A Supply-Chain for Computer-Mediated Communication and Visualization

Nils Jensen
University of Hannover
L3S / RRZN / DVR
German Pavilion, Expo Plaza 1
30539, Hanover, Germany
jensen@learninglab.de

Ralf Einhorn
University of Hannover
L3S / RRZN / DVR
Schloßwender Str. 5
30159, Hanover, Germany
einhorn@learninglab.de

Gabriele von Voigt
University of Hannover
RRZN / DVR
Schloßwender Str. 5
30159, Hanover, Germany
vonvoigt@rrzn.uni-hannover.de

ABSTRACT

The paper specifies modular software for synchronous and asynchronous computer-mediated communication and visualization in network-distributed environments. It is new because the distribution of the visualization pipeline is not prescribed, compared to other systems, and supports time-deferred collaboration and presentation by means of recordable sessions of use. Data source is a remote program that sends and receives data via a small linked library. The system supports real-time and time-deferred collaborative visualization over distance for e-Science.

Keywords

Visualization GRID, Visualization Pipeline, Software Engineering, CSCV, Virtual Environment, Virtual Reality

1. INTRODUCTION

A supply chain is a network of distributed facilities for the procurement of material, transformation to products, and delivery to customers [Gan03a]. The visualization GRID is an example that has lagged behind the development of the parts. We report a more tightly-structured combination of tools that (i) helps scientists and engineers to experiment with computer-simulated phenomena in distributed virtual environments (DVE), (ii) amplifies students' skills to work together via computer-mediated communication (CMC), and (iii) helps evaluators to study decentralized work that involves Virtual Reality (VR) [Lei97a]. The technical foundation is visualization software, databases, multimedia devices, and groupware. To structure and control the components in a centralized way, we use the supply-chain management (SCM) design pattern to combine application steering and visualization, synchronize the interplay between components, add verbal- and non-verbal communication, and transcribe logs and media streams for the recapitulation of sessions. The system is a modular implementation of the visualization pipeline (server), and a front-end for computer-supported collaborative visualization (CSCV client) in DVEs and browsers. The server receives visualized program results and the clients receive visualizations and multimedia data from videoconferencing units to augment a DVE. Sections 2, 3 and 4 specify related work, design (Fig. 1), and conclusion, respectively.

WSCG POSTERS proceedings
WSCG'2004, February 2-6, 200 , Plzen, Czech Republic.

2. OPTIONS FOR CSCV

The lowest layer comprises [Kau90a, p. 5] data generator, filter, map, render, and display. One partitions the pipeline to use large datasets [Lei97a] and help users to work together [Sin99a]. Partitioning techniques can be combined.

The first way is to separate data generation from filtering. The advantage is data are visualized remotely at the same time by use of different techniques. The disadvantage is network saturation.

The second way is to split filtering and mapping. The advantage is a reduction of replicated data because filtering selects subsets. The problem is that techniques to map data to *visualization objects* (**VOs**) use different filter results. We split them in those that depend on data, and those that depend on VOs.

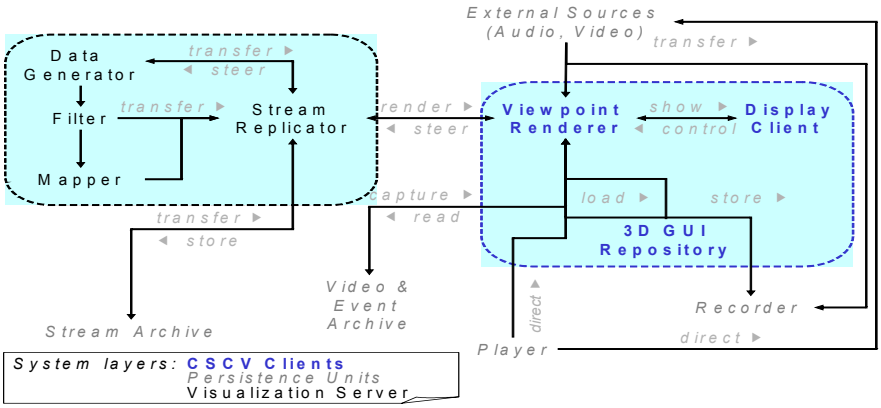

Figure 1. CSCV architecture.

Way three is to use "fat clients" which separate mapping and rendering. The advantage is that most visualization parameters are unbound until the client renders data. Fat clients must be fast enough to render complex scenes, and the data format for data transfer is complex, see [Hos02a, Jen02a].

Way four are "thin clients" that separate display from rendering for lightweight processors and periphery with limited ways of output and use. The scheme saves processor time because the front-end replicates images. But it is restricted to one viewpoint.

The middle layer manages persistent sessions. The synchronization of media helps one to reconstruct the interplay between users, data, and software. But synchronization for persistence is hard, which is reflected by the complexity of MPEG-4 and SMIL that specify the interplay and accurate timing of media streams in different ways [Hos02a, W3C01a]. But tools to create and author SMIL presentations do not integrate all formats, for example 3D graphics.

The highest layer splits the visualization pipeline in one stream per client. Modular, extensible GUIs combine heterogeneous content (live 3D animations and video) through DVEs, visualization GUIs, and VRML-browsers. [Jen02a; Jen03a] survey software.

In the paper, we decide to improve our system, DSVR [Jen02a], because it has a library for data generators to filter and map data to VOs. The server replicates and forwards VOs to fat clients which support CSCV and capture content. But limitations are (i) no methods for generator control, (ii) hard-coded coupling between generation, filtering and mapping, (iii) lack of extensible user interfaces, (iv) no persistent events, and (v) no support for thin clients. We specify the removal of the limitations.

3. DESIGNING THE SUPPLY-CHAIN

Filter Developers of data generators enhance the source code to derive visualizations and to control software remotely. Systems that visualize data from applications without changing source code [Ger01a] exist, but the flow of data between generator and pipeline is too coarse for parallel execution because command line parameters, text files, and Unix pipes are used. The alternative is to select memory to share data with a linked library that mediates the flow of visualization and steering data. We minimize manual changes by way of linking to a library that does not introduce new data types but that gives developers control over which data are shared. The library must read a specification of the source type from the generator, and read the destination type of data during run-time. To reduce programming effort, the developer specifies information by the use of meta-tags that are preprocessed to insert library calls in the source code in an automatic way. It is important to keep in mind that, for a flexible approach, tags must not prescribe which visualizations are available to users. The developer must specify visualization-dependent filters and mappings in files *outside* the source code. The library reads them during runtime.

We give an example in C. Comments with the prefix "///" are directives for our preprocessor. The example specifies a compound data structure that contains text and variables to represent simulated measurements. Tags describe how shared language-specific data types map to library-specific types (SCALAR…):

```
struct ecomposite { /// DEF ecomposite
   float size;      /// SCALARF size
   char *s;         /// SCALARB s SIZE
   int   slen;      /// SIZE s
   char name[30];   /// SCALARB name
SIZE 30
} aComposite[2]; /// ENDDEF ecomposite
/// STRUCT ecomposite aComposite SIZE 2
double magnitude[2];
/// SCALAR magnitude SIZE 2
double temperature[2];
/// SCALAR temperature SIZE 2
vector3D * coord[2];
/// STRUCT VECTOR3D SIZE coord SIZE 2
double volume[32][32][16];
/// SCALAR magnitude SIZE 32 SIZE 32
SIZE 16 ... /// VISUALIZE
```

Our preprocessor replaces tagged parts with data structures in C and with library calls to map data to VOs. The code is ready to compile and link (part):

```
main() {
initializeVisualizationLibrary();
useRoute("default.rou");
share(aComposite, ID_aComposite,
TYPE_STRUCT_ECOMPOSITE, 2);
share(magnitude, ID_magnitude,
TYPE_SCALAR, 2);
share(temperature, ID_temperature,
TYPE_SCALAR, 2);
share(coord, ID_coord,
TYPE_STRUCT_VECTOR3D, 2, 1);
share(volume, ID_volume, TYPE_SCALAR,
32 * 32 * 16);
/* VISUALIZE */ commitState();
```

The example initializes the library, sets up default routings for data replication, and specifies which data are read during calls to "commitState()" to generate VOs from application data. VOs are runs of VRML-1 objects in binary form that constitute a frame in a 3D stream [Jen02a]. The frame is committed and sent to remote servers and clients in "commitState()":

```
on each processor:
forward content of shared memory
for each object set 's' {
  for each object 'o' in 's' {
    for each attribute 'a' of 'o' {
      read shared memory location
      and choose all valid indices
      'i,j,k,...' that satisfy the
      filter expression
      cast and write data to 'a'
}}} submit and forward objects
```

Map Objects and attributes are specified in a file that contains names and links to data. Formal expressions select data (Tab. 1). We restrict expressions to properties of integers to denote array slots to support common selection criteria. A second file defines networked data flow. We try to balance between decoupling and performance.

Element	Meaning
*	Put the array in one VO
m..n	Put m-n+1 data in one VO each
Le	Is less than
Ge	Is greater than
Eq	Is equal to
Leq	Is less than or equal to
Geq	Is greater than or equal to
And	Logical and
Or	Logical or
Not	Logical not
Div	Divisible with remainder 0

Table 1. Data selection keywords to generate VOs.

The file for filtering and mapping has the following structure (we have chosen XML). Single letters denote indices. Object sets are selected by name, the special keyword "rawData" (tagged memory content), or a compound formal expression that uses "and" and "not". The example creates a 3D annotation at a default position, two streamlines, and an iso-surface. The volume for the surface is sub-sampled at level 2.

```
<!—visualization rules default.fil -->
<text>
<fontName> aComposite.name <filter/> i
eq 0 and j eq * </fontName>
<size> aComposite.size <filter/> 0
</size>
<string> aComposite.s <filter/> i eq 0
and j eq * </string>
<stringLength> aComposite.slen
<filter/> 0 </stringLength>
</text> <streamline>
<headSize> magnitude <filter/> 0..*
</headSize>
<color> temperature <filter/> 0..*
</color>
<tailPathX> coord <filter/> i eq 0..*
  and j eq 0 </tailPathX>
<tailPathY> coord <filter/> i eq 0..*
  and j eq 1 </tailPathY>
<tailPathZ> coord <filter/> i eq 0..*
  and j eq 2 </tailPathZ>
</streamline> <isosurface>
<regularField> volume <filter/> i div 2
and j div 2 and k div 2 </regularField>
</isosurface>
```

The routing file specifies "commitState()"sends geometries to host "3dserver", sends the iso-surface to "hapticserver", and saves data on "simarchive".

```
<!-- routing default.rou -->
<forward><from/> not rawData <to/>
3dserver.example.edu</forward>
<forward><from/> isosurface <to/>
hapticserver.example.edu </forward>
<forward><from/> rawData <to/>
simarchive.example.edu </forward>
```

Render The plugin renders 3D graphics in browsers and DVEs [Jen02a; Jen03a]. The latter uses an adapter to mediate between the DVE and the plugin's NAPI interface. Screen grabbing and logging are available, and we use RealVNC (www.realvnc.com) to share displays on other computers, e. g. handhelds.

Record and Archive VACE is a modular, extensible, and distributed toolkit to record media. Users record sessions through media formats. To record and play-out sessions in DSVR we combine it with VACE. We generate VACE-compatible metadata and synchronize it with recorded media data. We build an adapter to combine VACE and DSVR, or make DSVR read metadata, which requires changes to DSVR.

To find the required granularity of metadata we consult the VACE data model [Ein03a] that comprises hierarchical layer presentation, stream and

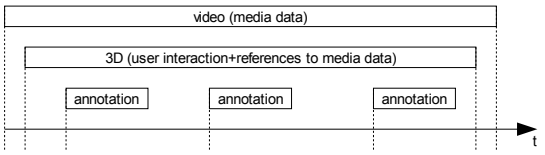

Figure 2. Media data types in VACE streams.

event. A stream is a medium that is shown during session replay. Streams are "temporal discrete" (e. g. text) or "temporal continuous" media (e. g. audio/video (AV) and graphics). A stream carries events to denote changes of the media stream, e. g. a new Web page or the start of a 3D stream. The question is if users' action in DSVR is "temporal discrete" (every mouse movement represents a VACE event) or "temporal continuous" (action must be stored in a separate way), so that only synchronization points are events (Fig. 2). We use the latter because the first way would generate many events. Hence, DSVR manages events and synchronizes other media during play-out via timestamps. Temporal positions of the SMIL player and DSVR must be coherent during playback, especially at start. In contrast to conventional multimedia applications, the same system is used for recording and play-out, and there is one recorded media type for DSVR. We manage interaction data as a container format for event and media data, like an HTML image reference.

Replay VACE uses tools for the play-out of content. A container format for multimedia presentations is SMIL [W3C01a]. Available applications for SMIL playback (e. g. RealPlayer) support some media types natively, and others are integrated by use of HTML files controlled by the player. The challenge is to synchronize between DSVR and media types that are integrated in the SMIL player. RealOne integrates plugins for supporting proprietary media formats, so we could add RealOne's plugin API to the DSVR front-end plugin. But the SMIL player may not use OpenGL-compatible drawing contexts and requires changes to DSVR. We prefer to control DSVR by use of synchronization points. When the user skips through the timeline using the SMIL player, the player sends timestamps to DSVR to reconstruct system states.

The SMIL player and DSVR exchange messages via the JavaScript engine of the Web browser. Measurements show synchronization granularity rate is ca. 0.1 seconds. The mechanism is sufficient for applications with "soft" real-time requirements. The next step to improve granularity of the synchronization is to implement a SMIL player plugin that mediates between the player and DSVR.

4. CONCLUSION

We have specified a modular visualization pipeline with full support for CSCV. See www.learninglab.de/vase3 for a case study.

5. ACKNOWLEDGMENTS

The Ministry for Science, Research, and Art of Lower Saxony and the German Ministry for Education and Research funded VASE 3 and VACE.

6. REFERENCES

[Ein03a] Einhorn, R., Olbrich, S. and Nejdl, W. A metadata model for capturing presentations in ICALT, Athens, Greece, IEEE CS Press, pp. 110-114, 2003

[Gan03a] Ganeshan, R. and Harrison, T.P. An introduction to supply chain management, http://silmaril.smeal.psu.edu/misc/ supply_chain_intro.html (accessed on 29th Sep. 2003)

[Ger01a] Germans, D., Spoelder, H.J.W., Renambot, L. and Bal, H.E. VIRPI: A high-level toolkit for interactive scientific visualization in virtual reality in 7th EGVE, Stuttgart Germany, ACM Press, pp. 109-120, 2001

[Hos02a] Hosseini, M. and Georganas, N.D. MPEG-4 BIFS streaming of large virtual environments and their animation on the web in Web3D, Tempe Arizona, ACM Press, pp. 19-25, 2002

[Jen02a] Jensen, N., Olbrich, S., Pralle, H. and Raasch, S. An efficient system for collaboration in tele-immersive environments in 4th EGPGV, Blaubeuren Germany, ACM Press, pp. 123-131, 2002

[Jen03a] Jensen, N., Seipel, S., Nejdl, W. and Olbrich, S. CoVASE – Collaborative visualization for constructivist learning in CSCL '03, Bergen Norway, Kluwer, pp. 249-253, 2003

[Kau90a] Kaufman, A. Volume Visualization. IEEE CS Press. 1990

[Lei97a] Leigh, J., Johnson, A.E. and DeFanti, T.A. Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in collaborative virtual environments in Supercomputing '97, San Jose CA, IEEE CS Press, p. 1-14, 1997

[Sin99a] Singhal, S. and Zyda, M. Networked virtual environments - design and implementation. ACM Press. 1999

[W3C01a] W3C: Synchronized multimedia integration language (SMIL 2.0), http://www.w3.org/TR/2001/REC-smil20-20010807/ (accessed on 14th Oct. 2003)