands of interacting users in a virtual environment", *2010 9th Annual Workshop on Network and Systems Support for Games*, pp. 1–6, Nov. 2010.

[149] C. Greenhalgh, "EQUIP: a software platform for distributed interactive systems", *The Equator Project, University of Nottingham,*, 2002.

[150] E. Berglund and H. Eriksson, "Distributed interactive simulation for group-distance exercises on the web", in *Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation*, 1998, pp. 91–95.

[151] J. Allard, V. Gouranton, L. Lecointre, and S. Limet, "FlowVR: a middleware for large scale virtual reality applications", *Euro-Par 2004 Parallel Processing*, vol. 3149, pp. 497–505, 2004.

[152] T. Arcila, J Allard, and E Boyer, "Flowvr: A framework for distributed virtual reality applications", in *Lière journées de lâ€™™association Française de réalité virtuelle, augmentée, mixte et dâ€™™interaction 3D*, Rocquencourt, France, 2006.

[153] S. Izadi, P. Coutinho, T. Rodden, and G. Smith, "The FUSE platform: supporting ubiquitous collaboration within diverse mobile environments", *Automated Software Engineering*, vol. 9, no. 2, pp. 167–186, 2002.

[154] J. Mandeville, T Furness, M Kawahata, and D Campbell, "Greenspace: Creating a distributed virtual environment for global applications", in *IEEE Proceedings of the Networked Reality Workshop*, 1995, p. 11.

[155] S Symington and K. Morse, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules (IEEE Std 1516-2000)", *IEEE Computer Society*, 2000.

[156] K. Hosoya, A. Kawanobe, and S. Kakuta, "Interactive cooperative learning system based on virtual shared space: HyCLASS", *CSCL '97 Proceedings of the 2nd international conference on Computer support for collaborative learning*, no. 2, pp. 106–113, 1997.

[157] L. Chan, J. Yong, J. Bai, B. Leong, and R. Tan, "Hydra", in *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07*, New York, New York, USA: ACM Press, 2007, pp. 37–42.

[158] B. Johanson, A. Fox, and T. Winograd, "The Interactive Workspaces project: experiences with ubiquitous computing rooms", *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 67–74, Apr. 2002.

[159] K. Raaen, H. Espeland, H. K. Stensland, A. Petlund, P. Halvorsen, and C. Griwodz, "A demonstration of a lockless, relaxed atomicity state parallel game server (LEARS)", in *2011 10th Annual Workshop on Network and Systems Support for Games*, IEEE, Oct. 2011, pp. 1–3.

[160] L. Zhang and Q. Lin, "MACVE: A Mobile Agent Based Framework for Large-Scale Collaborative Virtual Environments", *Presence: Teleoperators and Virtual Environments*, vol. 16, no. 3, pp. 279–292, Jun. 2007.

[161] P. Fuhrer, G. Kouadri Mostefaoui, and J. Pasquier-Rocha, "MaDViWorld: a software framework for massively distributed virtual worlds", *Software: Practice and Experience*, vol. 32, no. 7, pp. 645–668, Jun. 2002.

[162] C. Greenhalgh, J. Purbrick, and D. Snowdon, "Inside MASSIVE-3: flexible support for data consistency and world structuring", in *Proceedings of the third international conference on Collaborative virtual environments*, ACM, 2000, pp. 119–127.

[163] R. K. Balan, M. Ebling, P. Castro, and A. Misra, "Matrix : Adaptive Middleware for Distributed Multiplayer Games", in *Middleware 2005*, 2005, pp. 392–402.

[164] R. Hubbold, J. Cook, M. Keates, S. Gibson, T. Howard, A. Murta, A. West, and S. Pettifer, "GNU/-MAVERIK", in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '99*, New York, New York, USA: ACM Press, 1999, pp. 66–73.

[165] A. Bharambe and S Rao, "Mercury: a scalable publish-subscribe system for internet games", *and system support for games*, 2002.

[166] N. Santos and P. Ferreira, "Vector-Field Consistency for Ad-hoc Gaming", in *Middleware*, 2007, pp. 80–100.

[167] A. P. Yu and S. T. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games", *Proceedings of the international workshop on*, pp. 99–104, 2005.

[168] M. Román and R. H. Campbell, "A Model for Ubiquitous Applications", University of Illinois at Urbana-Champaign, Champaign, IL, USA, Tech. Rep., 2001.

[169] J. Chen and B. Loffin, "MUVEES: a PC-based multi-user virtual environment for learning", *IEEE Virtual Reality, 2003. Proceedings.*, vol. 2003, pp. 163–170, 2003.

[170] T. K. Das, G. Singh, A. Mitchell, P. S. Kumar, and K. McGee, "NetEffect", in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '97*, New York, New York, USA: ACM Press, 1997, pp. 157–163.

[171] M Capps and D McGregor, "NPSNET-V. A new beginning for dynamically extensible virtual environments", *Computer Graphics and*, no. December 1999, 2000.

[172] P. Hartling, C. Just, and C. Cruz-Neira, "Distributed virtual reality using Octopus", in *Proceedings IEEE Virtual Reality 2001*, IEEE Comput. Soc, 2001, pp. 53–60.

[173] P. Okanda and G. Blair, "OpenPING", in *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops*, New York, New York, USA: ACM Press, 2004, p. 111.

[174] R. B. Loftin, B. M. Pettitt, S. Su, C. Chuter, J. A. McCammon, C. Dede, B. Bannon, and K Ash, "PaulingWorld: An Immersive Environment for Collaborative Exploration of Molecular Structures and Interactions", in *NORDUnetâ€™98, 17th Nordic Internet Conference*, 1998.

[175] M. Capps, "The QUICK framework for task-specific asset prioritization in distributed virtual environments", in *Proceedings IEEE Virtual Reality 2000 (Cat. No.00CB37048)*, IEEE Comput. Soc, 2000, pp. 143–150.

[176] I. J. Grimstead, N. J. Avis, and D. W. Walker, "RAVE: the resource-aware visualization environment", *Concurrency and Computation: Practice and Experience*, vol. 21, no. 4, pp. 415–448, Mar. 2009.

[177] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel, "Reality built for two: a virtual reality tool", in *ACM SIGGRAPH Computer Graphics*, vol. 24, ACM, 1990, pp. 35–36.

[178] B MacIntyre and S Feiner, "A distributed 3D graphics library", *annual conference on Computer graphics*, pp. 361–370, 1998.

[179] T. A. Funkhouser, "RING: a client-server system for multi-user virtual environments", in *Proceedings of the 1995 symposium on Interactive 3D graphics - SI3D '95*, New York, New York, USA: ACM Press, 1995, 85–ff.

[180] C. Codella, P. Sweeney, G. Turk, R. Jalili, L. Koved, J. B. Lewis, D. T. Ling, J. S. Lipscomb, D. A. Rabenhorst, C. P. Wang, and A. Norton, "Interactive simulation in a multi-person virtual world", in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, New York, New York, USA: ACM Press, 1992, pp. 329–334.

[181] H. Hua, L. D. Brown, C. Gao, and N. Ahuja, "A new collaborative infrastructure: SCAPE", in *IEEE Virtual Reality, 2003. Proceedings.*, vol. 3, IEEE Comput. Soc, 2003, pp. 171–179.

[182] N. Gupta, A. J. Demers, and J. E. Gehrke, "SEMMO", in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, New York, New York, USA: ACM Press, 2008, pp. 1235–1238.

[183] W. Broll, "SmallTool - a toolkit for realizing shared virtual environments on the Internet", *Distributed Systems Engineering*, vol. 5, no. 3, pp. 118–128, Sep. 1998.

[184] W. Wang, Q. Lin, J. M. Ng, and C. P. Low, "SmartCU3D: a collaborative virtual environment system with behavior based interaction management", in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '01*, New York, New York, USA: ACM Press, 2001, p. 25.

[185] S. Benford, "A Spatial Model of Interaction in Large Virtual Environments", in *ECSCW'93 Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers Norwell, MA, USA Â©1993, 1993, pp. 109–124.

[186] Z. Szalavari, D. Schmalstieg, A. Fuhrmann, and M. Gervautz, "Studierstube: An environment for collaboration in augmented reality", *Virtual Reality*, vol. 3, no. 1, pp. 37–48, Mar. 1998.

[187] H. Abrams, "Extensible interest management for scalable persistent distributed virtual environments", PhD thesis, Naval Postgraduate School, 1999, p. 237.

[188] M. Macedonia and S. Noll, "A transatlantic research and development environment [3D virtual graphics]", *IEEE Computer Graphics and Applications*, vol. 17, no. 2, pp. 76–82, 1997.

[189] A. Basu, A. Raij, and K. Johnsen, "Ubiquitous collaborative activity virtual environments", in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, New York, New York, USA: ACM Press, 2012, pp. 647–650.

[190] Y Fabre, G Pitel, L Soubrevilla, and E Marchand, "An asynchronous architecture to manage communication, display, and user interaction in distributed virtual environments", in *6th Eurographics Workshop on Virtual Environments (EGVE'2000)*, Amsterdam, The Netherlands: Springer-Verlag New York, Inc., 2000, pp. 105–113.

[191] H. Backhaus and S. Krause, "Voronoi-based adaptive scalable transfer revisited", in *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07*, New York, New York, USA: ACM Press, 2007, pp. 49–54.

[192] J. C. D. Oliveira, "VELVET: An Adaptive Hybrid Architecture for Very Large Virtual Environments", *Presence: Teleoperators*, vol. 2000, no. 4, pp. 288–580, Dec. 2003.

[193] W Bricken, "The VEOS Project (Virtual Environment Operating Shell)", *Presence-Teleoperators and Virtual*, vol. 3, no. 2, pp. 111–129, 1994.

[194] B. Blau, C. E. Hughes, M. J. Moshell, and C. Lisle, "Networked virtual environments", *Proceedings of the 1992 symposium on Interactive 3D graphics - SI3D '92*, pp. 157–160, 1992.

[195] M. P. Dye, "Vesuvius : Interactive Atmospheric Visualization in a Virtual Environment", PhD thesis, University of Nevada, 2007.

[196] Y. Honda, K. Matsuda, J. Rekimoto, and R. Lea, "Virtual Society", in *Proceedings of the first symposium on Virtual reality modeling language - VRML '95*, New York, New York, USA: ACM Press, 1995, pp. 109–116.

[197] K Matsuda and Y Honda, "Virtual society: Multi-user interactive shared space on WWW", *of the 6th International Conference on Artificial Reality and Telexistance*, pp. 83–95, 1996.

[198] W. Lamotte, E. Flerackers, F. Van Reeth, R. Earnshaw, and J. Mena De Matos, "Visinet: collaborative 3D visualization and VR over ATM networks", *IEEE Computer Graphics and Applications*, vol. 17, no. 2, pp. 66–75, 1997.

[199] J. Ohya, Y. Kitamura, and H. Takemura, "Real-time reproduction of 3D human images in virtual space teleconferencing", *Virtual Reality Annual International Symposium*, pp. 408 –414, 1993.

[200] I. S. Pandzic, T. K. Capin, N. Magnenat-Thalmann, and D. Thalmann, "VLNET: A Networked Multimedia 3D Environment with Virtual Humans", in *Multi-Media Modeling*, World Scientific Press, 1995, pp. 21–32.

[201] C. Joslin, T. Molet, N. Thalmann, J. Esmerado, D. Thalmann, I. Palmer, N. Chilton, and R. Earnshaw, "Sharing attractions on the Net with VPark", *IEEE Computer Graphics and Applications*, vol. 21, no. 1, pp. 61–71, 2001.

[202] R. Kazman, "Making WAVES: On the design of architectures for low-end distributed virtual environments", *Virtual Reality Annual International Symposium,*, 1993.

[203] K. Gorman, D. Singley, and Y. Motai, "Synchronous collaborative systems for distributed virtual environments in Java", *International Journal of Computer Applications in Technology*, vol. 29, no. 1, p. 27, 2007.

[204] F. Chardavoine, S. Ageneau, and B. Ozell, "Wolverine: A Distributed Scene-Graph Library", *Presence: Teleoperators and Virtual Environments*, vol. 14, no. 1, pp. 20–30, Feb. 2005.

[205] N Magnenat-Thalmann, N. Nijdam, and S. Han, "InterMedia: Towards Truly User-Centric Convergence of Multimedia", *User Centric Media*, vol. 40, pp. 3–10, 2010.

[206] X Righetti, "A Modular Approach towards Wearable Computing", 2010.