Service-Oriented Information Visualization for Smart Environments

Conrad Thiede, Christian Tominski, Heidrun Schumann
Institute for Computer Science
University of Rostock
{*thiede,ct,schumann*}@*informatik.uni-rostock.de*

Abstract

Smart environments consist of several interconnected devices. The device ensemble can change dynamically as mobile devices enter or leave the environment.

To utilize such environments efficiently for information visualization, we propose a service-oriented architecture. Various services run on different machines and visualizations are generated dynamically depending on the environment's current situation. The necessary adaptation to available output devices is driven by instantiation of different service implementations, by parameterizing service invocations, and by adapting the visualization pipeline at run-time. We implemented a prototype that provides parallel coordinates, scatter plot matrices, and a map display.

Keywords— **Information Visualization, Service-Oriented Architecture, Smart Environments**

1 Introduction

Today's computing environments integrate a multitude of interconnected devices. In addition to classic computing and output devices, smart environments integrate sensor devices to monitor the environment and its inhabitants. These sensors plus appropriate analysis and prediction methods make a computing environment smart. A specific instantiation of a smart environment is the smart meeting room [3]. Its device ensemble consists of stationary devices such as desktop computers, projectors, lights, or motion trackers, and additional mobile devices such as laptops, PDAs, smart phones, but also mobile projectors, which may enter the ensemble as users carry them along. The latter devices form the ad-hoc character of the smart meeting room.

From the perspective of information visualization, smart ad-hoc environments pose an interesting research question: How can we make efficient use of computing devices and output devices available in a device ensemble that, however, is subject to ad-hoc changes? Classic information visualization provides data- or task-specific solutions that are computed on a single machine for a static output device. Approaches that utilize multiple loosely coupled devices to generate and present visualizations are scarce. It is not yet investigated well how the devices in smart ad-hoc environments can be utilized to better communicate information or how to harness more devices to communicate more information. New adaptation strategies are needed in smart ad-hoc environments to handle different output devices and to cope with the environment's ad-hoc character, i.e., to integrate into the overall visualization ensemble those devices that enter the environment, and detach the ones that leave. Moreover, the adaptation process needs to be smart, i.e., the need for user intervention should be reduced to a minimum.

This paper describes our ongoing work to develop a service-oriented architecture (SOA) that can be used as a generic basis for information visualization in smart ad-hoc environments. The architecture is a means to utilize distributed device ensembles and to address problems related to the ad-hoc character of smart meeting rooms. Instead of a hard-wired visualization pipeline, the SOA approach has been designed so as to instantiate the pipeline dynamically at runtime based on environment's current situation. We utilize Chi's data state reference model [4] as the starting point for our approach. The reference model describes a visualization pipeline as a sequence of data transformations, so called operators. Our idea is to map operators to separable services that may run on different machines and that are capable of generating visualizations adapted to different output devices.

In Section 2, we describe the basics of smart environments and service-oriented architectures (SOA) and take a look at related work. Section 3 presents our general approach, while specific implementation details are given in Section 4. We exemplify a usage scenario in Section 5, before closing with a summary and an outlook on future work in Section 6.

2 Basics & Related Work

We now briefly describe smart environments as the background of our research and introduce the basics of service-oriented architectures. Related work in distributed visualization is discussed in the third part of this section.

2.1 Smart Environments

Cook and Das [5] define a smart environment as *"one that is able to acquire and apply knowledge about an environment and also to adapt to its inhabitants in order to improve their experience in that environment."* Smart homes, smart class rooms, and smart meetings rooms are specific examples of smart environments [8, 1]. The device ensemble can be differentiated into stationary devices, which are firmly mounted, and mobile devices, which can enter or leave the environment. The smart meeting room that we address here is a mixed environment with a set of stationary devices, including computing devices (e.g., desktop computers, servers), output devices (e.g., projectors, canvasses, monitors, flat panels), environmental devices (e.g., lights, blinds, air conditioning), and sensor devices (e.g., motion trackers, infrared beacons, light sensors). Users of the smart meeting room can bring mobile devices like notebooks, mobile projectors, PDAs, and smart phones, which make up the dynamic ad-hoc character. In most cases, mobile devices act as output devices, but they can also be utilized to accomplish computing tasks.

The devices form a loosely coupled network, which allows for the necessary communication to accomplish tasks in a coordinated fashion. The network is driven by various technologies, including Bluetooth, wireless LAN, or hard-wired LAN. Here, we abstract from the underlying technology and assume that proper communication channels are available.

A smart environment also implements software that is responsible for providing "smart" support to users. It is an actively investigated research question how to accomplish "smart" support in a joint effort of several distributed software components. The task is to constantly assess the current situation of the environment and that of its inhabitants. Based on an analysis step, user intentions and tasks are predicted [13]. Preferably, the predictions are as accurate as to allow for fully automatic user support. For instance, if the user leaves his PDA on the desk and moves toward a large canvas, the output is automatically routed to the corresponding projector. In cases where predictions cannot be as accurate, users always have the possibility to revise the decisions made by the environment or to fine-tune the environment to their needs [12].

That said, our goal is to utilize device ensembles, communication infrastructure, and predictive software components to drive information visualization in smart meeting rooms. However, classic approaches do not fit well in this scheme of distributed architectures. It is necessary to adapt the visualization process to the requirements and constraints of the smart meeting room. We strive to achieve this by developing a generic service-oriented architecture for information visualization in smart environments.

2.2 Service-Oriented Architectures

A service-oriented architecture (SOA) is based on a set of software components that are loosely coupled [6]. This makes SOA a perfect match for implementing information visualization in the smart meeting room.

A service encapsulates a specific functionality and provides it to users through a well-defined interface [11]. At a high level of abstraction, SOA consist of three basic components [14, 20]: service providers, service consumers, and service brokers (or registries). Service providers make different services available by registering them at a service broker. Service consumers requests services from the broker to access specific functionality. To accomplish higher order tasks, services can be combined. Yang and Papazoglou differentiate between three different strategies [24]: Fixed composition provides static pre-defined service combinations; semi-fixed composition strategies allow for dynamic adaptations; explorative composition strategies generate service combinations dynamically at run-time.

SOA are advantageous with regard to several aspects. (1) Flexibility: A SOA makes it easier to adapt software to changing requirements. (2) Reduction of complexity: Services provide just basic functionality, but by combining services, higher order tasks can be accomplished. (3) Reusability: Services can be reused in different application contexts. (4) Compatibility: A well-defined interface warrants compatibility between different implementations of services. All these advantages bear relevance to adaptable information visualization in smart environments.

One can consider it a disadvantage that stricter rules must be obeyed during the development of services. However, these initial higher costs pay off later in the software life cycle. Another critical aspect are communication costs, which are particularly relevant for information visualization where larger volumes of data need to be transferred.

2.3 Distributed Visualization

Distributed visualization has been applied in several contexts [2]. Service-oriented architectures as well as agent-based models (ABM) are often used as the technical basis. Compared to services, whose operation relies on a higher-level control mechanism, agents operate autonomously to accomplish tasks. Despite this difference, there is much overlap between ABM and SOA. Next we review related approaches that utilize services or agents for visualization.

Automatic balancing of rendering workloads in collaborative visualization environments is presented in [9]. Available resources are allocated automatically, which enables visualization clients to request services regardless of the particular system implementation. Data services and renderer services are the only components provided by this approach.

A multi-level approach to service-based interactive visualization is described in [26]. All necessary computations are performed by services, which include data transfer services, filter services, mapping services, and render services. The approach allows for interactive visualization and multiple outputs. However, only fixed compositions of services with fixed parameterizations are possible.

MUVA is a flexible visualization architecture for multiple client platforms [19]. It is organized into four substructures. (1) Visualization tools generate visual mappings of some input data. (2) Platform drivers are responsible for the adaptive rendering on different platforms. (3) Application interfaces are used for input and output. (4) The service logic is the control and management component. It handles client requests, data transfer, access to visualization tools, and distribution of mapping results to platform drivers. Although MUVA is able to provide adaptive rendering, neither adaptations in the data space (e.g., clustering, filtering, or mapping) nor interaction are considered.

Zhao et al. developed a collaborative visualization approach based on web services [25]. Visualization operators are encapsulated as services, which are hosted on a central visualization server. This server creates visualizations and transfers them in form of bitmaps to the clients. Even though the visualization pipeline is separated into different services, distribution of these services to better use available resources in not considered.

Hagen et al. [10] apply agents to construct a three-tier multi-agent scientific visualization system. The kernel layer implements core visualization functionality. The extension layer provides the necessary tools for data im-/export, geometry generation, and visual mapping. The hardware layer encapsulates device-specific rendering functions. This architecture allows for adapted and distributed rendering, but does not considerer distribution of other visualization stages (e.g., clustering or visual mapping).

AVAM is an agent-oriented model for adaptive visualization over the internet [22]. It has been designed so as to adapt to the changing resources in internet scenarios. The main components are (1) a sensor component, which monitors available resources, (2) an arbitrator, which decides if and how agents are distributed depending on a cost-benefit analysis, and (3) the visualization agent, which realizes the commands issued by the arbitrator by calling visualization modules. This approach focusses on distribution of rendering calls and does not consider other data transformations.

A distributed information visualization approach that supports all steps of the visualization pipeline is presented in [18]. Reactive and deliberative agents are distinguished. Reactive agents have only little knowledge of the environment and perform low-level control tasks, whereas deliberative agents have a broader view on the environment, which enables them to accomplish more complex high-level tasks. Filtering and rendering are implemented as reactive agents and mapping is realized as deliberative agents, which choose automatically an appropriate visualization technique.

All reviewed approaches have in common that they implement loosely coupled software components for visualization. Many approaches focus on a specific aspect only. Balancing the workload, distributing the rendering step, or providing collaboration facilities are major objectives. We found no approach in the literature that targets the specific requirements of smart ad-hoc environments. Therefore, our goal is to make use of selected aspects of previous approaches and adapt them to the smart meeting room.

3 Visualization in the Smart Meeting Room

In this section, we first discuss requirements of the smart meeting room scenario and then describe on an abstract level our service-oriented architecture.

3.1 Requirements

The heterogeneous and ad-hoc character of the device ensemble is a core challenge: one can neither know of the involved devices nor can one depend on the availability of specific functionality. On the other hand, we want to take advantage of the devices that are currently available in the smart meeting room. We aim for distributing computations and for utilizing multiple display devices.

Relying on hard-wired implementations of the visualization pipeline is therefore not possible. Instead, the visualization pipeline must be composed from basic building blocks and must be adapted to the current situation of the environment at run-time. The adaptation has to take into account the availability and the characteristics of devices (e.g., computing power, memory, display size, resolution, color depth). Complex computations should preferably be performed on capable devices. Avoiding visual clutter (see [7]) is a particular requirement for smaller mobile devices. Strategies are needed to avoid mapping more information to an output device than can be displayed, but still the core message of the visualization must be preserved.

Although fully automatic support is desirable, users must always be able to control in how far their devices contribute to the environment. It is also a necessity to allow users to override system decision by offering methods for manual interactive adjustments.

On an abstract level, one can summarize these requirements as follows: (1) Composition of visualizations from basic building blocks in a distributed ad-hoc fashion, (2) adaptation of visualizations to the available computing and output devices, and (3) integration of appropriate user control mechanisms.

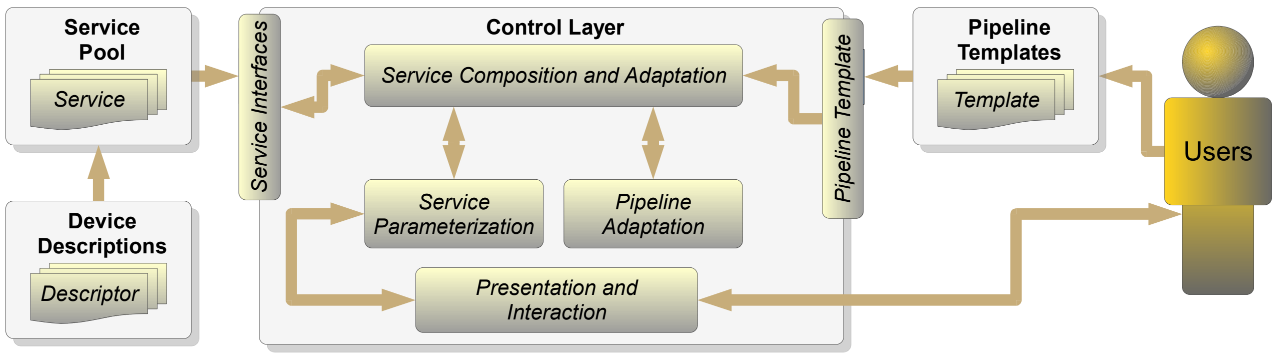


Figure 1: Service-oriented architecture for information visualization in the smart meeting room.

3.2 The SOA Model

We decided to use services as building blocks in favor of agents for the following reason. As the application context may change in the smart meeting room (e.g., one day the room is used for a technical presentation, the other day for collaborative exploration of statistical data), the particular requirements with regard to visualization also change. It is more suitable to build on top of a set of basic services a dedicated control layer that encapsulates several application-specific steering mechanisms, rather than integrating an overhead of control semantics to autonomous agents. With regard to the previously discussed requirements, we have to investigate three issues in detail:

1. What information visualization services are required in the smart meeting room?
2. How can services be linked and adapted dynamically at runtime to accomplish visualization tasks given the environment's current situation?
3. How are services invoked to create visual output?

We consider Chi's data state reference model (DSRM) [4] to identify required services. It is a well-accepted architecture for information visualization and models the visualization pipeline as a static network of data transformation operators. Our approach is to split up the hard-wired visualization pipeline and to implement operators as services that run on different devices. To facilitate service composition, it is necessary to abstract from concrete implementation details of services and to extract higher-level specifications. Therefore, we distinguish between *interfaces* and *implementations* of services. An interface $\mathcal{I} = \{I, O\}$ describes a service as a black-box with specified inputs $I = \{i_1, \ldots, i_n\}$ and outputs $O = \{o_1, \ldots, o_m\}$. Inputs and outputs include both data specifications and definitions of control parameters. Several implementations of an interface can exists, which then provide the same functionality, but differ in what control parameters they accept and how they perform the necessary computations.

The services are managed by a control layer whose major tasks are: (1) service registration, (2) service composition, parametrization, and adaptation, and (3) service invocation. Two mechanisms support the registration of services. First, by manually invoking a service it is registered at the control layer. Second, once started, a service reports itself to the control layer at fixed time intervals. This enables us to keep track of dynamic changes to the environment (e.g., a service ceases to exist). With the help of interface specifications, the control layer identifies those services that can be utilized to accomplish a particular visualization task.

The second task of the control layer is to dynamically link several services to form an adapted visualization pipeline at runtime. For that purpose, predefined problem-oriented *pipeline templates* are used. They describe an abstract visualization pipeline as a network of interconnected service interfaces (not specific implementations). At runtime, the control layer evaluates pipeline templates and binds specific device-oriented service implementations to create an executable visualization pipeline.

The necessary adaptation to the heterogeneous dynamic environment is realized in different conceptual ways: (1) providing different service implementations, (2) adjusting control parameters of services, and (3) by adapting the pipeline templates themselves. At the first level, if a device leaves the environment and with it a specific service implementation, the missing service is replaced with another implementation (of the same interface). At the second level, service parameters are set depending on characteristics reported by device description services. For instance, a device description service provides display resolution and color-depth as outputs O that can be connected to the input I of other services to control adaptation. In addition to automatic parameterization, users can manually tune service parameters. Thirdly, adaptation can be realized by automatically incorporating additional interfaces in a pipeline template at runtime. For instance, if a data set is too large to be presented on a smart phone, additional data abstraction steps are incorporated into the pipeline.

Finally, the control layer invokes the parameterized services in the order as described by the dynamically instantiated pipeline. This involves transferring data between successive services via network communication channels.

Figure 1 shows the main components of the service-oriented architecture as we are using it in the smart meeting room. In the next section, we will discuss in detail how specific aspects have been implemented and where future work is necessary to complete the architecture.

4 Implementation

4.1 Services

We coarsely categorized the operators of Chi's reference model based on the data transformation they perform. Interfaces have been inserted at those points in the visualization pipeline where major changes in the data structure are expected. In particular, the following basic interfaces are available: (1) data import, (2) data analysis, (3) mapping, (4) rendering, and an additional interface for (5) complementary services. This initial categorization works well with our proof-of-concept implementation. However, further investigation is necessary to adapt the interface design to different usage scenarios and to arrive at an appropriate level of granularity and a truly generic model.

A concrete implementation of a service interface is encapsulated as a Java program that can run as Web-Start application on any Java-ready device in the smart meeting room. The communication between services is based on Jini$^{\text{TM}}$. Currently, our prototype integrates the following services: *data import services:* CSV file import, SQL import; *data analysis services:* hierarchical clustering, filters; *mapping services:* color coding, parallel coordinates, scatter plot matrix, geographic map, halo highlighting [16]; *rendering services:* Java2D rendering, PDF output; *complementary services:* device description, data item identification, and clutter metrics.

4.2 Service Pool

While many of these services are running on the stationary devices of the smart meeting room, a user that enters the room can contribute the computing power of his mobile devices and can utilize it as display for visualization. For doing so, the user may invoke one or several Java Web-Start links to start services on his device. After that, the necessary registration at the service pool is done automatically. To address the ad-hoc environment (e.g., services on mobile devices may temporarily leave the environment and then re-enter it), services notify the control layer of their existence continuously at fixed time intervals.

4.3 Service Composition and Adaptation

To facilitate the dynamic construction of adapted visualizations at run-time, we introduced pipeline templates that connect service interfaces, rather than specific implementations. A pipeline template is based on an XML description (see Figure 2) that is evaluated by the control layer in order to test validity and to bind interface implementations at run-time. We make use of Java's Reflection API to identify candidates in the service pool that implement the interfaces used in a template.

Now the first way of adaptation comes into play: the device properties as reported by device description services (see [21]) are fetched to the candidates. Each candidate checks if it can perform the request successfully given the device characteristics. This enables us to provide dedicated service implementations for different devices, which in turn are automatically chosen during service composition. For instance, a customized parallel coordinates implementation for small displays will deny requests whose display resolution parameter exceeds a certain size.

As a second way for adaptation, a service implementation can be inherently adaptive, that is, it accepts as input parameter any device characteristics and internally adapts its operation accordingly. Our scatter plot service for instance automatically adjusts the number of bins, which are used to maintain an overview and to accelerate the rendering [17], to the target device resolution (see Figure 4(b)).

During service composition, the most simple case is that only a single service matches interface and device descriptions; it is bound immediately. In the case that no suitable service is found, a third way of adaptation is applied: by means of a rule-based mechanism, we incorporate additional service interfaces into the pipeline template at runtime. Adding a service for transforming the color space of a generated visualization to that of the target device is one example. We also implemented an automatic data abstraction rule that applies when the target device cannot handle large data sets. In that case, two additional service interfaces are inserted into the pipeline template: a data abstraction interface right after the data import and a clutter metric interface before the rendering. As implementation for the data abstraction and the clutter metric we use hierarchical clustering and data density (see [23]), respectively. If the clutter metric exceeds an empirically determined threshold, which depends on the visualization technique used, the visualization is not transmitted to the display, but instead the data abstraction is run to reduce the amount of data to be visualized. This also causes subsequent services to re-perform mapping and rendering and finally the clutter metric re-checks whether the abstraction is sufficient. This iterative loop finishes when the amount of clutter has been reduced to the desired level.

```
<template>
 <obj name="filtering" type="Data">
   <arg>count(diagnosis=influenza and
              year=1999)</arg>
 </obj>
 <obj name="colorParam" type="Color">
   <arg><readObj objName="filtering"/></arg>
 </obj>
 <obj name="deviceParam" type="Device">
   <arg>SEARCH</arg>
 </obj> ...
```

```
...
 <obj name="mapping" type="Mapping">
   <arg><readObj objName="filtering"/></arg>
   <arg><readObj objName="colorParam"/></arg>
   <arg><readObj objName="deviceParam"/></arg>
 </obj>
 <obj name="rendering" type="Renderer">
   <arg><readObj objName="deviceParam"/></arg>
   <arg><readObj objName="mapping"/></arg>
 </obj>
</template>
```

Figure 2: Sample pipeline template.

The result of the template evaluation is a set of parameterized services. The Java Reflection API is used to create referential links between input and output of the services as described by the pipeline template. This also includes linking to user interface services that provide GUI elements for interactive adjustments of parameters.

4.4 Service Invocation

Finally, a fully parameterized and interconnected network of services has been created that can be scheduled for execution. A service is invoked by the control layer once it has been successfully added to a visualization pipeline. The control layer then starts transferring input data to the services. To reduce data transmission costs, we use data compression methods (ZIP for plain data; PNG for visual content). If all necessary data are available at the input ports, a service performs its computation. After that, the control layer is notified to transfer the computed results to all services that depend on them. This procedure continues until all services have been executed, in particular those services that transmit visualizations to output devices.

In contrast to classic web-services, the data transformations that an information visualization service computes can be complex. Therefore, we implement a caching strategy to avoid unnecessary re-computations. The increased costs in terms of memory usage are acceptable since services run on different devices. Because one and the same service implementation can be part of several visualization pipelines, a session identifier is used to distinguish between cached content of several service invocations.

5 Usage Scenario

As a first usage scenario for service-oriented information visualization in our smart environment [3], we have chosen a meeting of physicians and statisticians and a data set with 10 dimensions and 1095 tuples. This scenario allows us to test and demonstrate our concepts for the ad-hoc generation and adaptation of visualizations.

Suppose a number of medical doctors and data analysis experts who meet to discuss the current health situation in their federal state. At a primary projector display, the waves of influenza of recent years are presented on a color coded map visualization. The experts discuss similarities of past years with the current situation to assess acute risks and necessary precautions. This is supported by automatic highlighting of important data portions (see Figure 3).

The visualization is distributed as follows: the laptop ARGON runs the data input service and the desktop computer NEON is responsible for mapping and rendering. A projector that is connected to NEON is used as output device.

While the map-based visualization of influenza is discussed, the question arises which other illnesses are closely related. This question is to be answered with a second visualization: a scatter plot matrix service is run on the machine XENON, which is connected to a second projector (see Figure 4(a)).

In order keep an overview, even when speaking to the expert group, the discussion leader decides to redirect a copy of the scatter plot matrix to his PDA (FERRUM). The low display resolution is recognized by the control layer and an adapted visualization is automatically generated for the PDA (see Figure 4(b)).

During the discussion, one expert has to leave the meeting unexpectedly. His laptop device ARGON is no longer available and the data import service gets disconnected. Another participant has a copy of the data on his machine and provides access to it by invoking a data import service. The control layer automatically integrates the newly available service from RADON into the running visualization environment, and the meeting can continue (see Figure 5).

6 Summary & Future Work

We presented preliminary results of our ongoing work on service-oriented information visualization in smart environments. The presented architecture allows us to make use of the available computing devices to generate visualizations in a distributed fashion. The ad-hoc character of the environment is dealt with by automatically managing a service pool. Vanishing service implementations are automatically substituted by other readily available ones. Our architecture also facilitates utilization of multiple display devices. To address the heterogeneous display characteris-

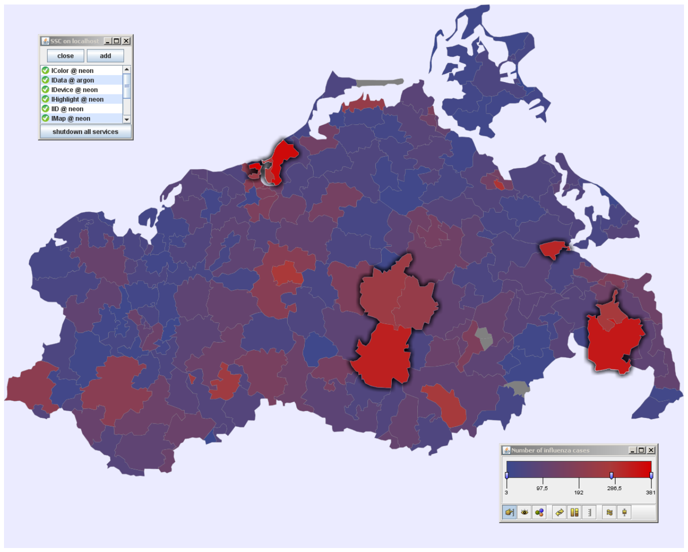
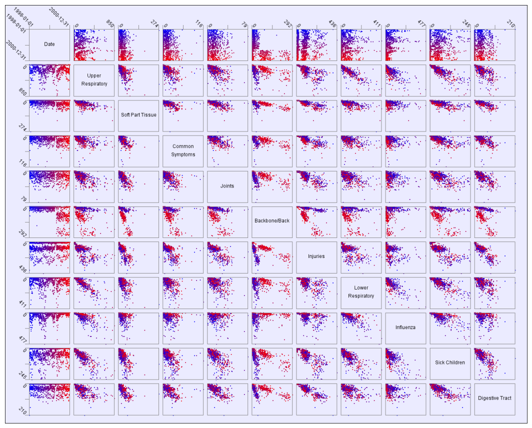
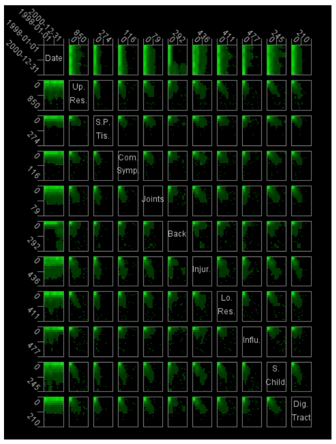


Figure 3: The occurrence of cases of influenza is color-coded on a map display.

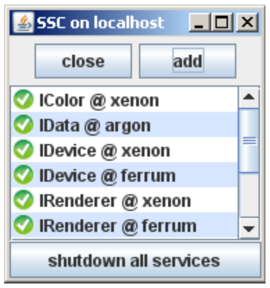


(a) On a full resolution projector display all information is shown in details.



(b) On a PDA, binning is used to reduce complexity.

Figure 4: To compare influenza to other diagnoses, a scatter plot matrix is utilized (rendering times between 0.5s and 3.5s).



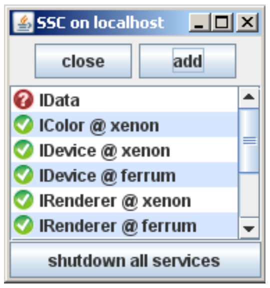
SSC on localhost

close add

- ✓ IColor @ xenon
- ✓ IData @ argon
- ✓ IDevice @ xenon
- ✓ IDevice @ ferrum
- ✓ IRenderer @ xenon
- ✓ IRenderer @ ferrum

shutdown all services

(a) List of services on different devices.



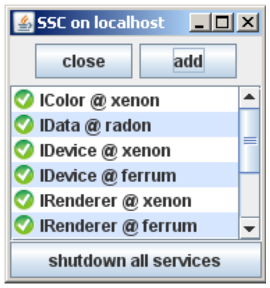
SSC on localhost

close add

- ? IData
- ✓ IColor @ xenon
- ✓ IDevice @ xenon
- ✓ IDevice @ ferrum
- ✓ IRenderer @ xenon
- ✓ IRenderer @ ferrum

shutdown all services

(b) The data service on ARGON is no longer available.



SSC on localhost

close add

- ✓ IColor @ xenon
- ✓ IData @ radon
- ✓ IDevice @ xenon
- ✓ IDevice @ ferrum
- ✓ IRenderer @ xenon
- ✓ IRenderer @ ferrum

shutdown all services

(c) A new data service on RADON is automatically integrated into the running pipeline.

Figure 5: Automatic replacement of missing services.

tics, we integrated adaptations strategies (e.g., abstraction-metric-loop or binned rendering). A critical point of the SOA architecture is the data transmission between services. We implemented caching strategies and apply compression methods where possible to reduce the data transmission costs.

One aspect for future works is linking our model with research on security issues in the smart meeting room [15]. We will also have to investigate in how far our model is sufficiently generic to allow its application in other smart environments. An important aspect that we have not considered in this work is collaborative interaction across visualizations displayed on multiple devices.

With regard to output adaptation, we envision several future ways to utilize multiple displays. If users work collaboratively, it makes sense to automatically present the same information on all available user devices simultaneously. To facilitate the visualization of larger data sets, one could split a visualization into several portions each of which being presented on a separate device. Another idea is to employ a details-on-demand strategy where one display shows an overview and selected details are shown on other displays. We also plan to implement advanced adaptation mechanisms that utilize the smart environment's sensory devices and situation analysis modules. If, for instance, two render services are available for two projector devices, it makes sense to bind the one that is closer to the user.

Acknowledgements

The research presented in this paper is conducted within the graduate school "MuSAMA – Multimodal Smart Appliance Ensembles for Mobile Applications", which is funded by the Deutsche Forschungsgemeinschaft (DFG).

References

[1] E. Aarts and J. Encarnação. *True Visions: The Emergence of Ambient Intelligence*. Springer, 2006.

[2] K. Brodlie, D. Duce, J. Gallop, J. Walton, and J. Wood. Distributed and Collaborative Visualization. *Comp. Graph. Forum*, 23(2), 2004.

[3] C. Burghardt, C. Reisse, T. Heider, M. Giersich, and T. Kirste. Implementing Scenarios in a Smart Learning Environment. In *Proc. of IEEE Intl. Workshop on PervasivE Learning*, 2008.

[4] E. Chi. A Taxonomy of Visualization Techniques Using the Data State Reference Model. In *Proc. of IEEE Sym. on Information Visualization*, 2000.

[5] D. Cook and S. Das. *Smart Environments*. Wiley-Interscience, 2005.

[6] I. Duda, M. Aleksy, and T. Butter. Architectures for Mobile Device Integration into Service-Oriented Architectures. In *Proc. of Intl. Conf. on Mobile Business*, 2005.

[7] G. Ellis and A. Dix. A Taxonomy of Clutter Reduction for Information Visualisation. *IEEE Trans. on Vis. and Comp. Graph.*, 13(6), 2007.

[8] J. Encarnação and T. Kirste. Ambient Intelligence: Towards Smart Appliance Ensembles. In *From Integrated Publication and Informations Systems to Virtual Information and Knowledge Environments*. 2005.

[9] I. Grimstead, N. Avis, and D. Walker. Automatic Distribution of Rendering Workloads in a Grid Enabled Collaborative Visualization Environment. In *Proc. of ACM/IEEE Conf. on Supercomputing*, 2004.

[10] H. Hagen, H. Barthel, A. Ebert, A. Divivier, and M. Bender. A Component- and Multi Agent-based Visualization System Architecture. In *Proc. of Intl. Sym. on Mobile Computing*, 2000.

[11] T. Hau, N. Ebert, A. Hochstein, and W. Brenner. Where to Start with SOA: Criteria for Selecting SOA Projects. In *Proc. of Ann. Hawaii Intl. Conf. on System Sciences*, 2008.

[12] T. Heider and T. Kirste. Automatic vs. Manual Multi-Display Configuration: A Study of User Performance in a Semi-Cooperative Task Setting. In *Proc. of BCS HCI Group Conference*, 2007.

[13] T. Heider and T. Kirste. Minimizing Cognitive Load by Automatic Display Management. In *Ubicomp Workshop on Attention Management in Ubiquitous Computing Environments*, 2007.

[14] M. Huhns and M. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, 9(1), 2005.

[15] D. Hutter, G. Müller, W. Stephan, and M. Ullmann, editors. *Security in Pervasive Computing*, number 2802 in LNCS. Springer, 2003.

[16] M. Luboschik and H. Schumann. Illustrative Halos in Information Visualization. In *Proc. of Advanced Visual Interfaces*, 2008.

[17] M. Novotny and H. Hauser. Outlier-Preserving Focus+Context Visualization in Parallel Coordinates. *IEEE Trans. on Vis. and Comp. Graph.*, 12(5), 2006.

[18] J. Schädlich and K. Mukasa. An Intelligent Framework for User-Oriented Information Visualization in the Production Automation Area. In *Proc. of Intl. Conf. Information Visualisation*, 2004.

[19] L. Skorin-Kapov, H. Komericki, M. Matijasevic, I. Pandzic, and M. Mosmondor. MUVA: a Flexible Visualization Architecture for Multiple Client Platforms. *Journal of Mobile Multimedia*, 1(1), 2005.

[20] J. Street and H. Gomaa. Software Architectural Reuse Issues in Service-Oriented Architectures. In *Proc. of Ann. Hawaii Intl. Conf. on System Sciences*, 2008.

[21] C. Thiede and H. Schumann. Beschreibung des Kontextes zur Adaption visueller Interfaces in multimedialen adhoc-Umgebungen. Rostocker Informatik-Berichte, Vol. 31, 2007, (in German).

[22] K. Tsoi and E. Gröller. Adaptive Visualization over the Internet. Technical report, TU Vienna, 2000.

[23] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Press LLC, 2. edition, 2006.

[24] J. Yang and M. Papazoglou. Web Components: A Substrate for Web Service Reuse and Composition. In *Proc. of Intl. Conf. on Advanced Information Systems Engineering*, 2002.

[25] Y. Zhao, C. Hu, Y. Huang, and D. Ma. Collaborative Visualization of Large Scale Datasets Using Web Services. In *Proc. of Intl. Conf. on Internet and Web Applications and Services*, 2007.

[26] E. Zudilova-Seinstra and N. Yang. Towards Service-based Interactive Visualization. In *Proc. of Intl. Sym. on Ambient Intelligence and Life*, 2005.