



## **Proceedings of the 2. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering**

Prof. Dres. Christoph Meinel, Andreas Polze, Mathias Weske, Jürgen Döllner, Robert Hirschfeld, Felix Naumann, Holger Giese, Hasso Plattner (edt.)

## **Technische Berichte Nr. 23**

des Hasso-Plattner-Instituts für Softwaresystemtechnik  
an der Universität Potsdam



# Proceedings of the 2. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering

---

Prof. Dres. Christoph Meinel, Andreas Polze, Mathias Weske,  
Jürgen Döllner, Robert Hirschfeld, Felix Naumann, Holger  
Giese, Hasso Plattner (edt.)

### **Bibliografische Information der Deutschen Nationalbibliothek**

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber: Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Redaktion: Prof. Dr. C. Meinel, Prof. Dr. A. Polze, Prof. Dr. M. Weske, Prof. Dr. R. Hirschfeld, Prof. Dr. F. Naumann, Prof. Dr. H. Giese, Prof. Dr. H. Plattner (edt.)

Verlag: Universitätsverlag Potsdam  
Am Neuen Palais 10  
14469 Potsdam  
Fon +49 (0) 331 9774623  
Fax +49 (0) 331 977 4625  
E-Mail: [ubpub@uni-potsdam.de](mailto:ubpub@uni-potsdam.de)  
<http://info.ub.uni-potsdam.de/verlag.htm>

Druck: allprintmedia GmbH  
Blomberger Weg 6a  
13437 Berlin  
E-Mail: [info@allprint-media.de](mailto:info@allprint-media.de)

© Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 2008

Dieses Manuskript ist urheberrechtlich geschützt. Es darf ohne vorherige Genehmigung der Herausgeber nicht vervielfältigt werden.

**Heft Nr. 23 (2008)**  
**ISBN 978-3-940793-42-3**  
**ISSN 1613-5652**

# **Contents**

- 1. Styling for Service-Based 3D Geovisualization**  
Benjamin Hagedorn
- 2. The Windows Monitoring Kernel**  
Michael Schöbel
- 3. A Resource-Oriented Information Network Platform for Global Design Processes**  
Matthias Uflacker
- 4. Federation in SOA – Secure Service Invocation across Trust Domains**  
Michael Menzel
- 5. KStruct: A Language for Kernel Runtime Inspection**  
Alexander Schmidt
- 6. Deconstructing Resources**  
Hagen Overdick
- 7. FMC-QE – Case Studies**  
Stephan Kluth
- 8. A Matter of Trust**  
Rehab Al-Nemr
- 9. From Semi-automated Service Composition to Semantic Conformance**  
Harald Meyer



# Styling for Service-Based 3D Geovisualization<sup>1</sup>

Benjamin Hagedorn

benjamin.hagedorn@hpi.uni-potsdam.de

Styling geovisualizations means to control the geovisualization process, i.e., to define which geoinformation to include and how to visualize them. It is a crucial aspect for enabling flexible and adaptable geovisualization systems and applications as it allows authors and users to influence the visual appearance of the geovisualization and so to adapt it to specific tasks and preferences. Especially for the service-based implementation of geovisualization systems, it has to be specified how a service consumer can define the visual appearance of a geovisualization which might base on distributed heterogeneous geoinformation and might be synthesized by the combination of independent geovisualization services. This paper describes the styling of three-dimensional geovisualizations in two scenarios. First, a Web Perspective View Service (WPVS) for the generation of high-quality visualization of domain specific building information within its GIS context is described. Second, the Web View Annotation Service (WVAS) and its combination with a WPVS for the synthesis of textual annotated views are introduced. Both implementations are demonstrated for a 3D campus model.

## 1 Introduction

### 1.1 The Role of Visualization in High-Level Geoservices

Geoinformation resources and capabilities for geodata processing and geovisualization are increasingly distributed among various providers. Service-oriented computing is concerned with describing, finding, and using these capabilities for different processes, users, and tasks. In the field of distributed geoinformation systems we can distinguish between geodata access, geodata processing, and geodata visualization services. The Open Geospatial Consortium (OGC) has standardized a set of geoinformation services that define interfaces to these functionalities. Important OGC web service standards are the Web Feature Service (WFS) for geodata access, the Web Map Service (WMS) for 2D maps, and the Geography Markup Language (GML) for the interoperable description of geospatial and georeferenced data. The Web Perspective View Service (WPVS), a first

<sup>1</sup> This work has been published in part by Hagedorn and Döllner (2007) and Hagedorn et al. (2007). It has been adapted for the special purpose of this technical report.

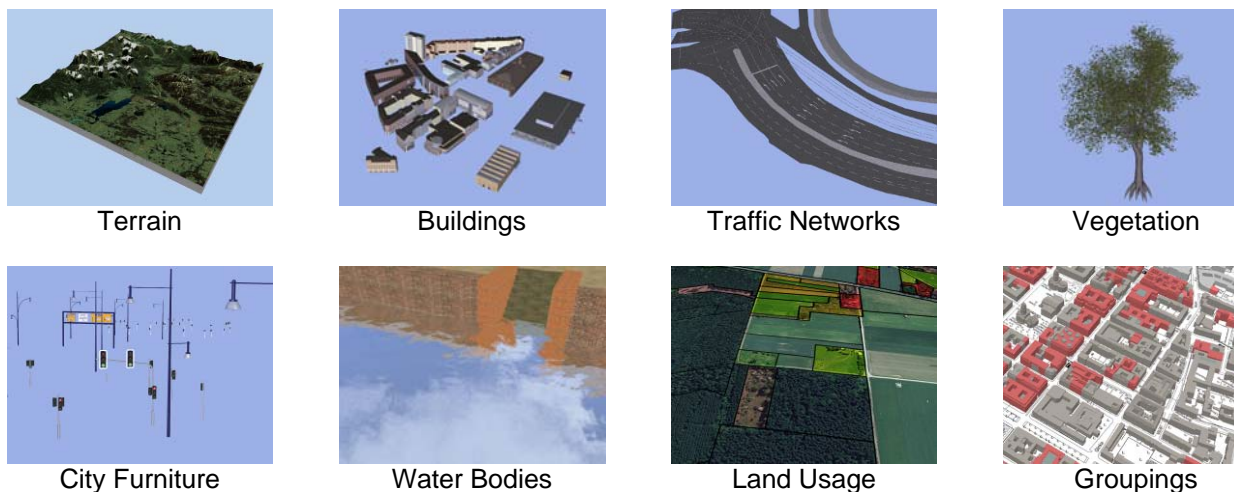


Figure 1: Aspects of complex geoinformation considering 3D GeoVEs.

approach for 3D portrayal, is currently under discussion to become an OGC standard.

The OGC web services such as Web Feature Service (WFS), Web Coverage Service, and Web Map Service (WMS) provide means to communicate geodata and can be considered generic, low-level geoinformation services. However, current geodata infrastructures need to evolve towards integrated systems for the provision of customized information and services (Morales and Radwan, 2004). High-level geoinformation services are generally characterized by the following capabilities and functionalities:

**Enhancing geoinformation:** Include services that enhance geodata by adding, manipulating, correcting, or transforming geodata, e.g., a specialized mass coordinate transformation.

**Provision of specific business functionality:** The usage of (value added) geoinformation for special purposes in a variety of business processes will be a main issue in the future development of service-based geoinformation provision. An example of such specific business functionality is a plausibility check for edited and updated geoinformation. This check includes domain-specific knowledge about the information structure, semantics, and rules for consistency.

**Integration of complex geoinformation:** 3D geovirtual environments (3D GeoVE) provide a conceptual and technical framework for the seamless integration of geoinformation that is different in format, scale, and amount of details. They provide mechanisms to enable the composition, management, editing, analysis, and visualization of this integrated geoinformation. 3D GeoVEs include complex geoinformation such as illustrated in Figure 1. For the integration of complex geoinformation there are two principal ways, "integration at data level" and "integration at visualization level". (Döllner and Hagedorn 2007a)



Integration at data level means the unification of geoinformation by transformations into common data formats or data models. While data format integration is at the syntactical level, data model integration works on a semantical level and introduces a higher added value to the integrated data itself.

Integration at visualization level means the provision of imagery and therefore inherently includes the issues of data selection, data mapping, image synthesis, image transfer, and user interaction.

**Provision of high-quality visualizations:** About 80 percent of all information has a spatial reference which can be used as a basis for the integration and the effective and consistent visualization of such information. Geovisualization can support a user in analysis tasks and in gaining new insights into geodata. In common (two dimensional) geoinformation systems visualization is an essential part. And so it is in distributed geoinformation systems which implies that visualization is a main issue in geoinformation services.

**Support of user interaction:** Especially for visualization services, the support of user interaction can be an essential capability. Additionally, high-level user interaction services are imaginable, which are set up on the top of other geovisualization services.

In this context user interaction means to explore and analyze the information space. For fulfilling these tasks, navigation is an important interaction technique as it allows the user to move around and perceive the information of interest.

**Support of context awareness:** Dey and Abowd (1999) define a system to be context-aware "if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's tasks". Thereby context "is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction of a user and an application, including the user and applications themselves."

In the field of service-based provision of geoinformation, context-awareness refers to identifying and providing information about the tasks the user fulfills and the devices that the users deploys.

Besides high-quality visualization itself, geovisualization plays a role for the implementation of other high-level functionalities, too:

- The integration of complex geoinformation can be performed at the visualization level.
- User interaction is closely linked to visualization as user input by devices such as keyboard, mouse, and stylus lead to updates of the visualization that itself can support interaction by presenting items to interact with.
- Context awareness can be supported by the appropriate visualization for a specific user with a specific task.

## 1.2 Separation and Distribution of Geovisualization Concerns

The visualization pipeline (Spence, 2001) of 3D GeoVEs includes accessing and selecting geodata, mapping them to computer graphical representations, and rendering of depictions, which can be perceived by humans. Using this visualization pipeline as a basis for portrayal raises the question about the exchanged information and the separation of rendering concerns between the portrayal service provider and consumer. The following types of exchanged information can be distinguished:

- Raw data, e.g., a terrain model in raster format.
- Geographic model, e.g., a CityGML city model.
- Computer graphic model, e.g., a VRML scene graph.
- Generated images, e.g., a two-dimensional map or a three-dimensional perspective view.
- Sequences of images, e.g., an MPEG video.

The different types of transferred information also mean different separations of visualization tasks.

In the case of raw data and of geographic models the service consumer has to implement the whole visualization process, i.e., data selection, information mapping, and rendering, which are intensive in hardware, processing, and storage requirements. When exchanging computer graphic models, the service consumer does not need any geographical knowledge for mapping geodata to computer graphic objects. But as a drawback the semantics of this geodata is not available for the rendering process or for supporting interaction. When the portrayal service provides already synthesized images, the service consumer is freed from any computer graphical processing. By this way geovisualization also gets applicable on small devices. Additionally, the protection of the underlying geoinformation against unauthorized usage is inherent in images as transfer formats. For raw data additional effort is necessary for digital rights management. Image sequences are a kind of image provision which especially can improve the usability of portrayal services.

Mixed forms of geographical portrayal services are possible, which provide information from different sources and thereby can reduce the named constraints. As an example a WMS can provide the operation `GetFeatureInfo` and thereby support additional semantical information about a graphical object at a specific pixel position of the previously provided 2D map.

A core concept of service-based systems represents the composition of distributed functionality in a standardized manner. This enables the construction and flexible adaptation of complex and value-added systems and applications. In the geoinformation domain, service composition is often referred to as *geoinformation service chaining*. Alameh (2003) distinguishes three service composition patterns: Client-based chaining, aggregate services, and workflow service-based chaining.

## 1.3 Styling 3D-Geovisualizations

For controlling the steps of the geovisualization pipeline and so the final outcome, different visualization aspects are important. These are especially views, styles, and

rendering and interaction constraints. In Figure 2, they are illustrated in the context of creating geovisualizations by a high-level geoservice and their provision to human users or integration into business processes.

**Views:** Views represent different selections of the information which can be provided by a concrete portrayal service. Views allow customizing the visualization to concrete user groups and user tasks. The views supported by a service correspond with the abstraction level of the service, i.e. its granularity. E.g., a portrayal service for urban management visualization could provide a city planning view, an emergency response view, etc. Views correspond to the filtering stage of the rendering pipeline, as they may define the data which is integrated (e.g., imported from other geoinformation services).

**Styles:** Styles address the mapping of the geographic model on the computer graphic model. Styles influence the graphical character of the synthesized image. For example, they can define:

- Elements to hide or unhide.
- Opacity of objects.
- Graphical character of objects (sketchiness, cartoon style, etc.).

The distinction between view and style is not a strict one. Especially in higher level visualization services, views might include style issues. E.g., in an emergence response view the coloring of buildings of interest might be predefined by the view and is not changeable by user-defined styles.

Both, possible views and possible styles must be represented in the service description and must be considered in the service interface.

**Rendering constraints:** Rendering constraints address aspects such as requirements on image size, image resolution, color modes or transfer formats to use. E.g., imagery can be provided by a static image showing the situation at one point in time or by video which can be used for showing spatial changes of the

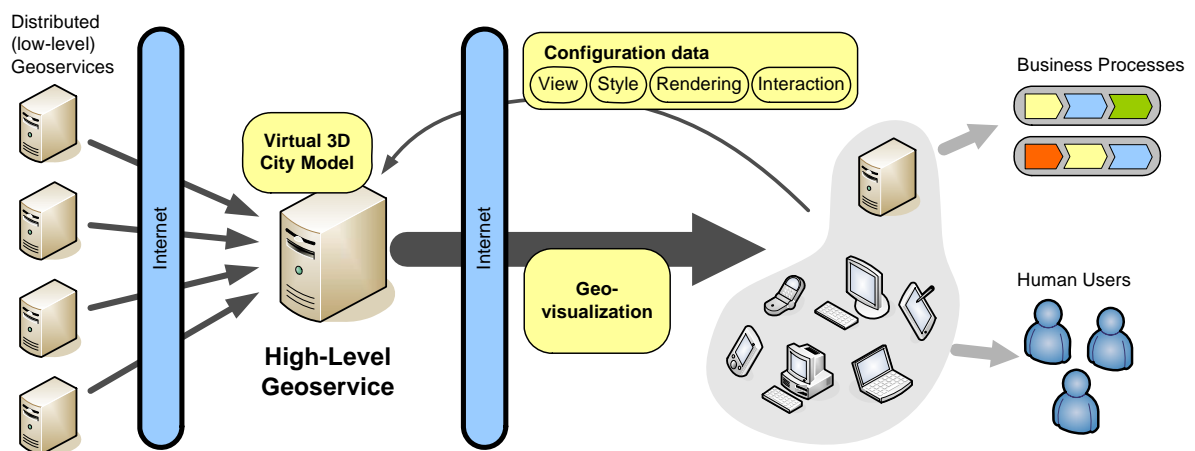


Figure 2: Geovisualization within the context of high-level geoservices.

camera position or for representing changes in time, e.g., for flood visualizations, or for showing historic scenes and their development over time.

**User interaction aspects:** User interaction may be restricted by different constraints, as devices (keyboard, mouse, stylus, etc.), user constraints (age, handicaps, handkerchiefs, etc.), or environmental circumstances (little light, noise, etc.).

## 2 Styling for BIM Visualization

This section describes the implementation of a high-level geoservice for the visualization of building information models (BIM) within its geospatial context. This geovisualization service is designed to integrate complex geoinformation, and provide high-quality geovisualizations for the specific domain of building information. This work has been published by Hagedorn and Döllner (2007). Here, the question of styling is addressed especially by providing different views and visualization styles to the service consumer.

### 2.1 Visualization of BIM by Virtual 3D City Models

While a growing number of manifold geoinformation sources become available, detailed information about individual buildings is typically not managed nor represented by GIS data. To enhance and complement virtual 3D city models in this respect, information provided by CAD models and other building-management tools needs to be integrated and helps to bridge the gap between the BIM, CAD, and GIS domains (Figure 3).

We describe an approach to visualize and analyze CAD-based 3D building information models (BIM) within 3D virtual city models. This way, detailed georeferenced building information about usage, structure, properties, and associated workflows becomes available embedded into their spatial context. Without such integration, we would not be able to see the spatial context of BIM. Furthermore, the approach represents a general strategy to seamlessly integrate georeferenced data from the domain of CAD with GIS data at the visualization level.

For the visualization of building information we identify two challenging tasks:

- The visualization of internals of composite structures (e.g., enabling an inside-view for a building).
- The visualization of intangible information (e.g., the usage of a room or groupings such as stories)

Both are regarded by the building visualization techniques that are described in this section. We present two techniques for the automated ad-hoc visualization of BIM and GIS information. This high-level functionality can be provided in a service-based manner and so can be included into ad-hoc rescue-processes for enabling, e.g., high-quality visualization.

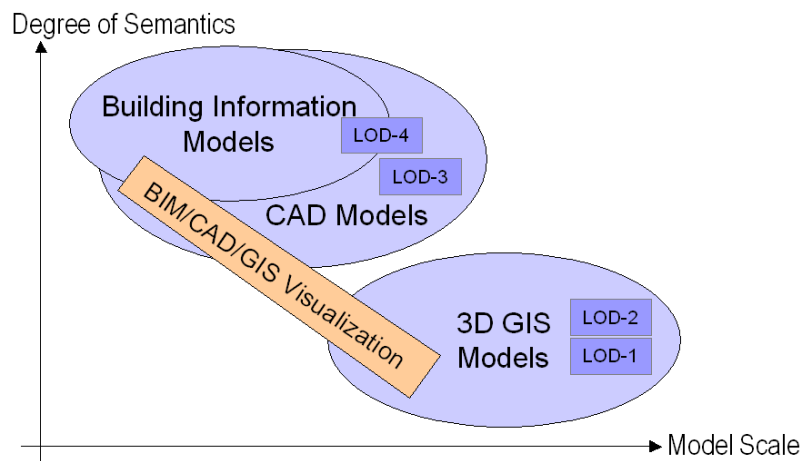


Figure 3: Scales and degree of semantics found in 3D building models with corresponding CityGML level-of-detail categories; the proposed visualization bridges the gap between BIM, CAD, and GIS model use.

As a use case we defined a fire and rescue scenario, which represents an application domain where the availability of building information is of high interest for saving people and assets. The following geo-referenced building information can be considered relevant for such fire and rescue scenarios:

- Fire extinguishers;
- Sensors such as smoke detectors, temperature sensors, or motion detectors for estimating the source of fire and the overall condition;
- Sprinklers, hydrants, hose reels, rising mains;
- Statics and material of building structures, e.g. windows;
- Cables, ducts, and funnels spreading heat, fire, and smoke;
- Information about locking of windows and doors (e.g., lattices) and key owners;
- Storage locations of dangerous substances (e.g., gas, oil, cleaning supplies, and chemicals);
- Expected overall number of people, number of people who need assistance (e.g., young and older ones);
- Navigation hints (e.g., room numbers) for supporting the orientation of helpers;
- Tracking information of people and objects.

## 2.2 Mapping BIM to Geometry

For each BIM data to be mapped onto the city model there must be an existing or derived geometry whose appearance can be adjusted according to the type and value of the building information.

If the original building information is of type geometry, this geometry is reused for the city model object. If no such inherent geometry exists, either another BIM object can be used for representing the building information or a corresponding geometry has to be generated. For example, derived geometry is required for visualizing relationships such as room's connectivity, escape routes (topology) or stories (grouping of rooms). The geometry can be composed from surfaces, e.g., for extracting the story geometry from all associated walls, windows, etc.

Non-geometric building information can be mapped to attributes of geometry. For example, the energy consumption of buildings could be mapped to the height of the building's block model. Alternatively, non-geometric building information can be represented by annotations of the 3D scene, for example, to visualize the presence of extinguishers, HVAC installations, or first-aid kits. 3D annotations can be visualized by texts, symbols, or 3D objects (e.g. an extinguisher model) and are integrated into the 3D scene or positioned within the view plane as described by Maass and Döllner (2006a, 2006b).

### 2.3 Enabling Insight for BIM Visualization

One approach for visualizing BIM information hides and emphasizes building structures: Important objects are highlighted and less important objects are reduced in perceptibility or even removed from the visualization. Apart from removal case, this technique leaves geometry unaltered, which enables to perceive and judge the real-world spatial configuration and relationships.

- A *BIM-view* configuration describes the objects that are included or excluded from the visualization and which BIM-style to apply.
- A *BIM-style* controls the visual variables and defines the appearance of city model objects and building objects.

For the visualization of values, a BIM style offers the possibility to define one of the

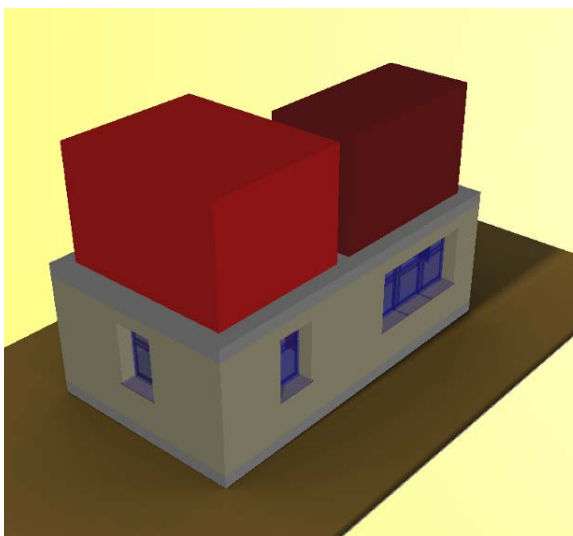


Figure 4: View showing the relative temperature on the second floor.

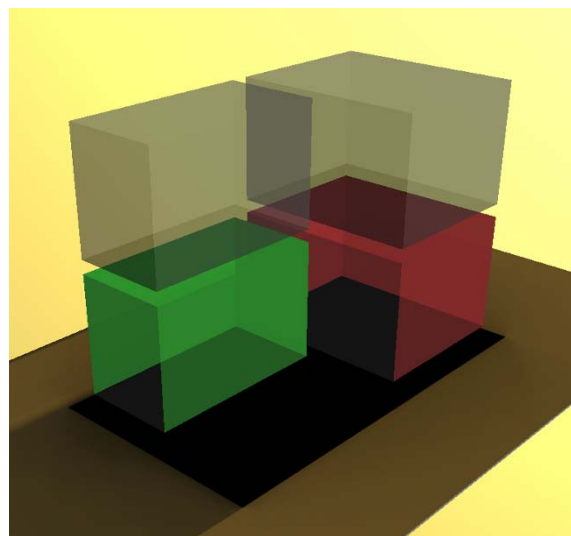


Figure 5: View showing the rooms with hazardous materials.

object attributes whose values modify the original object color. Furthermore, a base color hue (whose value is modified) and a normal range for the attribute's values are defined.

According to the rendering, we are using only color, transparency, and outlines for this visualization technique.

Figure 4 shows a 3D building. Only its second floor is inspected in detail. The view contains all elements of the first floor in normal style. For the second floor only room objects are included. For the story separation CityObjectGroups have been evaluated. The rooms on second floor are colored according to their temperature value from light red to dark red. In the sense of the fire scenario this can indicate the fire source.

Figure 5 shows another view for identifying any rooms in the building that include any hazardous materials, e.g., oil, gas, or chemicals. Therefore, only rooms of the building are displayed in transparent mode and only those rooms are colored which contain such materials.

## 2.4 Deforming Building Structures for BIM Visualization

As another approach to visualizing BIM we distort the geometry of building elements on the basis of their semantics. In detail CityGMLGroupings are evaluated for identifying the stories of a building and all story elements. Those are used for exploding the building model vertically and for popping open the building model vertically, respectively.

For explosion views, a geometrical translation is applied to the building stories and roof. Thereby insight into every story is enabled and indoors elements such as furniture can be perceived without modifying the building objects' appearance, e.g., by using transparency. Such visualization is essential when dynamic data such as



Figure 6: Tilting up of the building for gaining insight.

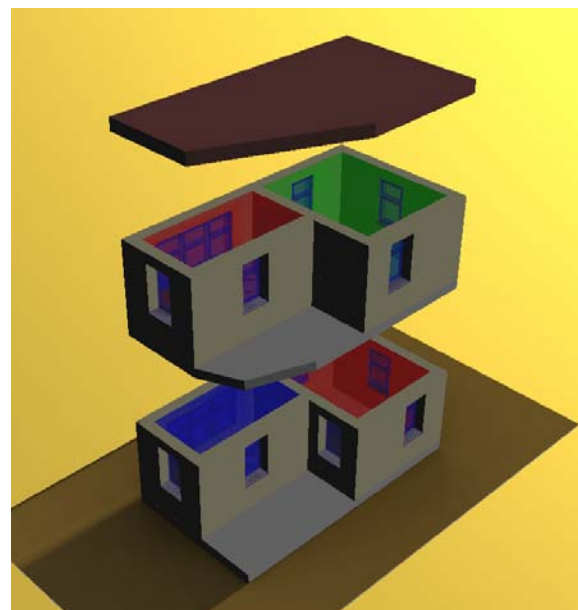


Figure 7: Explosion view for gaining insight; room usage is color coded.

tracking data of firefighters shall be integrated with this building model and all their positions shall be visible at the same time.

A second deforming technique just tilts up the building structures above a defined storey, see Figure 6. On the one side this tilting provides less information than the explosion view, on the other side it needs less image space and still contains more information than just removing the upper building parts from the visualization.

As shown in Figure 7 this deforming visualization technique can be combined with the mapping technique described above. The view includes all building information; only for the second floor the room usage is color-coded.

### 2.5 Web Perspective View Service for BIM (BIM-WPVS)

In general, the BIM-related high-level geoinformation service could be implemented on top of the following OGC web services:

- **Data-oriented:** A Web Feature Service (WFS) would deliver and modify feature objects and their attributes;
- **Scenegraph-oriented:** A Web 3D Service (W3DS) would deliver a scene graph, i.e., a specification of a virtual 3D world including 3D objects, their attributes und hierarchical structure; this data is processed by 3D rendering engines for visualization.
- **Visualization-oriented:** A Web Perspective View Service (WPVS) would provide perspective views of a static 3D scene as image, i.e., a rendering of a 3D GeoVE.

The BIM-WPVS is implemented in a visualization-oriented way. Using a WPVS has the key advantage that there is no need for consumer-side 3D rendering because image synthesis is performed at the server side; the server can be equipped with appropriate 3D computer graphics hardware. In particular, we can deploy high-quality 3D rendering without having to consider the diverse client 3D rendering capabilities since BIM visualization results need only to be transferred to and displayed by light-weight client applications (Singh, 2001). This separation allows us to implement specialized 3D visualization techniques such as for natural phenomena (e.g., including water or sky), to apply processing-intensive techniques (e.g., non-photorealistic rendering, vegetation rendering, or ambient occlusion for simulating global illumination in 3D GeoVEs).

As a disadvantage, WPVS (as well as W3DS) do not transfer any further semantics to the client than the information contained in the synthesized image (or scene graph, respectively). Semantical information is currently only provided by data services, such as the OGC WFS, which, e.g., could deliver a semantical model of a geospatial scene in the CityGML format. This approach would require additional knowledge about the format and processing of CityGML at the service consumer side.

Compared to the OGC WPVS, our BIM-WPVS is extended as follows:

- BIM-WPVS includes integration capabilities for different geoinformation, i.e., terrain data, vegetation, several building data, and additional building information.



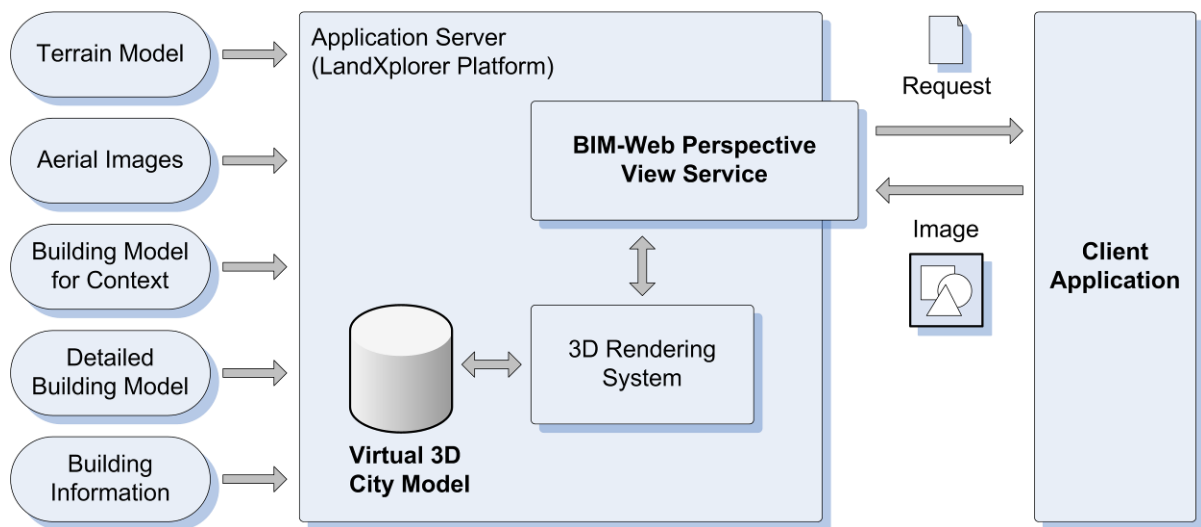


Figure 8: System architecture of the BIM-WPVS.

- BIM-WPVS provides different views that are configurable with respect to visualized data and visualization style.

The perspective view provided by the BIM-WPVS can be integrated into different processes for giving users insight into the geoinformation in an effective way. The application of the BIM-WPVS can be further enhanced by the combination with other services which could enable the access to further complex geoinformation (e.g., to ad-hoc sensor network data), provide functionality for further improving the visual representation of the virtual 3D environment (e.g., by adding annotations to the view), or supply interaction capabilities (e.g., for navigation and editing). Security is another important issue that can be addressed with service composition. In the context of a BIM-WPVS this includes user authentication and authorizing the access to the underlying data which has to be appointed with user restrictions, accordingly.

### 2.5.1 Service Architecture

Figure 8 illustrates the system architecture of the BIM-WPVS. Various sources and formats of geoinformation are accessed, integrated and composed by the central LandXplorer-based server component. It is capable of integrating terrain data, aerial images, LOD-4 building models, additional detailed building information, and further context buildings for the geospatial surrounding on the basis of a virtual 3D city model.

In our case the different geoinformation is accessed from a central database, but it might be distributed and could be accessed by using WFS and WMS web service adaptors that we have added to the LandXplorer CityGML viewer as a contribution to the CAD/GIS/BIM thread within OGC's web services initiative, phase 4 (OGC, 2007). In contrast to that work, the BIM-WPVS deploys a fat-server/thin-client approach.

### 2.5.2 Service Interface

Similar to the OGC WPVS, the BIM-WPVS provides two operations, *GetCapabilities*, which provides information about the service and its capabilities, and *GetView*, which provides the synthesized image.

**GetCapabilities:** Describes the geoinformation layers that are available through the service and can be selected by the service consumer for visualization. Building information is modeled as one layer of the BIM-WPVS. The *GetCapabilities* response further describes different general analysis views, which can be chosen for the server-side image rendering process. These views define the in-scope building information and how they are visualized.

**GetView:** Provides the capabilities of the BIM-WPVS to the service consumer, i.e., the rendering of an image that emphasizes specific building information. For its configuration, the service provides several parameters which a) define the geoinformation layers to include, b) define the position of the virtual camera or leave this to the server for automated calculation, c) define the requested image size, d) define the general analysis view to apply, or e) define a task- and domain-specific view with explicitly setting the relevant entities and the visualization style.

Figure 9 shows the results for two *GetView* requests which integrate detailed building information for a large campus building within its geospatial context (e.g., terrain data, aerial image, and surrounding buildings). They use the explosion view to allow the service consumer to gain insight into the building structure and properties. The examples use color-coding highlighting a single room and for showing the room temperatures within the building.

## 2.6 Results

We have outlined design and implementation of a high-level geoinformation service that helps to close the gap between spatial information at the building level and spatial information at the city model level. We have explained how to map BIM data to components of virtual 3D city models, and exemplified the approach by two BIM-

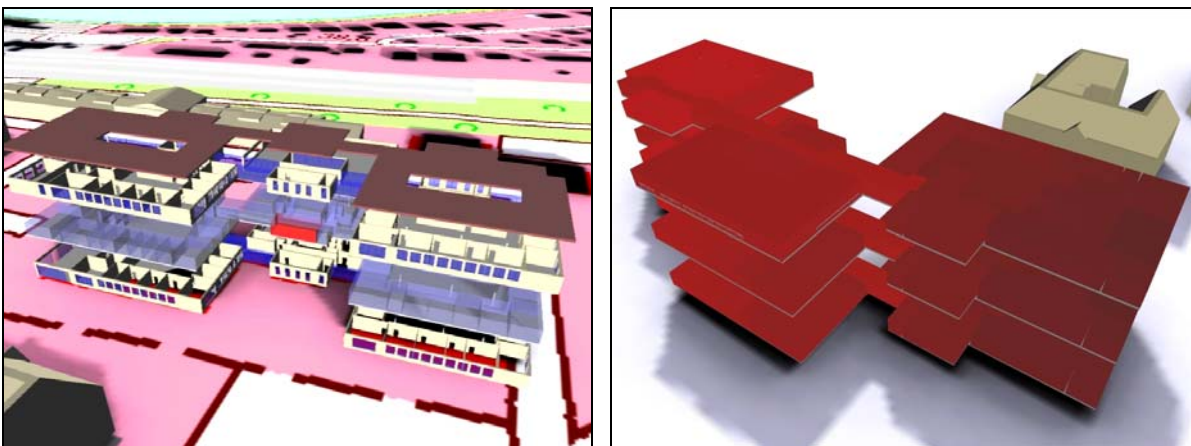


Figure 9: Explosion views of a large campus building within its geospatial context for finding a single room (left) and for showing room temperatures.

specific visualization techniques. We demonstrated the applicability of the approach by our implementation of a BIM-WPVS, which successfully utilizes virtual 3D city models for seamlessly integrating and visualizing GIS and BIM data; rendering styles can be configured by the service operation parameters and different views for analysis purposes can be obtained.

As a general insight, we consider the BIM-WPVS as an example of a high-level geoinformation service that synthesizes complex data and applies advanced 3D visualization techniques while offering a high degree of interoperability due to the server-side 3D rendering.

### **3 Styling with Distributed 3D Geovisualizations**

This section describes an approach for the textual annotation of a perspective view generated by a WPVS. This functionality corresponds to the TextSymbolizer offered by the SLD specification but is extended to 3D geovisualization and is additionally implemented by distributed portrayal and processing services. A composition client uses and combines the functionality of these independent services. This work has been published by Hagedorn et al. (2007).

#### **3.1 Annotation of 3D Geovirtual Environments**

Annotations are essential elements to communicate textual and symbolic information for cartographic maps and within 3D GeoVEs. Traditionally, they name point, line, or area features, such as locations, streets, rivers, districts, or lakes. A number of criteria, e.g., a clear correlation with the feature, the legibility of the annotation, and the occlusion of other annotations or important image parts have to be considered to optimize the information transfer to the user and to achieve an aesthetical appearance of the annotated depiction. The use of electronic media for map creation and presentation raises the need for automated annotation techniques. This need has been strengthened by the increased use of interactivity in today's geovisualization applications and systems, which allows users the real-time exploration, analysis, and modification of geospatial data.

A first approach for the automated labeling of interactive 3D GeoVEs was presented by Bell et al. (2001). They developed a view management data structure that efficiently supports the registration and query of rectangles on the view plane. This is used to mark such regions as occupied that show important scene elements or formerly placed labels. Maass and Döllner (2006b) present another view management strategy that is optimized for point feature labeling of terrains. Furthermore, they develop a technique that directly embeds annotations as 3D elements into the 3D scene instead of presenting them as screen overlays (Maass and Döllner 2006a, 2007).

#### **3.2 Composition Concept**

Our approach to annotated 3D views of GeoVEs combines two main services, an extended Web Perspective View Service (WPVS) and the Web View Annotation Service (WVAS). The service chain is implemented as client-based service composition. Complementary to the thick geovisualization service, this client is constructed as a thin client. It knows about the component services and completely

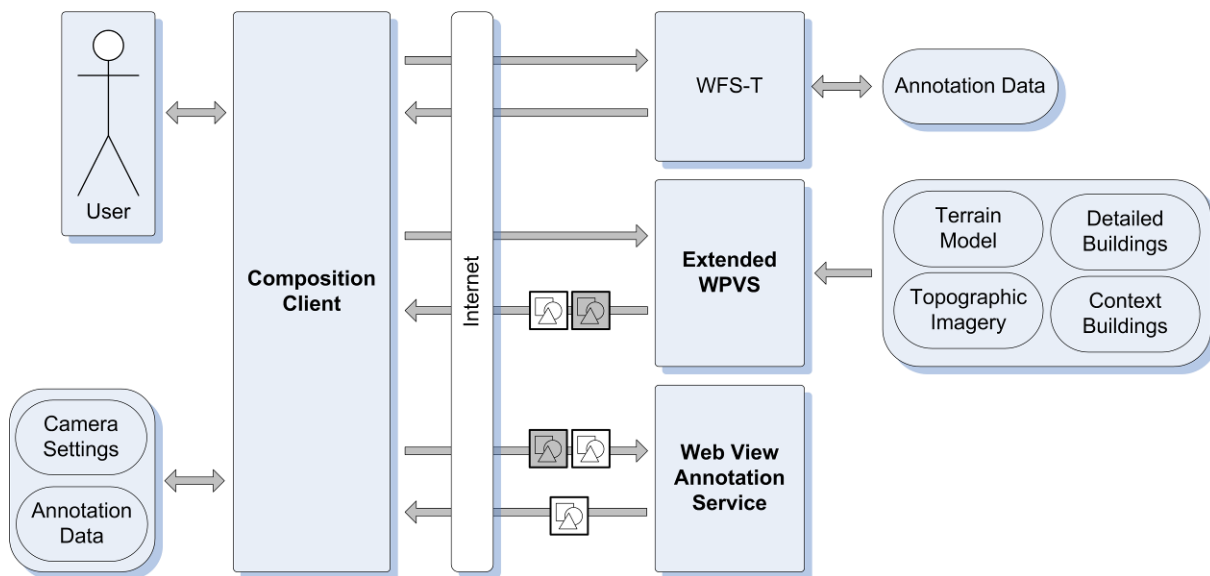


Figure 10: Architecture of the service chain for annotated views of GeoVEs.

controls the service-based geovisualization process. Figure 10 illustrates the overall architecture of our implementation and the main message transfer between the involved components.

Our approach to an extended WPVS encapsulates the access to and the integration of geoinformation in one 3D GeoVE comprising large terrain models, large aerial images, and building data in different formats (e.g., CityGML, 3DS). It is capable of synthesizing high-quality images using rendering techniques for ambient occlusion and atmospheric effects such as sun and clouds. In our case study, the service provides access to the integrated GeoVE of a 3D campus model composed of a terrain model, topographic imagery, and several building models such as main building, auditorium, library, cafeteria, and nearby train station. The composition client receives a perspective view and a depth image from the extended WPVS and forwards them to the WVAS along with a set of annotation descriptions and configurations. Finally, the WVAS calculates and overlays the embedded textual annotations. Figure 11 shows a screenshot of the composition client.

For adding user-defined annotation to the 3D GeoVE, we have integrated a transactional WFS in our WPVS. This data can be requested by other users and can be utilized for creating the WVAS request.

Functionality and interface of the WPVS have been extended to enable their composition with the WVAS. Both services are described in Section 3 and Section 4.

### 3.3 The Web View Annotation Service

The WVAS provides the single operation `GetAnnotatedView`. Service request and response are encoded as SOAP messages sent over Http. For both, input and output, images can be transferred in different ways. First, a URL pointing to the image on a web server or representing a URL-encoded service request, e.g., to a WPVS, can be used. Second, the images can be submitted within or attached to the

SOAP message as described by Powell (2004), e.g., using the SOAP extension DIME (Direct Internet Message Encapsulation) for attaching binary data. In our current implementation the color and depth images are encoded as base64 strings and sent within the SOAP messages.

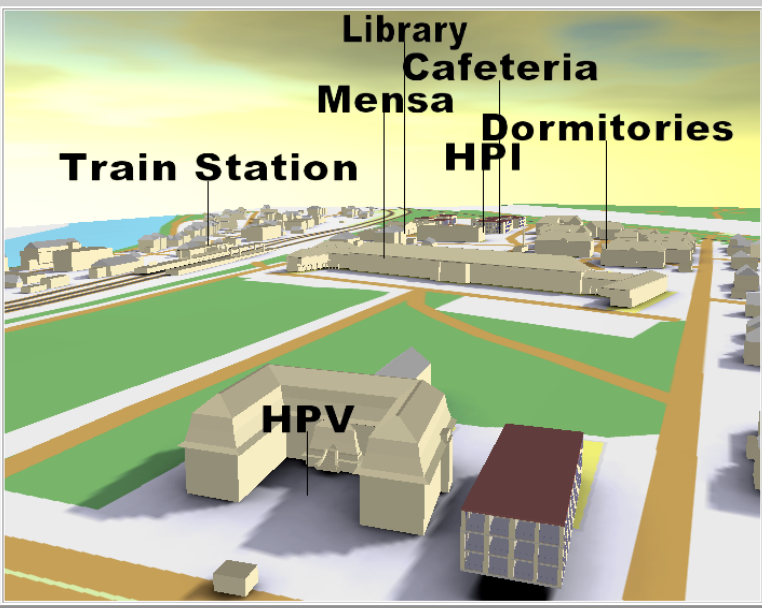
The WVAS is implemented as a stateless service: It does not provide any user or session handling and does not store any camera, canvas, or annotation descriptions. Therefore all input data has to be included in every service request. Some of the data that defined the preceding image rendering process has to be included into the service request as well. In detail the GetAnnotatedView operation use the following input parameters:

- *List of annotations:* An annotation is defined by an annotation text and a 3D position, the annotation's reference point. The annotation can be described as

### WPVS and WVAS Chaining Client

zoom in  
 zoom out  
 move focus  
 insert anno

no annotations  
 with annotations



You pointed at  , .

#### Insert Annotation

Annotation text:	<input type="text" value="text"/>	<input type="button" value="Insert Annotation"/>
Annotation context:	<input type="text" value="undefined"/>	
Reference point:	<input type="text" value="3372370,5806320,62.98"/>	<input type="button" value="Send to Server"/>

Figure 11: Screenshot from the composition client showing an overview of the campus area.

an abstract GML feature containing a georeferenced `gml:Point`.

- *2D color image*: The 2D color image represents a perspective view of a 3D scene for a specific viewpoint and contains RGB values for each pixel.
- *2D depth image*: The 2D depth image is related to the color image. Each pixel stores the distance of the visible scene element to the camera with float precision.
- *Camera definition*: As usual in the 3D computer graphics domain, the camera is defined by the look-from vector, look-to vector, look-up vector, near plane, far plane, and field-of-view angle. This is different from the camera model of the OGC WPVS which is defined by the point of interest (that is the look-to point in our model), the camera distance to that point, angles describing the north direction, pitch, and field-of-view. However, both models can be transformed into each other.
- *Canvas definition*: The canvas definition describes the width and height of the input color and depth images.
- *Annotation configuration*: The appearance of the annotations generated by the technique can be adjusted by parameters such as the placement variant, color, font, and annotation size.

The WVAS is currently implemented as .NET Web Service executed by the Microsoft Internet Information Services (IIS). Its implementation is based on the Virtual Rendering System, “a computer graphics software library for constructing interactive 3D applications. It provides a large collection of 3D rendering components.” (VRS 2007, Döllner and Hinrichs 1995)

### 3.4 Extended WPVS

#### 3.4.1 Extension for Depth Image Provision

The WPVS has been extended by providing additional image-encoded scene information, i.e., the service not only delivers the RGB image of a 3D GeoVE but also an additional image that encodes scene depth. The depth image cannot be created by SLD or SE feature visualization styles, which only influence the mapping of geoinformation to graphical primitives. Instead, the rendering stage in the visualization pipeline has to be modified. We identified at least the following possibilities for integrating such rendering functionality into WPVS:

- *Extending the GetView operation by an additional RenderingStyle parameter*: Depending on this parameter the service decides about the creation of the default color image or the depth image and the format in which they are delivered (e.g., as application/octet-stream for raw binary data).
- *Extending the WPVS service interface by an additional operation GetDepthView*: This operation generates the depth image and only provides appropriate transfer data. The further parameters of the operation are identical to the GetView request.

Both ways provide access to intermediate data of the visualization pipeline, which have not been accessible before to service consumers. This data is not intended for perception by humans but serves as input for new service-based visualization techniques.

Our WPVS implements the additional operation `GetDepthView`. Nevertheless, the option of explicitly supporting `RenderingStyles` seems to be very promising as it is a more general concept for enabling additional rendering techniques.

### 3.4.2 Service-Based User-Interactivity

To support user interaction, we have extended the WPVS by a `GetFeatureInfo` operation. Corresponding to the WMS `GetFeatureInfo` operation, it provides additional information about features in the perspective view of the 3D GeoVE that is returned by a previous `GetView` request. Because of the stateless implementation of the WPVS the `GetFeatureInfo` request has to specify most of the parameters of the `GetView` request, i.e., canvas and camera settings, layers to include, and styles to apply. As further parameter the 2D image position of interest is specified.

The server-side implementation of `GetFeatureInfo` uses a ray intersection test for identifying the selected objects. A 3D ray request is originated at the camera position and shot into the scene. The primarily hit GeoVE object is evaluated and thematic information such as the GeoVE object type (e.g., building, roof, terrain, etc.), object identifier, and geospatial position can be derived and included into the `GetFeatureInfo` response message.

For the interactive specification of annotations, this position can be used as annotation anchor point. Additionally, object-related information such as the object center, footprint center, or other predefined building-related points of interest (e.g., meeting places, elevators, emergency exits) can be delivered to support the user in specifying annotations.

## 3.5 Composition Client

The combination of the extended WPVS and the WVAS is performed by the composition client. The client application catches the user input and forwards it to the extended WPVS. This is the same for the user's navigation input, which is mapped to a new camera position and results in a new execution of the service chain. Additionally, session state handling is addressed, including tracking the camera settings and determining the annotations to be added to the perspective view. In our case, the session handling was performed by the composition client itself. Server-side session handling is another approach, which could be implemented with a service-based composition approach. The sequence diagram in Figure 12 illustrates the steps performed by the composition client to process the user input (for annotation definition) and combine the extended WPVS and WVAS (for annotated view creation). This scenario contains the storage of user-defined annotations at the WFS-T.

Currently, the client application is implemented on a JavaScript basis. As with AJAX, this enables asynchronous communication with web services using the `XMLHttpRequest` object with SOAP messages. The input and output image data is encoded as base64-string and transmitted as part of the SOAP request or response messages, respectively.

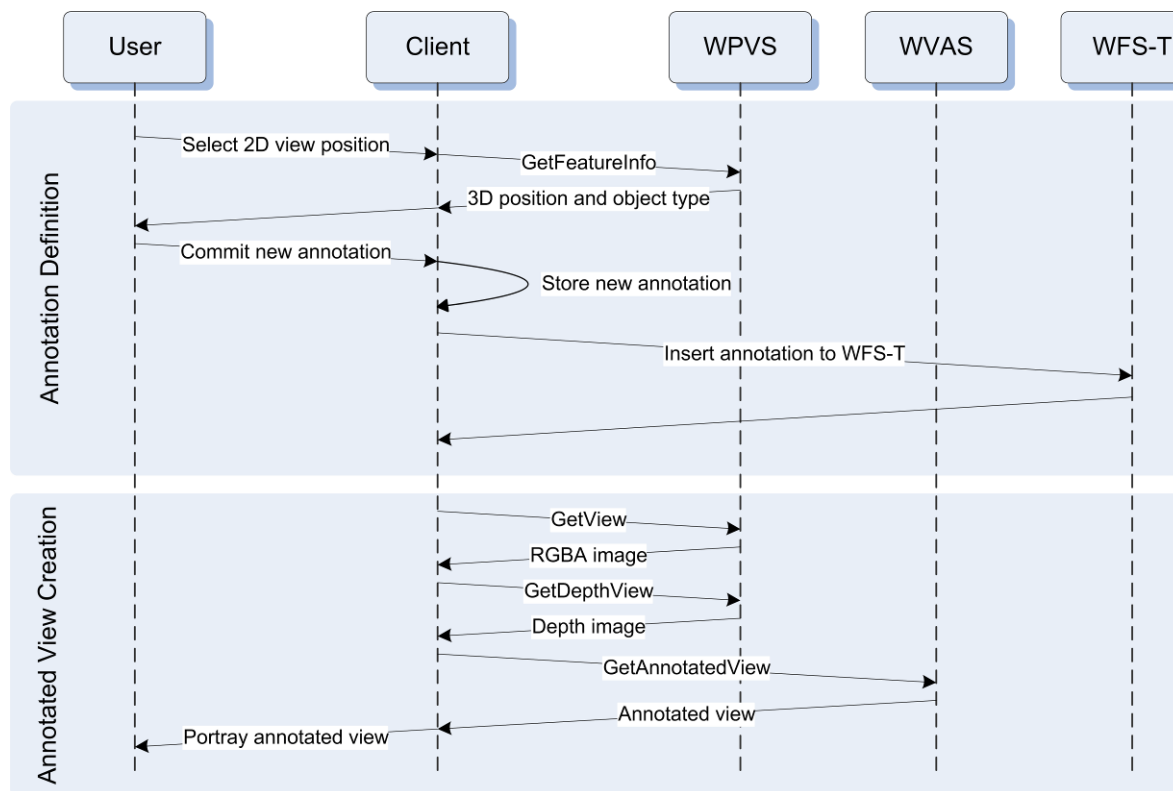


Figure 12: Sequence diagram describing the overall workflow for defining and creating the annotated perspective view.

### 3.6 Results

We have implemented a client-controlled service-chain supporting the 3D annotation of 3D GeoVEs. Figure 13 shows two annotated views of the 3D GeoVE of our campus and how the automatic annotation supports a high legibility by preventing occlusions with scene objects and other annotations.

The presented concept and prototype show how two complementary 3D visualization techniques can be seamlessly combined by implementing these techniques by two independently designed, implemented, and deployed web services; chained together they form a higher-level web service chain. In particular, separating core scene rendering of 3D GeoVE from specialized 3D visualization techniques, for example annotation rendering, facilitates the systematic, modular composition of complex visualization applications and simplifies their implementation. For example, the annotation service could be reused in other web service chains, or it could be enhanced by a navigation information service.

The presented concept can be applied to all visualization services that basically operate in image space provided that the WPVS offers additional scene information. This way, we can use an optimized 3D rendering engine for the most time-critical part, the rendering of complex 3D GeoVEs, and other aligned web visualization services do not depend on the underlying internal representation – they rely on a clearly defined, image-based interface. This approach also offers a high degree of



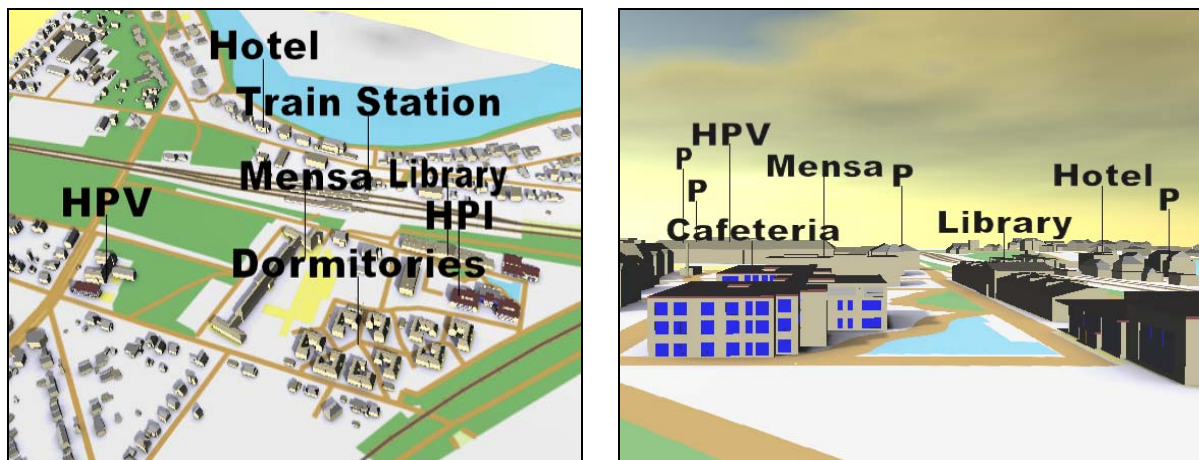


Figure 13: Screenshots of annotated views of the 3D campus model from a bird's eye (left) and close to ground (right) perspective generated by the service chain implementation.

interoperability because the individual web services do not need to exchange contents of 3D GeoVE, for which no commonly accepted standards exist so far. The extensions to the WPVS can be generalized: In all standard 3D graphics systems (e.g., OpenGL, DirectX), the provision of image-encoded scene information (e.g., depth, surface normals, object identities, etc.) represents a common feature. Therefore, we suggest including these extension into the official WPVS definition. For example, higher-level web services that provide advanced stylized images (e.g., illustrations) could be implemented this way.

## 4 Related Work

For the specification of visualization styles for 2D maps the OGC supports the Styled Layer Descriptor (SLD) and Symbology Encoding (SE) specifications (Lalonde 2002, Müller 2006). Together with a reference to the input data, a WMS consumer can use predefined styles for the geovisualization or specify own styles to be applied. Annotations can be integrated into the WMS-based geovisualization in two ways: The SLD and SE specifications support textual and graphical annotations by TextSymbolizers (for label placement) and PointSymbolizers (for 2D graphics placement).

Until now, for the styling of 3D GeoVEs, no standards emerged. Neubauer and Zipf (2007) have proposed extensions to the SLD specification for third dimension. E.g., they suggest SolidSymbolizers and SurfaceSymbolizers as new features, 3D legends, billboard integration, 3D placement, and the extension of material definitions for supporting enhanced shading. This extension is leaned against SLD styling mechanism for 2D simple features (such as points, lines and polygons) but does not address the visualization of more complex structured features such as CityGML buildings, which include walls, doors, windows, etc.

Regarding the service composition for geovisualization, the OGC WMS specification together with the SLD specification supports the definition of external geodata sources for being used within the geovisualization process. Neis et al. suggest an

Accessibility Analysis Service (AAS, Neis et al. 2007a) and an Emergency Route Service (Neis et al. 2007b), which employ service composition by aggregate services. For example, they combine street network data from WFS with processing capabilities of their AAS (which generates new geometry described as GML) and WMS mapping capabilities. Weiser et al. (2007) show the possibility of using BPEL engines together with WSDL service descriptions of OGC web services for a workflow-based service orchestration. We do not know about other work about the service-based externalization of intermediate rendering results for being used with other geovisualization services.

## 5 Conclusions and Future Work

This paper introduces two approaches for enabling styling within 3D geovisualization. First, it targets at a domain-specific styling of complex building information and the visualization within its geospatial context. Second, the externalization of intermediate rendering results and combination with additional rendering functionality in a distributed visualization system is shown.

In our future work, we will address the following aspects: First, we will develop advanced BIM visualization techniques that focus on providing insight into complex spatial assemblies using non-photorealistic, illustrative 3D rendering techniques. Second, we will implement the service chain by an aggregate service, which includes the logic for service chaining, currently implemented by the composition client. Third, we will provide web-service adapters to general web services (WMS, WFS), which provide access to geodata (e.g., maps, 3D terrains, and city models), offer lookup services whose results can be visualized by annotations (e.g., taxi stations, restaurants, or parcel tracking information), and use the annotation service for analysis and exploration tasks. Fourth, we want to enhance user interaction implemented by web services. This includes functionality to interactively specify annotations also for line and area features, used to add user comments in urban city planning scenarios, as well as to support network-based multi-user collaboration.

## References

- [1] Alameh, N. 2003. Chaining Geographic Information Web Services. In *Internet Computing*, IEEE Computer Society, Sept.-Oct. 2003, pp. 22-29.
- [2] Dey, A.K. and Abowd, G.D. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing* (Karlsruhe, Germany, 1999).
- [3] Bell, B., Feiner, S., and Höllerer, T. 2001. View Management for Virtual and Augmented Reality. In *Proceedings of the 14th ACM Symposium on User Interface Software and Technology (UIST)*, ACM Press, pp. 101-110.
- [4] Döllner, J. and Hagedorn, B. 2007. Integrating Urban GIS, CAD, and BIM Data by Service-Based Virtual 3D City Models. *26th Urban Data Management Symposium* (Stuttgart, Germany, October, 2007). UDMS 2007.

- 
- [5] Döllner, J. and Hinrichs, K. 1995. The Virtual Rendering System - A Toolkit for Object-Oriented 3D Rendering. *EduGraphics - CompuGraphics Combined Proceedings*, pp. 309-318.
- [6] Hagedorn, B. and Döllner, J. 2007. High-Level Web Service for 3D Building Information Visualization and Analysis. *Proceedings of the 15th ACM International Symposium on Advances in Geographic Information Systems (Seattle, USA, Nov. 2007)*. ACM GIS'07.
- [7] Hagedorn, B., Mass, St., and Döllner, J. 2007. Chaining Geoinformation Services for the Visualization and Annotation of 3D Geovirtual Environments. *Proceedings of the 4th International Symposium on LBS and Telecartography (Hong Kong, China, Nov. 2007)*. LBS'07.
- [8] Lalonde, W. (Ed.) 2002. *Styled Layer Descriptor Implementation Specification*. Open Geospatial Consortium (September 2002).
- [9] Maass, S. and Döllner, J. 2006a. Dynamic Annotation of Interactive Environments using Object-Integrated Billboards, In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (Plzen, Czech Republic), WSCG'2006*. pp. 327-334.
- [10] Maass, S. and Döllner, J. 2006b. Efficient View Management for Dynamic Annotation Placement in Virtual Landscapes. In *Proceedings of the 6th Int. Symposium on Smart Graphics (Vancouver, Canada, 2006)*, pp. 1-12.
- [11] Maass, S. and Döllner, J. 2007. Embedded Labels for Line Features in Interactive 3D Virtual Environments. In *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualization and Interaction in Africa (October 2007)*, ACM AFRIGRAPH 2007.
- [12] Mäs, St., Reinhardt, W., and Wang, F. 2006. Conception of a 3D Geodata Web Service for the Support of Indoor Navigation with GNSS. In *Innovations in 3D Geoinformation Science*, Abdul-Rahman, A., Zlatanova, S., and Coors, V. (Eds.) *Lecture Notes in Geoinformation and Cartography*. Springer. pp. 306-316.
- [13] Morales, J., and Radwan, M. 2004. Architecting Distributed Geo-Information Services: Beyond Data Infrastructures. In *Proceedings of the 20th Congress of the International Society for Photogrammetry and Remote Sensing (Istanbul, Turkey, July 12-23, 2004)*. ISPRS 2004. pp. 1227-1232.
- [14] Müller, M. (Ed.) 2006. *Symbology Encoding Implementation Specification*. Open Geospatial Consortium (July 2006).
- [15] Neis, P., Dietze, L., and Zipf, A. 2007a. A Web Accessibility Analysis Service Based on the Open LS Route Service. *10th AGILE International Conference on Geographic Information Science (Aalborg, Denmark)*.
- [16] Neis, P., Schilling, A., and Zipf, A. 2007b. Interoperable 3D Emergency Routing Based on OpenLS. *3rd International Symposium on Geoinformation for Disaster Management (Toronto, Canada, May 23-25, 2007)*, GI4DM.
- [17] Neubauer, St., Zipf, A. 2007. Vorschläge zur Erweiterung der OGC Styled Layer Descriptor (SLD) Specification in die dritte Dimension – eine Analyse möglicher Visualisierungsvorschriften für 3D Stadtmodelle. *Symposium Angewandte Geoinformatik (Salzburg, Austria, July 4-6, 2007)*. AGIT'07.
- [18] OGC 2007. Summary of the OGC Web Services, Phase 4 (OWS-4) Interoperability Testbed. [http://portal.opengeospatial.org/files/?artifact\\_id=21371](http://portal.opengeospatial.org/files/?artifact_id=21371).

- [19] Powell, M. 2004. Web Services, Opaque Data, and the Attachments Problem. Web Services Technical Articles. The Microsoft Developer Network Library. <http://msdn2.microsoft.com/en-us/library/ms996462.aspx>.
- [20] Singh, R.R. (Ed.) 2001. OpenGIS OGC Interoperability Program Report, OGC Web Terrain Server (WTS). Open Geospatial Consortium.
- [21] Spence, R. 2001. Information Visualization, Addison Wesley.
- [22] Weiser, A. and Zipf, A. 2007. Web Service Orchestration (WSO) of OGC Web Services (OWS) for Disaster Management. 3rd International Symposium on Geoinformation for Disaster Management (Toronto, Canada, May 23-25, 2007), GI4DM 2007.
- [23] VRS 2007. The Virtual Rendering System, <http://www.vrs3d.org>.

# The Windows Monitoring Kernel

Michael Schöbel

michael.schoebel@hpi.uni-potsdam.de

We describe the Windows Monitoring Kernel (WMK), a custom-built version of the latest Windows Server 2003 operating system that includes a fine-grained logging infrastructure for operating system kernel events. Event traces can be used for analyzing the Windows operating system behavior at runtime. Furthermore, the event trace can be used to investigate behavior of arbitrary applications running on the Windows operating system.

Besides describing the WMK design and implementation, we present reporting tools that visualize the logged events and their relationships. We discuss how the WMK can be used to get a better understanding of OS behavior and application behavior.

Our work is based on the Windows Research Kernel, a version of the Windows sources available to academic institutions.

## 1 Introduction

In this section we define the context of the work described in this paper. First, we show how the Windows Research Kernel (WRK) can be used for dynamic program analysis and introduce the developed *Windows Monitoring Kernel* (WMK). Afterwards, the usefulness of the WMK in the field of server system analysis and performance tuning is described. An outlook on the application of the WMK on server systems is given.

### 1.1 WRK and dynamic program analysis

Since the introduction of the Windows NT operating system in 1993, very few details about the kernel structure and behavior have been known to the research community. While UNIX and Linux sources are freely available for research and analysis, Windows NT has always been a *closed source*. Besides the definitive book by Mark Russinovich and David Solomon [6] only limited documentation is available of how things work inside the Windows kernel.

In 2006, Microsoft changed its policy for the kernel sources [5] and published the core part of the Windows Server 2003 Enterprise Edition kernel sources as the Windows Research Kernel (WRK). The WRK is available to academic institutions for research purposes [3, 11] and can be used to build customized versions of the Windows kernel. It requires a running instance of a Windows Server 2003 Enterprise Edition, as it only provides a modified version of `ntoskrnl.exe`.

Available kernel source code can be used in different ways to analyze the Windows operating system or applications running on the Windows platform. Static code analysis can be done. We used the open source tool *doxygen* to generate call graphs and HTML documentation for the kernel<sup>1</sup>. The subject of this paper is our tool for dynamic analysis: we have prepared a modified WRK version called Windows Monitoring Kernel (WMK).

The WMK adds an event logging infrastructure to the Windows kernel that allows logging of any arbitrary function/activity inside the source code. We also developed a reporting tool that allows post-mortem analysis and generates diagrams or graphs.

To analyze an arbitrary Windows application the original Windows kernel has to be replaced by the WMK version. The WMK can operate in two modes: either *global* logging is enabled, which can be used to log every event in the system independent from a specific application, or *local* logging which can be used to monitor only specific applications. The global logging mode allows the monitoring of the Windows boot process and all inter-process dependencies. Local logging of specific applications can be used for debugging or analysis of specific intra-process activities.

Different event types provide different pieces of information about runtime behavior. Basic events such as scheduling activities, process synchronization or file access in combination with timestamps can be used to analyze important aspects of applications. Profiling, dead lock detection, and determining lock contention can be done.

The WMK builds the foundation for a collection of tools for dynamic program analysis on the Windows platform. The availability of source code allows the implementation of complex instrumentation directly in the kernel. This allows the tracing of events which are otherwise not accessible.

## 1.2 Dynamic program analysis and server systems

Monitoring and observing runtime behavior of server systems can be used in different ways [1, 7]. Two main application areas are debugging and optimization:

**Debugging** - Most server systems use a set of components which are connected in a complex way, e.g. a web server, an application server, and a database management system. Subtle runtime errors can be detected by observing the connected components and the exchanged data.

**Optimization** - By continuously observing performance metrics (e.g. throughput or latency) it is possible to detect bottleneck components or components which are never used. Afterwards, such components can be optimized or removed.

Furthermore, concepts as described in [9, 10] rely on a fine-grained, low-level understanding of server request processing. The Windows Monitoring Kernel can be used as a flexible foundation for this problem area and others. It allows for event logging of arbitrary events as described in the remaining parts of this paper.

<sup>1</sup>Access via the web is provided on request [11].

### 1.3 Contributions and structure

The main contributions are as follows: (1) We built a modified version of the Windows NT kernel which can be used for dynamic program analysis, (2) we implemented an efficient event logging infrastructure therein, (3) we implemented tools that analyze and visualize recorded events, and (4) we demonstrate the application of the WMK system for analyzing system behavior.

The WMK can be used as basis of a complete logging and debugging infrastructure as described above.

The remaining part of the paper is organized as follows: First, we present design and implementation of our event logging infrastructure. Secondly, we evaluate the performance impact of the WMK. Afterwards, detailed case studies are presented which show the possibilities offered by the WMK. Finally, we review related work and conclude the paper with an outlook on future work.

## 2 Design and Implementation

In this section we present the overall architecture of the WMK logging infrastructure. As it is a straight-forward implementation derived from our requirements, we will start motivating those requirements, presenting details of the architecture, introducing currently supported event types, and finally presenting the API of WMK as well as tools that help evaluating and/or interpreting traced logfiles.

### 2.1 Requirements

The Windows Monitoring Kernel is designed for providing an efficient event logging infrastructure that allows for logging arbitrary kernel events. For this reason, we define the following conditions to be met by the WMK:

**Reliability** : WMK events are used for system evaluation. Therefore, it is crucial that the underlying infrastructure is reliable with respect to missing or dropping events. That is, events must not be dropped silently without any confirmation. Otherwise evaluation results might be corrupt.

**Low Overhead** : WMK must not disturb the system's performance by orders of magnitude, as, for instance, a kernel debugger does. However, we consider a small amount of overhead suitable for the sake of monitoring arbitrary kernel events. We will discuss the performance impact in more detail in section 3.

**Availability** : The event logging infrastructure must be accessible from within all kernel modules and must be executable under any arbitrary circumstances, like processor mode or interrupt request level (IRQL).

**Non-blocking Synchronization** : Logging kernel events requires efficient synchronization between different event providers. The synchronization must be implemented in a non-blocking, wait-free manner, i.e., logging an event must not lead

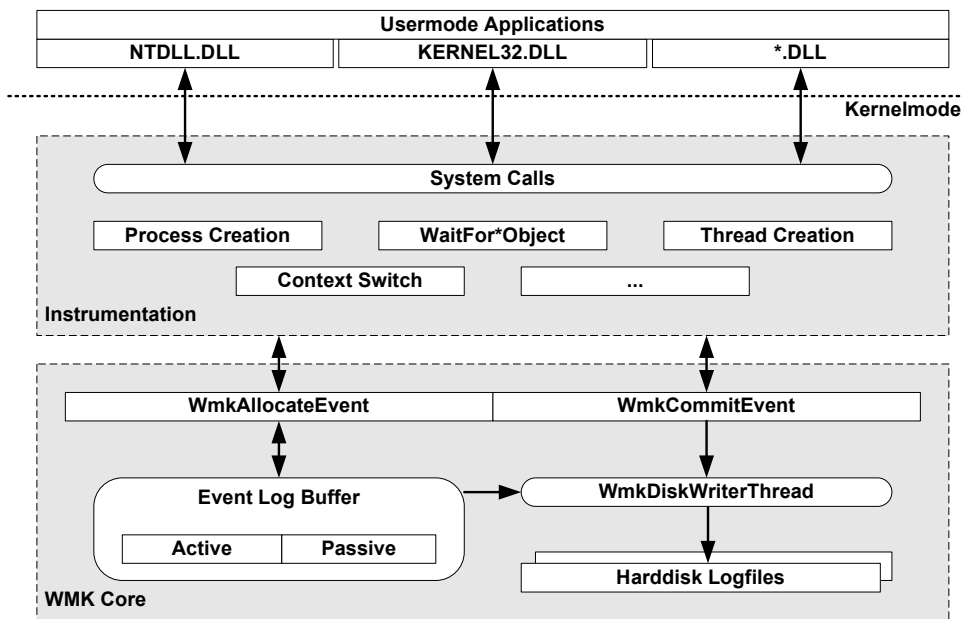


Figure 1: WMK components - overview

to the preemption of the thread or task currently being executed. For example, if a blocking system function call is executed while instrumenting an interrupt service routine like the page fault handler, the system will halt with the infamous *blue screen of death*, because no other task is scheduled until the ISR is completed.

**Variable Size Events** : In order to achieve logging of arbitrary events, WMK must provide means to define custom event types regardless of their size, i.e., different event types may have different sizes. Also WMK must provide means for dynamic-size events, i.e., events whose size is unknown at compile time. Such events, for example, include file access events where it might be considered useful to also log the name of the file.

**Ease of Use** : WMK must be easy to use for both operating system developers and application developers or application maintainers.

## 2.2 Architecture

The architecture of WMK is shown in Figure 1. The *WMK Core* component forms the basis of the WMK logging infrastructure. It is implemented directly as part of the kernel in order to achieve access to it from within instrumented kernel modules. WMK Core contains the event log buffer and implements both the disk writer thread and the WMK event logging API.

The event log buffer is allocated from the non-paged pool section of the operating system's address space, i.e., it is not subject to paging. This is essential in order to fulfill the availability requirement: page faults are not always allowed to occur when an event might be logged. For example, when instrumenting an ISR like the page fault



```

WmkAllocateEvent(WMK_E_TIMER_EXPIRATION)
    WmkEvent->Object = Timer;
    WmkEvent->ExpirationType = WMK_EXPIRED;
    WmkEvent->DueTime = Timer->DueTime.QuadPart;
    WmkEvent->Hand = Index;
WmkCommitEvent(WMK_E_TIMER_EXPIRATION)

```

Figure 2: Logging a timer expiration event

handler or a deferred procedure call (DPC), the occurrence of a page fault in such a condition will cause the system to crash.

The event log buffer is divided into two independent parts; each is dynamically assigned either the role of the active buffer (= currently used for storing events) or the role of the passive buffer. When the active buffer is exhausted, its events must be written to disk. Therefore, the passive buffer is switched to the “active” state and the other buffer is written to disk by the disk writer thread.

The capacity of each part of buffer must be big enough to cache data while the other part is written to disk. Also, the capacity of each buffer has an impact on the performance penalties caused by the WMK. If the buffer is too small, the disk writer thread is almost always active to flush events to disk and events might be lost. On the other hand, if the event log buffer is too big, the performance might be influenced indirectly as OS resources might run short. For our prototypical implementation, we experienced a buffer size of 4 MByte for each part of the buffer (= 8 Mbyte for the complete buffer) as sufficient for a typical workload as defined in section 3.

### 2.2.1 Event Logging API

A kernel event provider must invoke the WMK API for creating an event in the logfile. There are three steps that are executed by every event provider: it first allocates the event, secondly it fills the event with data, and finally, it commits the event.

A kernel event is allocated by invoking either `WmkAllocateEvent` or `WmkAllocateEventEx`, depending whether the event provider wants to create a fixed-sized event or a variable-sized event. In either case, the WMK tries to allocate the appropriate buffer size from the active buffer. In order to provide a lock-free synchronization between concurrent event allocators, we use hardware instructions that can atomically compare and exchange a value, e.g., `CMPXCHG` on Intel architectures.

Afterwards, the event data structure can be filled with appropriate data items, like in figure 2, for a timer expiration event. In that case, the event structure contains four fields: *Object* denotes the address of the timer object, *ExpirationType* denotes why the timer was released, *DueTime* denotes the absolute system time when the timer has to expire, and *Hand* identifies the timer’s timer list. When all items have been set appropriately, `WmkCommitEvent` must be invoked to signal that the event is ready to be flushed to disk.

While other implementations of how to log kernel events are proposed in recent

publications [4, 13], we decided for this allocate-commit approach for the following reasons: on the one hand, `WmkAllocateEvent` transparently checks whether the given event type is enabled for logging. If not, nothing else is executed. On the other hand, event providers can enclose additional function calls in the allocate-event bracket, so these functions are only invoked as demanded.

### 2.2.2 Disk Writer Thread

For performance issues, events are stored in the active buffer. If the active buffer is exhausted, data needs to be written to some secondary storage, like a disk. But as I/O accesses are slow, the chance is high to miss some events in that time. Therefore, we use the passive buffer: If `WmkAllocateEvent` detects that an event will not fit in the remainder of the buffer, it switches both buffers, i.e., the active buffer becomes the passive buffer and vice versa. Then, the disk writer thread is signaled, i.e., it resumes from suspension, and starts to write the contents of the “new” passive buffer to disk as soon as all pending events are committed. When the thread finishes writing it marks the passive buffer as empty and blocks.

The size of both buffers is crucial to proper event logging: if buffer size is too small, the disk writer thread will be unable to write the passive buffer to disk before the active buffer is exhausted. On the other hand, if buffer size is too big, the OS might indirectly be influenced as only fewer system resources like non-paged pool are available. We further evaluate proper buffer sizes in section 3.

### 2.2.3 User-mode API

The WMK also provides an API to user mode, namely a set of system calls, that can be used for both controlling and configuring dynamic properties. The properties of the WMK are part dynamic, part static. Static properties are set at compile-time. Such properties include, for example, the buffer size of the event log buffer, and the initial logging type, like global or per-process logging. Static properties are crucial as the WMK is available to kernel modules at a very early stage of the operating system boot process. Dynamic properties can be set and/or modified during the run-time of the OS. As the WMK is a part of the operating system, we implemented new system calls to allow for configuration. The following properties can be configured dynamically:

**Enabling or disabling global logging** : The WMK allows for global logging, i.e., all events occurring in the system will be logged. In contrast with per-process logging, only those kernel events will be logged that happen in the context of a certain process or in the context of any of its child processes. If global logging is disabled, per-process logging is enabled. Kernel events are logged no sooner than a process identifier is specified.

**Attaching the WMK to a process** : We provide a system call that allows for logging only those events of a certain process or any of its children that were created after this process was selected for per-process logging.

Event	Description
ProcessCreation	a new process is created
ProcessTermination	a process terminates
ThreadCreation	a new thread is created
ThreadTermination	a thread terminates
WaitEvent	a thread waits for one or more synchronization objects
WaitRelease	a thread actively signals a event or releases a synchronization object
Syscall	a system service call occurs
SyscallExit	a system service call returns
ContextSwitch	the scheduler passes the CPU to another thread
QuantumEnd	a thread exhausts its quantum
CreateObject	an object reference is created
CreateFile	a file reference is created
TimerExpiration	a (kernel) timer expires

Table 1: WMK Event Types

**Defining a subset of kernel events** : As we will motivate in section 3, it can be helpful to reduce the amount of traced kernel events if only a certain aspect of the kernel is of interest. For example, when we evaluated the Windows timer management we disabled all other kernel events to concentrate only on that type of events.

**Requesting the active buffer to be flushed** : Normally the event log buffer is only flushed to disk if its space is exhausted. However, if the event arrival frequency is not high, it may take time until the buffer is actually flushed. To enhance this process, a user can request the WMK to flush the buffer anyway. Especially, if only few events are enabled for logging and a certain process is monitored, we found this feature to be useful.

## 2.3 Supported Events

Table 1 shows the currently supported event types. Following, each event type is described in more detail.

**ProcessCreation** If a new process is created the corresponding event contains information about the process creator (thread ID and process ID) and the newly created process (process ID, process object address, and section base address). The section base address holds the memory address where the process executable is loaded. With this information the process tree can be derived.

**ProcessTermination** On process termination the calling thread and the process ID of the terminated process are logged.

**ThreadCreation** If a new thread is created the logged event contains information about the thread creator (thread ID and process ID), the newly created thread (thread ID and process ID), thread object address, thread start address, and the executable image name the thread belongs to. The thread start address in combination with the section base address of the corresponding process can be used to resolve the thread function name if a PDB file with debug information is available.

**ThreadTermination** On thread termination the thread ID and process ID of the terminating thread are logged.

**WaitEvent** If a thread invokes a wait function the logged event contains the thread ID and the process ID of the waiting thread, a flag indicating whether the thread starts or finishes waiting, a list of objects the thread is waiting for, and (if applicable) the signaled object. Additionally, the return value of the wait function is logged to detect timeouts. The timestamps of wait start and wait end can be used to calculate the timespan the thread was blocked.

**WaitRelease** If a thread signals an event or releases a semaphore, the signaling thread (thread ID and process ID), the address and type of the signaled synchronization object and the resulting priority boost are logged. The object address can be associated with current WaitEvent entries and the synchronization relation between two concurrent threads can be detected.

**Syscall** If a user mode application calls the kernel, the thread ID, process ID and the system service call number are logged.

**SyscallExit** If the kernel has processed the system service call of a thread, the thread ID and process ID of the calling thread are logged. In combination with the latest Syscall event for the specific thread, the processing time for the system service call can be determined. Additionally, the WMK logfile can be used to identify system service calls which may not return to user mode (e.g. `NtTerminateProcess`).

**ContextSwitch** If the Windows scheduler selects a new thread for execution, the information about the new thread is logged: thread ID, process ID, thread priority and thread quantum length. With this information Windows scheduling concepts like priority boosts and quantum expansion can be analyzed.

**QuantumEnd** If a thread's quantum expires, the according thread ID and process ID are logged. A subsequent ContextSwitch event shows which thread was selected to run next.

**CreateObject** If a thread creates a new object which is managed by the object manager, the creator information (thread ID and process ID), the object type, the object name, and the object address are logged.

**CreateFile** A special type of objects are files. If a thread tries to access a file, the thread information (thread ID and process ID), the requested access type (e.g. write access), the result of the `CreateFile` call, and the file name are logged. With this information, files used by a specific process can be detected.

**TimerExpiration** If a kernel timer expires this event logs the kernel timer object address, the expiration time, the timer index value, and the expiration status of the timer. This event can be used to analyze the Windows timer management (see section 4.2.3 for further explanations).

## 2.4 WMK Tools

We have created a set of tools for WMK logfile analysis. In this section we give a short overview of the different kinds of analysis tools we created for the WMK.

Before any analysis the logged events must be re-sorted: possibly the timestamps of the WMK events are not in the right sequence. This could happen if a `WmkAllocateEvent` call is preempted directly after the timestamp counter was read but before the space for the event was allocated in the buffer. Actually, this happens very seldomly.

The logfile analysis tools can be subdivided into three groups:

**Reporting** WMK events are logged in binary format. Different tools can prepare the information in a human readable form. Additionally, different statistics can be derived from the logfile and displayed in textual form.

**Profiling** The logged events are tagged with a timestamp counter value. This information combined with the test system timestamp counter frequency (= CPU frequency) can be used to derive timing information, e.g. the time spent in different system service calls or the time waiting for a specific synchronization object.

**Graphical Visualization** The logfile can be used to derive the process/thread hierarchy and the wait-for relation of threads and synchronization objects. These dependencies can be visualized in a graphical way and allow a good exploration of the logged information.

Where possible, external sources of information such as PDB files with debug information are used to enhance the reports generated by the WMK tools. For example a thread start address can be resolved with a PDB file to get the real source code name of a specific thread function.

In section 4 the application of the described tool set is demonstrated. Additional event types can be used to create more tools which analyze different aspects.

Besides these tools for analyzing logfiles, we created a set of tools for the WMK configuration. The API described in section 2.2.3 was used to implement tools for: (1) enabling/disabling certain events, (2) starting a monitored process, (3) attaching/detaching the WMK to running processes, and (4) flushing and closing the logfile for further analysis.

## 3 Evaluation

The Windows Monitoring Kernel affects the execution of applications. An evaluation of overhead is subject of this section.

Three main factors determine the overhead introduced by the WMK: the number of logged events, the size of logged events, and the buffer size. The event quantity depends on the characteristic of the executed workload. An IO intensive application with many system service calls naturally leads to much more (system call) events than a pure user mode computation. The event size depends on the event types. The CreateFile event saves the file name in its unicode string representation which leads to a bigger event size than a QuantumEnd event which stores only the thread information. With more/bigger events, the buffer size determines the frequency the log file must be flushed to secondary storage. A smaller WMK buffer must be written to disk more frequently, given a constant amount of events. On the other hand, the time required to flush the buffer increases with the buffer size. Therefore, the time the executed application is influenced by the WMK is bigger.

If an instrumentation point is reached, the WMK checks whether the event has to be logged or not. This depends on the WMK configuration and on the configuration of the executed application. Besides these checks the WMK has no influence on the executed application. Of course, the amount of main memory occupied by the WMK buffer may influence running applications in terms of available memory.

The more interesting case is the presence of many events. We chose the following test case, which leads to a high amount of events: for test purposes the Windows Research Kernel is compiled, and the elapsed time is measured. The high IO activity of compiler runs leads to many CreateFile and Syscall events. The required time to generate the Windows kernel is a good measure for system performance. Additionally, the number of dropped events is evaluated. The size of the WMK buffer should be chosen in a way that no events are lost.

The test machine was a single processor Pentium 4 (2.66 GHz, 768 MByte RAM) machine, with Windows Server 2003 Enterprise Edition SP1. The original kernel was replaced with the WMK version to test. The WRK is compiled ten times and afterwards the logfile is closed and analyzed. This experiment is repeated ten times, leading to 100 WRK compilations for the different WMK configurations described in the remaining of this section.

Using the original Windows kernel the compilation of the WRK source code took 165.7 seconds. This base value was determined by 250 builds.

The WMK overhead must be related to the event frequency. All events shown in table 1 were activated and logged during the compilation. This configuration leads to around 113 million events. Each event had an average size of 32 Bytes, leading to a 3.3 GByte logfile. With a compilation time around 170 seconds per WRK build the WMK has to log about 64k events per second.

Table 2 shows the slow down of the compilation with different buffer sizes. Two observations can be derived from the test results:

1. The overhead introduced by the WMK is nearly constant and independent from buffer size. The trade-off between buffer size and disk write frequency seems to be balanced.
2. With a buffer size  $\geq 8$  MByte no events are lost. Losing events is not acceptable for any system analysis task. Therefore the determined 8 MByte buffer size builds

BufferSize [MB]	Slow Down [%]	Dropped Events [%]
2	6.5	14.3
4	6.3	7.2
8	6.3	—
16	6.1	—
32	6.4	—
64	6.3	—
96	6.2	—

Table 2: WMK Buffer Size Effects

Event	Slow Down [%]	Events [ $\approx \frac{1}{s}$ ]
<i>no events</i>	1.1	—
Process*	1.1	3
Thread*	1.3	4
Wait*	1.2	580
Syscall*	5.2	53604
ContextSwitch	1.1	395
CreateObject	1.8	5159
CreateFile	1.8	4901
TimerExpiration	1.1	97

Table 3: WMK Event Type Influence

the absolute minimum.

The actual value of the sufficient buffer size could depend on the speed of the hard disk used for logfile storage. That influence was not investigated further and should only have slight impact.

Another point to mention is that our test system could not boot successfully with a buffer size  $> 128$  MByte. Allocating more than 128 MByte of the available RAM of 768 MByte as non-paged area seems to acquire too much system resources. The non-paged pool size is determined by heuristics at system startup. Therefore, instead of using fixed buffer sizes one may consider sizing the buffer dynamically.

In short, the WMK buffer size should be chosen in a way that no events are dropped. With an event frequency of roughly 64k events per second the overhead is 6.1%.

Table 3 shows the influence of each event type on the WMK overhead. The third column shows the approximated number of events per second of the specific category.

The main observations derived from the test results are:

1. For our test workload the basic overhead is 1.1%. With the WMK infrastructure in place and disabled events a basic overhead arises (due to checking whether the event has to be logged or not). This value holds for the frequency of 64k events/checks per seconds.

2. Most events are Syscall events. More than 80% of the logged events are system service calls. The Syscall event alone is responsible for around 5.2% of the WMK overhead. If the system service call information is not needed the WMK impact can be reduced significantly.
3. Circa 13k events per second lead to additional 1.0% overhead. With this rule of thumb the WMK overhead can be estimated considering the expected event frequency.

In conclusion, the WMK has a performance impact on running applications. This impact is composed of (1) the cost for checks, whether an event has to be logged or not, and (2) the event logging cost itself. The actual overhead depends on the frequency of event occurrence. Of course, the actual values for “cost of one event check” and “cost of logging one event” depend on the underlying hardware (CPU, RAM, and secondary storage).

For the applications we had in mind when designing the WMK, like understanding the Windows operating system or analyzing the dynamic behavior of arbitrary applications, we believe the overhead is acceptable.

## 4 Case Studies

This section presents some detailed case studies which demonstrate the applicability of the Windows Monitoring Kernel. First, we show how the WMK can be used to analyze the synchronization behavior of Windows applications. Afterwards, the WMK is used to monitor Windows kernel activities.

The methodology of using the WMK is always the same: replace the original kernel, reboot the machine, run some applications, get the WMK logfile (by either rebooting again or by using the `NtWmkFlushLogfile` system call), and finally use some of the analyzing tools to process the logfile.

### 4.1 Monitoring Windows Applications

A common problem in complex multi-threaded applications is the analysis of their synchronization behavior and the analysis of whether a potential deadlock may arise. Given only a static documentation or the source code of an application, it is difficult to figure out which thread waits for which synchronization object and which other threads wait for the same synchronization object. This behavior is monitorable at a systems’ runtime.

Basing on our wait event monitoring, we created a tool that analyzes and visualizes dependencies concerning synchronization objects acquired by different threads. For convenience, we used a simple consumer-producer application which synchronizes using a mutex object. Figure 3 demonstrates the resulting wait-for dependency graph.

Inside this graph, an ellipse denotes a process object, it contains the executable file name and unique process ID. A diamond denotes a waitable object, e.g., semaphores,



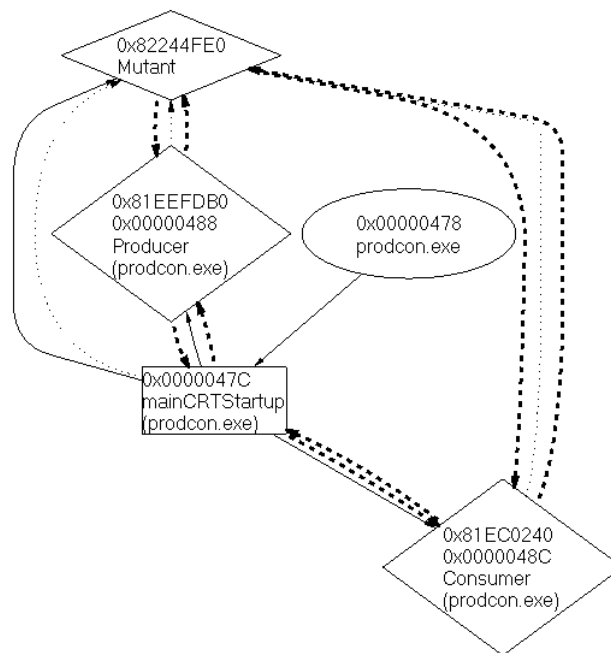


Figure 3: Producer/Consumer synchronization using a mutex

events, or threads. It contains at least the object address and the object type. If the diamond denotes a thread object, the thread function name (or thread start address if no symbol files are available), the executable file name and the unique thread ID are displayed as well. A thread object which is not used as a synchronization object is denoted as rectangle.

There are three types of directed edges connecting the nodes: (1) a solid line shows “create” relationship, i.e., between threads and synchronization objects, or between threads and other threads, (2) a bold, dashed edge shows “wait” relationship, i.e., a thread waits for the specific synchronization object, and (3) a thin, dotted edge shows “release” relationship, i.e., a thread releases a semaphore or an event by calling `ReleaseSemaphore` or similar Windows API functions.

To reduce the diagram complexity and to allow for focusing on the synchronization relations, all synchronization objects which are not part of a “wait” or “release” relation between multiple threads are omitted. Such synchronization objects could relate to local procedure calls or (maybe unused) callback functions. Our example concentrates on intra-process synchronization between concurrent threads; with enabled global logging the inter-process synchronization can be visualized as well.

Figure 3 shows the example consumer-producer application. The main thread creates two other threads which execute the `Consumer` or `Producer` function respectively. Additionally, the synchronization object (`Mutant`) is initialized. Now, the producer and the consumer thread both wait for the mutant object, for assuring mutual exclusion. Also, the main thread (executing the `mainCRTStartup` function) waits for both child threads to finish their execution.

The Wait-for Graph can be created by connecting the WMK to the specific application that should be observed. The process handle is given to the logging infrastructure

via a system service call. Note that this connection can even be established at runtime when the application has already started.

Other diagram types might also be useful: In section 4.2.1 the process-thread diagram is described. Thread-priority diagrams or Scheduling diagrams can also be derived from WMK logfiles but are not presented in this paper. Further types of diagrams might be useful depending on the data classes acquired from the WMK infrastructure and are subject to future work.

The WMK logfile can be processed by reporting and profiling tools which derive values such as average waiting time for specific synchronization objects, or average duration of different system service calls. Applying these tools to complex server systems such as the JBoss application server is also subject to future work.

## 4.2 Monitoring the Windows Kernel

The WMK logging infrastructure is available early in the system startup phase. Therefore, the boot process and other activities can be monitored. The next section demonstrates how the Windows kernel boot activities can be monitored and analyzed by the WMK tools. Afterwards, the Windows scheduler thread quantum lengths are compared for different workloads. Finally, we briefly present how the WMK was used to analyze an anomaly in the Windows kernel timer management.

### 4.2.1 Boot Process

The Windows operating system is an example of a mature, complex system. The WMK allows to trace the (operating) system behavior starting very early in the boot process. To monitor the system the WMK must be globally enabled. This allows to trace the system startup phase and inter-application dependencies.

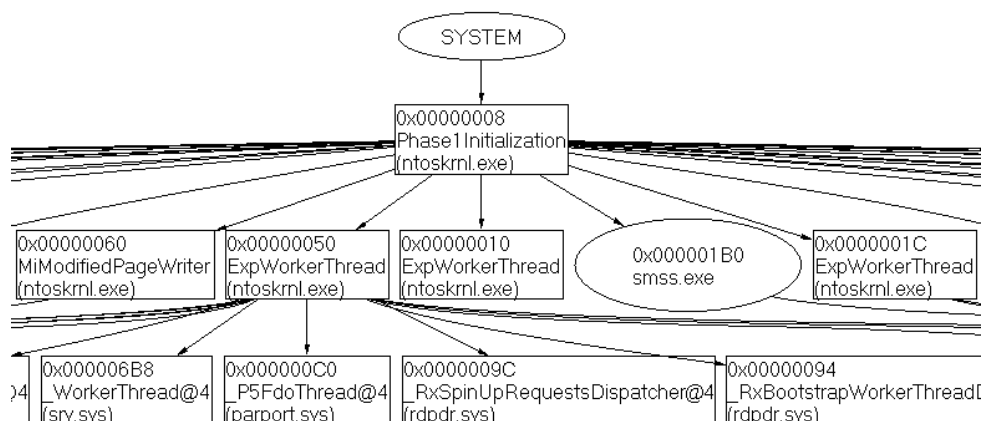


Figure 4: Windows boot process - small snapshot showing the thread/process relation

As an example, figure 4 shows a section of the process/thread relation of the Windows boot procedure. Processes (ellipses) and threads (boxes) are shown. A graph

Filename
\\??\ACPI\FixedButton#2&daba3ff&0#{4afa3d5 ..
\\??\Root#dmio#0000#{53f5630e-b6bf-11d0-94 ..
\\??\Root#ftdisk#0000#{53f5630e-b6bf-11d0- ..
\\??\STORAGE#Volume#1&30a96598&0&Signature ..
\\??\Volume{d059368a-6504-11db-a41d-806e6f ..
\\DosDevices\C:
\\Device\HarddiskVolume1
\\??\STORAGE#Volume#1&30a96598&0&Signature ..
\\Device\RawDisk
\\Device\RawCdRom
\\Device\NetWareRedirector
\\Device\LanmanRedirector
\\Device\Harddisk0\Partition0
\\Device\Harddisk0\Partition1
\\ArcName\multi(0)disk(0)rdisk(0)partition(1)
\\SystemRoot>LastGood
\\SystemRoot\System32\ntdll.dll

Table 4: First files accessed during the boot process

edge illustrates the hierarchy dependency. In this way it can be analyzed which thread or process is started by which thread.

For processes, the graph shows the executable image file name and the unique process ID. For threads, the unique thread ID and the thread start address is shown. If possible, the thread function name is extracted from the application debug information (PDB files).

You can see that the Windows system process (at the top of the graph) starts the `Phase1Initialization` thread which starts all other system threads and initializes the whole system.

Analyzing the process-thread creation can be useful to get a high-level overview of the components of a monitored system. Such an analysis can be the starting point of further analysis, e.g. considering synchronization behavior.

The WMK CreateFile event can be used to analyze the files which are accessed by the Windows kernel during the boot process. Table 4 shows the results.

The very first file the Windows kernel uses is related to the ACPI (Advanced Configuration and Power Interface). A *fixed feature button* is an ACPI concept which relates to “Power” or “Sleep” keys on the PC box or keyboard. Supposedly, some handler routines are initialized and registered for such buttons.

Subsequently, the disk information is read and different devices are opened. On our test system, the devices `NetWareRedirector` and `LanmanRedirector` could not be opened successfully by the booting kernel.

Afterwards, the kernel tries to read the `\\SystemRoot>LastGood` file which contains the last known system state which actually booted successfully. This file was also not

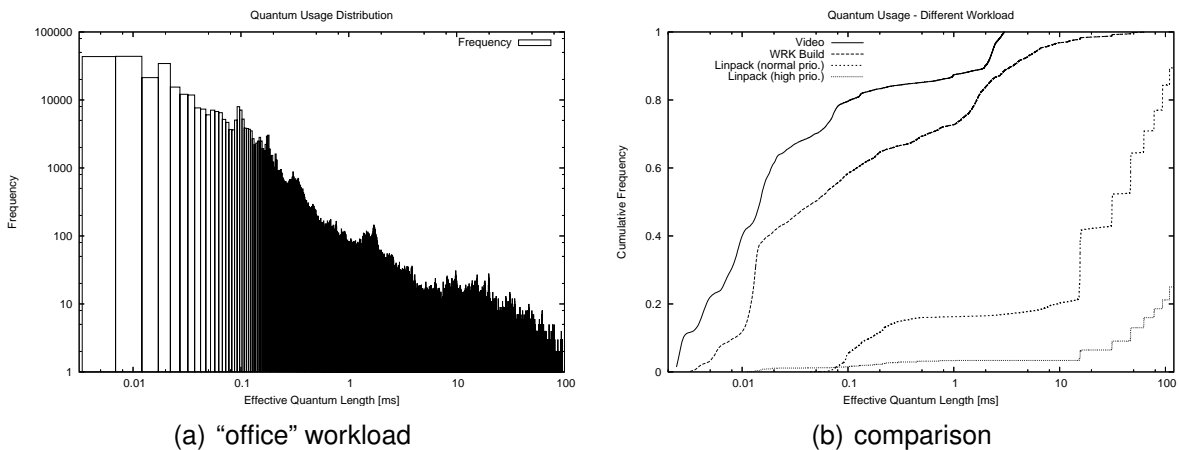


Figure 5: Quantum Length

present in our test run and the file could not be opened.

Finally, the `ntdll.dll` is opened and loaded into the memory. From now on, user-mode applications can be initialized and executed.

The list of accessed files can be a good starting point for a deeper analysis of the Windows boot procedure. Even for application monitoring the accessed files list can be helpful for troubleshooting, e.g., find missing DLLs or configuration files.

#### 4.2.2 Analyzing Quantum Lengths

In [4] the *effective quantum* (= the time an activity was actually allowed to use the CPU) of different workloads was determined on the Linux platform using KLogger. The WMK can be used to repeat the experiment on the Windows platform.

Three different workloads were executed under WMK observation: (1) the WRK was compiled as described in the evaluation section, (2) a 30 seconds video was displayed using the Windows Mediaplayer 10, and (3) as pure computational workload the Linpack benchmark was executed.

Figure 5(a) shows the distribution of effective quantum lengths when using a WMK observed system as workstation (e.g., for reading email or doing text processing) for some time. More than the half of all thread activity was processed in less than 1ms. This observation accords to the measurement presented in [4].

On our test system the maximal quantum which could be used by a thread was 180ms. This value can be calculated as follows: the distance between two clock interrupts (15ms) multiplied by the number of quantum units (36 as it is a Windows server system [6]) divided by three (because three quantum units are subtracted at every clock interrupt).

In figure 5(b) the effective quanta of the different test workloads are compared using a cumulative distribution function (CDF) diagram. Playing a video or building the kernel leads to a similar distribution of the effective quantum lengths. Almost all thread activity can be done in less than 1ms.

The CPU bound Linpack benchmark was executed in two different modes: (1) as

Count	Activity
252	(csrss.exe)
142	ExpWorkerThread
98	ExpWorkerThreadBalanceManager
	<i>446 preemptions omitted</i>
1	MiModifiedPageWriter

Table 5: Linpack Preemptors

“normal” application with the Windows standard priority of 8, and (2) with “time critical” priority set by the `SetThreadPriority` API, leading to a priority value of 15.

Running at time critical priority the Linpack benchmark was able to use more than 100ms effective quantum in 75% of all cases. At normal priority this fraction decreases to only 10%.

Another point to mention are the steps in the Linpack CDF graph: the distribution “jumps” at 15ms steps. This leads to the assumption that there are system activities using timers which wake up at 15ms intervals.

The WMK logfile can be used to extract all threads which preempt the Linpack benchmark application and to do further analysis. The available PDB files with debug information are used to resolve the thread functions. We did this additional analysis for one exemplary Linpack benchmark run at normal priority.

Table 5 show the analysis results: different system activity is executed in a relatively high frequency. Due to incomplete PDB files the `csrss.exe` activity which has preempted the Linpack benchmark 252 times could not be identified. The `csrss.exe` updated the graphical user-interface for character-based applications, therefore we assume that the task is Windows GUI related. Threads executing the `ExpWorkerThread` function do different work on behalf of the executive component of the Windows system, e.g. load drivers at boot time. The Windows balance manager gets activated once every second and has preempted the Linpack benchmark 98 times. Even the modified page writer (writes modified memory pages to the page file) has preempted the benchmark.

The conclusion is that the defined maximum quantum length has nearly no influence on the behavior of the executed applications. Even CPU bound workload is preempted at a high frequency which prevents it from using a whole quantum. The quantum length can be used in its entirety more often if activities are executed at a higher priority level.

### 4.2.3 Windows Timer Expiration

In this section we present how we used the WMK to verify the existence of an anomaly in the Windows timer management that we concluded from analyzing the particular source files. Therefore, we first briefly introduce the anomaly we discovered and argue about possible consequences, secondly we describe how we used the WMK to instrument appropriate parts of the kernel, and, finally, we present a solution to the problem.

**Windows Timer Management** In most COTS operating systems, timers are organized in a list of timer objects sorted ascendingly to their due time. Doing so enables efficient checking whether a timer has expired or not: the operating system simply has to check against the first due time in the list. In the Windows Server 2003, timers are organized in 512 lists of timers. These lists are called the timer table. In which list a timer is inserted depends on its due time: the due time divided by the tick count granularity modulus the number of entries in the timer table—in our case 512—determines the index of the timer list. The distribution of timers in such a fashion favors the insert operation of a timer in the timer list.

The problem we figured out is that the timer table consists of 512 entries. In Windows, each timer has a header of the type `DISPATCHER_HEADER` that contains, among others, a field named `Hand`. This field is used for timers to indicate the particular timer list the timer belongs to. For design reasons, this field only has a width of 8 bits; thus, the range of valid values lasts from 0 to 255. As the timer table contains 512 timer lists, in our particular case, this field might suffer an overflow. We scanned through the Windows sources to find a location where that field is populated for some reason. We finally found it in the only two functions that are invoked to remove a timer from its list. The functions are called `KiRemoveEntryTimer` and `KiRemoveTreeTimer`. In both functions the `Hand` field is evaluated to determine the list head of the timer's list. If the timer is the last one in the list, the head of that list must be modified to indicate that the list is empty. The list head contains a field, called `DueTime`, that holds the due time of the first timer in the list. If the list is empty, by definition, the most significant 4 bytes of this 8 byte value must be set to `0xffffffff`. If the original index was greater than 255, `Hand` refers to the wrong list head, so if this wrong list is not empty but the actual list is empty, the actual list head keeps its due time, i.e., a past due time. (However, the timer is removed properly anyway, as the kernel provides a list modification API the list head is not essential for.)

Windows considers these timer list heads for checking timer expiration. In the interrupt service routine (ISR) for the clock interrupt, Windows checks the current timer list head, depending on the current tick count value whether its due time is less than or equal to the current time. As mentioned above, if the index of the current list is greater than 255, it may happen that the head of an empty list is not marked as appropriate. So, the due time of such an inconsistent list head remains a value pointing to a time in the past. And although the list is empty, Windows invokes timer expiration routines. Nonetheless, the inconsistent timer list head is repaired when a new timer arrives at that list.

**Experimental Verification** All the issues mentioned above have been identified by pure code reviewing. Of course, we wanted to see whether our hypothesis about inconsistent timer list heads might happen at run-time of the system or whether we missed some important fact (which is absolutely possible due to the amount of source files).

We used the WMK to instrument appropriate code locations responsible for DPC handling and timer expiration handling: `KiTimerExpiration` is responsible for dequeuing an expired timer of its queue and scheduling further DPCs or asynchronous procedure calls (APC) if any, and `KeCancelTimer` that is responsible for canceling a timer, i.e.,

Timer event	Occurrences	Index	%
Expiration	110848	0..511	67.81
Cancellation	2606	0..511	1.59
Re-Expiration	50024	256..511	30.59

Table 6: Tracked timer expiration events during the Windows build process.

simply removes it from its list. These are the only two functions that invoke one of the suspicious `KiRemove*Timer` functions. We instrumented in such a way that we could track timer expiration events, timer cancellation events, and events that were caused erroneously executed due to an inconsistent timer list head. We called such events “re-expiration” events. Table 6 contains the observed results.

In total, we measured 163478 timer related events during the boot process of the Windows OS and 10 builds of the Windows Research Kernel. 30.59% of those events are caused due to the inconsistent timer list head. In addition, we also determined the index value (by calculation) of the expired timer. All index values recorded for re-expiration events were in range 256 to 512. We consider this as proof for our observation.

**Proposed Solution** We propose a solution that removes the existence of re-expiration events completely. Therefore, we modified both functions `KiRemoveEntryTimer` and `KiRemoveTreeTimer` to not evaluate the index value stored in the `Hand` field of the timer but to compute by invoking `KiComputeTimerTableIndex`. The function is invoked every time a timer is removed from the list, in our particular case, 113454 times. Let us assume, that there are at least 10% re-expiration events, which is in our experience a reasonable lower boundary. In that case, invoking `KiComputeTimerTableIndex` is sensible, if the costs for executing `KiTimerExpiration` are at least 10 times as much as for `KiComputeTimerTableIndex`. Indeed, preliminary measurements prove that the costs for executing `KiTimerExpiration` are actually higher than a factor of 10. So, we conclude deploying our solution as sensible.

## 5 Related Work

On Windows operating systems, the major comprehensive analysis tool at operating system level is *perfmon* [6]. *Perfmon* queries built-in Windows performance counters and generates diagrams that help system administrators to identify bottlenecks in the operating system. Performance counters have to be queried by a Windows API, restricting the frequency of the requests. For example, it is impossible to trace every context switch, because the context switch counter might already have changed while the result of a query is transferred to the monitoring thread.

Better monitoring facilities are provided by the Windows Management Instrumentation (WMI) and the Event Tracing for Windows (ETW) tools [6, 12]. The general concept of WMI is the separation of event provider (= instrumentation point) and the tracing

tools (= event consumer). The ETW “NT Kernel Logger” is implemented as WMI event provider.

The ETW toolkit is part of the Windows driver development kit and provides a unified logging infrastructure for (kernel-mode) drivers and (user-mode) applications. It can be enabled/disabled at runtime. Therefore, ETW is better suited than the WMK for monitoring production environments.

For a better comparison to the WMK we repeated the WRK build test as described in section 3 and measured the overhead of the ETW tool. With 50 tests, we determined the average duration of a single WRK build as 167.17s. This results in a slow down of 0.86% if the ETW tool `tracelog` is started in “NT Kernel Logger” mode. Additionally, we specified the following command line parameters: `-nonet` to disable logging of networking events and `-UseCPUCycle` to get more precise event timestamps.

However, the ETW and the WMK are not really comparable:

1. ETW aims at driver developers who want to optimize their driver implementation and system administrators who want to detect faulty components or want to resolve performance problems. The WMK primarily aims at researchers who want to investigate operating system or application behavior.
2. Because of the first point and because of the fact that the ETW has to be applicable in production environments, the ETW environment is more generic and more complex to program. The WMK interface is simpler: just insert the instrumentation code and recompile the kernel.
3. The ETW kernel logger interface is not extendable: there is a small set of predefined event providers, e.g., context switch, thread creation, or file access. Therefore, the WMK offers more possibilities for Windows kernel experiments. The timer expiration analysis or analyzing system service calls is not possible with ETW.
4. The 0.86% overhead were measured with an event frequency of only 4700 events per second. The WMK can handle around 13k events per second for 1% overhead as shown in the evaluation section. At 4700 events per seconds the WMK has an overhead around 0.4%.

In short, for monitoring scenarios in production (server) environments the WMK is not an option: a reboot with a modified kernel is just not possible. In research environments, however, the WMK shows a better performance due to a more lightweight integration into the Windows kernel. Furthermore, the WMK provides a better extensibility regarding available instrumentation points.

In the Linux and UNIX domain, event tracing tools were discussed in the past: the Linux Trace Toolkit (LTT) [13] was created to instrument the Linux operating system at source code level. It also provides an efficient event logging infrastructure but requires a user mode daemon to transfer logged events to the file system. This implies that the buffer containing logged events can be dumped to the file system at a pretty late stage in the boot process. To overcome this drawback, a bigger buffer size is required to guarantee that no events will be lost. As a per-processor buffer approach is chosen,



this approach does not scale well, especially on multiprocessor systems. The same might hold for KLogger proposed by Etsion et.al. [4]. Also, KLogger focuses more on kernel developers to improve system performance while WMK is aimed to improve understanding of applications and the Windows kernel.

Another, yet more flexible approach is taken by *DTrace* [2] which allows dynamic modification of kernel code in order to instrument it. It instruments the kernel by replacing the opcode at a specified instruction address with a jump instruction to an instrumentation routine. The ability to add/remove events at runtime eliminates any overhead induced by DTrace when event logging is disabled. However, DTrace is not more flexible with regard to arbitrary kernel events: The locations where instrumentation code can be placed are fixed inside the kernel. Also, the usage domain of DTrace differs from that of the WMK: DTrace is a tool for administrators in data centers and should help analyze bottlenecks and identify misconfiguration. In contrast to the WMK, it is not intended to facilitate the detailed understanding of processes going on inside the kernel.

## 6 Summary and conclusion

In this paper, we presented the Windows Monitoring Kernel, a customized version of the Windows Server 2003 operating system extended by an efficient fine-grained event logging infrastructure, that provides means for logging variable-sized, arbitrary kernel events. We achieved a small overhead induced by the WMK; the slow down is only about 6% at a considerable high event rate of approximately 60,000 events per second. At event rates typical for other instrumentation frameworks, such as ETW, the overhead is around 0.4%.

We demonstrated how the WMK can be facilitated for analyzing operating system behavior under certain aspects: we investigated the Windows boot process, measured effective quantum lengths, i.e., how long can a thread use a processor before it gets preempted, and experimentally proved the existence of an anomaly in the Windows timer management. The WMK has proven to be highly useful for a fine-grained analysis of applications behavior on the Windows platform. The analysis may include, for example, synchronization behavior or profiling.

We provide web access to the WMK for the research community through [11] and will further develop the infrastructure by adding new types of observable events.

## 7 Next steps

Currently, we prepare a technical report describing the WMK in detail. This report is intended as a manual for WMK users and developers. Furthermore, a case study is being prepared in which the WMK is applied to a complex server infrastructure consisting of web server, database system, and different other applications (e.g., external CGI scripts and applications).

The WMK can be used for fine-grained monitoring of system activities. These activities can be dynamically adapted with concepts as described in [9, 10] depending on the observed behavior and depending on other properties (e.g. service level agreements or quality of service requirements).

## Acknowledgments

The WMK was presented at the workshop on “Applied Program Analysis” [8] as a cooperation with Alexander Schmidt. Part of this work on the “Windows Research Kernel” was funded by Microsoft Grant No. 15899.

## References

- [1] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magic for request extraction and workload modelling. In *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.
- [2] Bryan Cantrill, Michael W. Shapiro, and Adam H. Leventhal. Dynamic Instrumentation of Production Systems. In *USENIX Annual Technical Conference, General Track*, pages 15–28, 2004.
- [3] Microsoft Corporation. Windows Academic Program: Windows Research Kernel. <http://www.microsoft.com/resources/sharedsource/Licensing/researchkernel.msp>.
- [4] Yoav Etsion, Dan Tsafir, Scott Kirkpatrick, and Dror G. Feitelson. Fine Grained Kernel Logging with KLogger: Experience and Insights. In *Proceedings of the EuroSys 2007*, pages 259–272, March 2007.
- [5] Andreas Polze and Dave Probert. Teaching operating systems: the Windows case. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 298–302, New York, NY, USA, 2006. ACM Press.
- [6] Mark E. Russinovich and David Solomon. *Microsoft Windows Internals*. Microsoft Press, 4th edition, 2005.
- [7] Alexander Schmidt. Towards fine-grained SOA instrumentation, April 2007.
- [8] Alexander Schmidt and Michael Schöbel. Analyzing System Behavior: How the Operating System Can Help. In *LNI GI Proceedings 110*, 2007.
- [9] Michael Schöbel. Operating System Abstractions for Service-Based Systems. In *Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering*, 2006.

- [10] Michael Schöbel. Operating System Resource Partitioning for Service-based Systems, April 2007.
- [11] Michael Schöbel and Alexander Schmidt. Windows Research Kernel @ HPI. <http://www.dcl.hpi.uni-potsdam.de/research/WRK>, May 2007.
- [12] Craig Tunstall and Gwyn Cole. *Developing WMI solutions: a guide to Windows management instrumentation*. Pearson Education, Inc., 1st edition, 2002.
- [13] Karim Yaghmour and Michel R. Dagenais. Measuring and Characterizing System Behavior Using Kernel-Level Event Logging. In *Proceedings of the USENIX Annual Technical Conference*, pages 13–26, June 2000.



# A Resource-Oriented Information Network Platform for Global Design Processes

Matthias Uflacker

matthias.uflacker@hpi.uni-potsdam.de

Multinational organizations are pressing hard to develop a culture of distributed collaboration and communication to retain efficiency and effectiveness in a global competition. The transfer and dissemination of domain knowledge between regionally disconnected communities is playing a vital role in this process. Information technology can support knowledge-intensive processes and stimulate remote collaboration by assisting in information sharing activities. In this context I present *d.store*: a resource-oriented platform to deploy information networks in distributed collaboration groups. The platform allows semantic annotation and relation of distributed resources by means of domain-specific taxonomies. Thus, it extends the concept of social bookmarking systems in hypertext networks through the incorporation of information context, encouraging knowledge transfer and inference. While the platform is being designed versatile and multi-purpose, this work focuses on the instantiation of knowledge networks in global communities specialized in human-centered software design.

**Keywords:** Computer-Supported Cooperative Work, Information Sharing, Semantic Networks, Software Design

## 1 Introduction

Being a field of interdisciplinary theories ever since, Knowledge Management (KM) has become a major subject for research in the information technology sector over the past recent years. Yet despite the wealth of analysis, it is not surprising that IT-supported Knowledge Management is still an actively discussed and challenging matter. Possibly, this is due to its multidisciplinary nature, rooted in diverse disciplines such as economics, business, education, or psychology. An isolated, inside-out view on this subject is therefore futile and misleading, forcing software engineers to team up with other stakeholders in the development of Knowledge Management Systems (KMS) [14, 40]. Other reasons for complexity are grounded in the social and cultural diversity in the addressed audience, especially when KM is applied in large and global organizations [2].

However, contributing decisively to the challenges in the application of IT-supported Knowledge Management is its cross-border nature between the digital world of data and the real, intellectual world, where it has to demonstrate value to organizations, groups, and individuals, who are having or seeking knowledge [35]. Satisfying the end-users' needs is clearly an outstanding factor for the success of Knowledge Management Systems and a precondition for knowledge being managed effectively [28, 45]. According to Kulkarni et al. [28], the quality of the knowledge content and the overall quality of the KMS are two key enablers for achieving a rich user experience. In this case, the quality of a KM system can be understood as a degree of meeting expected functional and non-functional requirements. Obviously, a close investigation of the internal work structures within an organization is required in order to get a better understanding of the end-user needs, the system and organizational requirements. To this end, *Communities of Practice* (CoP) have received wide attention for examination and definition of knowledge exploitation in organizations. Communities of Practice describe a group of people within an organization who share a common set of information needs or problems [12]. Rather than being a formal unit inside an organization, they represent informal networks that focus on knowledge and explicitly enable the management of knowledge to be placed in the hands of practitioners [45].

Consequently, knowledge management initiatives must be closely aligned to the specific needs of the targeted communities. It is the group of individuals which determines what knowledge needs to be articulated, shared, and transferred. Due to the nature of knowledge, this is first and foremost an interpersonal process. Any system incommensurate with this process will therefore be rejected by the community and rendered useless.

However, progressing globalization of organizations is raising demands to support knowledge transfer between collaboration groups with the help of information and communication technology (ICT). The work, learning, and innovation capabilities of an organization heavily depend on knowledge being timely disseminated and accessible by other organizational units, regardless of their geographical location. This is true on an individual, intra-community level as well as between communities. Accordingly, Brown and Duguid [6] describe organizations as *communities of communities* and adequately illustrate the interdependencies of organizational knowledge structures within and between heterogeneous groups.

This diversity and informality in community requirements complicates the installation of a valuable and desirable knowledge management system extensively and raises several questions. How can IT systems be aligned with the knowledge and communication needs of collaborating individuals, all being experts in a specific domain of practice? How can existing technology be seamlessly integrated into the work habits of geographically distributed communities in order to support efficient knowledge transfer and dissemination in a meaningful way? The objectives of this work are to address these questions and to present a software platform that attempts to be in line with the demands: d.store.

The goal of the d.store platform is to support collaboration processes in global Communities of Practice. The software provides services for the deployment of community-specific semantic networks on top of arbitrary, distributed information resources in hypertext networks. It combines the idea of social bookmarking with the semantic annotation and relation of concepts to represent common, declarative knowledge structures, which evolve in communities during the pursuit of a common goal. Collaborating parties such as project teams are able to describe, interlink, and share the semantic context of information resources based on domain taxonomies. The information context is formally described through graph structures represented by community-defined vocabularies for concepts (nodes) and relations (edges). The utilization of the provided services can help to organize, browse, transfer and derive knowledge from the spanned resource network.

To further motivate the work, chapter 2 discusses collaboration issues in the context of progressing globalization and distributed work environments in the software industry. Special emphasis is placed on the needs and demands of global software design communities, representing the exemplary target user group for the platform. Chapter 3 presents some key challenges of knowledge management systems. Chapter 4 introduces resource-oriented knowledge networks in the context of IT-supported knowledge management as means to represent declarative domain knowledge in hypertext networks. This provides the basis for the d.store platform presented in chapter 5. A service interface to resource-oriented knowledge networks is defined and demonstrates how the platform can be realized in a HTTP environment. Details on a prototypical service implementation are provided.

## **2 Collaboration vs. Globalization in the Software Industry**

Facing the inevitable change caused by industrial globalization, organizations must revise traditional structures of collaboration, communication and coordination. Globalization needs to be understood as a complex, non-linear, and multidimensional process, affecting communication technology, economics, work organization, culture and civil society in divergent ways [24]. Giddens summarizes this multi-faceted view, when he defines globalization as "*[...] acting and living (together) over distances, across the apparently separate worlds of national states, religions, regions and continents [...]*" [20]. The effects are omnipresent and prevailing in our everyday life, in the way we trade,

work and organize, forcing us to adapt and respond to the occurring change process. In this process, globalization is often perceived diametrically and discussed as both: a threat and an opportunity. In order to survive on globalized markets, multinational enterprises are more and more exposed to the pressure of aligning their business processes to achieve homogenization, coordination, standardization, and performance optimization. Along with markets and competition, costs and government are further factors, which render the globalization of business activities a *must*, rather than a *can do* for many organizations [24, 27].

For the software industry, globalization has several consequences. Software design, and engineering design in general, constitutes knowledge-intensive processes that require well-implemented communication and collaboration practices. Teams spend a considerable amount of time to structure, organize, and leverage information. In this multi-disciplinary setting, a shared understanding of the domain, the requirements, the artifacts, and the design process itself is fundamental [41]. In a global context, the challenges for knowledge transfer are remarkably aggravated. Virtually sharing and disseminating context-specific knowledge across geographical boundaries puts high demands on the information technology employed [24], especially in an interactive domain such as design, where common ground and a shared understanding is essential. At the same time, we see a trend towards globalization in the software itself. Service-oriented software systems are composed of re-usable functional entities, that are potentially distributed across multiple deployment sites and units of responsibility. The information required to construct distributed systems is more and more decentralized, which increases the demand for structuring, organizing, and communicating pieces of information in a collaborating collective.

The complexity of communicating design knowledge within a software project becomes also apparent in the diversity of involved process stakeholders. Team members communicate externally with customers, users and non-users, domain experts and people otherwise related to the design context. Information is gathered e.g. during interviews, contextual observations, or feedback sessions. User researchers, designers, engineers, and business people communicate and collaborate with each other to proceed in the design project and to create design solutions that comply with technical, business, and user requirements. The acquired information needs to be shared with the rest of the team to guarantee a concerted view on the process. In case of globalized organizations, those stakeholders of a design project can be geographically distributed, rendering this process intricate and knowledge sharing difficult to achieve. Typical means to capture, communicate, and share design information electronically are email (attachments), text and multimedia documents, which are stored in distributed locations, often detached from any context. However, this decoupled handling of information resources fails short to provide quick and easy access to important project data and facilitates knowledge loss.

While knowledge sharing within collocated teams happens mostly implicitly and naturally, the geographic distribution of collaboration partners and information sources renders this process an explicit activity that demands for support and control. This must be addressed appropriately by globalized software organizations in order to fully benefit from a decentralized set up. Information and communication technology is able



to connect distributed units of collaborating communities and can provide a channel to establish consensus and shared understanding among development teams.

In order to get a better understanding about the knowledge needs and the general structure of software design teams, a closer view on the targeted user group of the proposed system is required. For this purpose, the notion of *Communities of Design* is introduced. The term refers back to work in the area of collaboration and learning in social environments, such as business departments, government organizations, or learning groups. Wenger has shaped here the concept of *Communities of Practice* (CoP), which are defined as "*groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly*" [44, 46]. Communities develop and share knowledge to pursue a common purpose. They collaborate in problem solving, project coordinations, and discussions.

Communities of Design form a subclass of Communities of Practice with the goal to find an optimal design solution to a given design problem in a user-centered way. Thus, they delineate a collective of collaborating participants contributing to a design process. Consequently, end-users involved in the design process also become members of the community, which further adds to the heterogeneity among process stakeholders. End-users substantially contribute to the group's learning process and provide critical information during user analysis and evaluation phases, a fact which is often neglected in the design of tools to support in knowledge sharing.

A Community of Design is usually centered around a particular project and its existence bounded to the project's term. It generally features a loose composition of a core design team, a more or less fluctuating set of internal and external domain experts, and individuals of an end-user group, targeted by the project. Through this mix of competence and roles, the effects of globalization are likely to result in an increased level of distribution among community members. Communities of Design accumulate and leverage design knowledge in order to achieve a common goal. A shared understanding and effective dissemination of knowledge across community members is crucial for a successful outcome of the design process. The demand for adequate tools to support in the management of data, information, and knowledge in the context of software design is exceptionally present.

### **3 Data, Information, Knowledge**

Before arguing on how community knowledge can be managed with the support of information technology, the meaning of *knowledge* needs to be rendered precisely and has to be differentiated from related terms such as *data* and *information*.

A common conception of the relationship between those terms is founded in the DIKW hierarchy (Data, Information, Knowledge, Wisdom) articulated by Ackoff [1] and earlier references to Cleveland and Eliot [10, 15, 37]. *Data*, the lowest entity in this hierarchy, is a raw, purely syntactic sequence of quantifiable symbols. It can be totally described through a structural, formal representation and thus can be processed by computers [36]. It has no context except for its relationship to other pieces of data. In short, data is an uninterpreted, objective resource that *may* transfer information when

processed and brought into a semantic context.

Accordingly, *information* can be described as data that has been processed, or to be more precise, a meaningful, organized abstraction represented through data, e.g. as text, pictures, or sounds. Information has a semantic context so that conclusions can be drawn from it. This reveals a fundamental challenge for *information systems*: computers, being purely syntactical machines, are unable to process information by themselves. Data, in order to be considered as information, depends on an interpreting agent (e.g. a human being), capable to map a syntactical representation to a semantic, meaningful expression. However, there is a strong dissent in this point: While some are pointing out that e.g. relational database systems process and provide information, Setzer argues that information cannot be processed by computers at all [36]. Not delving further into this philosophical dispute, we want for the rest of this article keep to the definition of information as being an interpreted, meaningful representation of data.

*Knowledge*, according to Merriam-Webster Online Dictionary, is "[...] *the circumstance or condition of apprehending truth or fact through reasoning*" and "*the fact or condition of having information [...]*" [13]. Knowledge is purely subjective and directly connected to the information someone has; the result of internal application and combination of data, information, prior experience and knowledge (apprehension).

Having *context* is a deciding requirement where knowledge ought to be formed out of information. When one has context, one can weave the various relationships of information. The greater the context, the greater the variety of information that one is able to pull from [9]. This is also reflected in the definition given by Davenport and Prusak [11], where knowledge is "*a fluid mix of framed experience, contextual information, values and expert insight that provides a framework for evaluating and incorporating new experiences and information*".

### 3.1 Managing Knowledge, Processing Data

From an organizational point of view, the goal of Knowledge Management is to promote knowledge growth, knowledge communication and knowledge preservation [14]. According to the previous definitions of data, information, and knowledge, knowledge is personal, subjective, and internalized by the knower. It cannot exist objectively and detached [23]. Thus, speaking of a *knowledge base* is a rather imprecise expression in computer terminology. To be even more strict in the definition, the only system where knowledge management takes place is the human mind. What needs to be done in order to meet the goals of Knowledge Management and Knowledge Management Systems is to link knowledge to articulated information and context, which in turn has to be mapped to a data representation. This data can then be stored, processed, and transferred electronically within an organization. Once received, data has to be decoded and can again be interpreted as information. Note that the information that is received by someone interpreting the data later on is not guaranteed to be consistent to the information that was intended to be transferred by the sender. This is where *having context* is playing an enabling role. Provided by the KMS, context helps to correctly interpret the data and to understand its true meaning.

By using and combining this reconstructed information, knowledge can be elicited

by the receiver. Nonaka and Takeuchi [30] try to explain the process as "*Information is the flow, and knowledge is the stock, [...] knowledge is created by accumulating information. Thus information is a necessary medium or material for eliciting and constructing knowledge*". Thus, organizations and communities that leverage information technology to optimize the management of their individual knowledge base need to successfully implement the mapping process from knowledge to articulated information to data and vice versa. Information networks can help in this process by putting data representations in context, which supports data interpretation and re-formation of knowledge.

### 3.2 Information Networking

Information networking defines the activity of classifying information resources and putting these in context by establishing relationships with other resources. By this means, a semantic link between these sources of information is defined. This link provides additional context, helping to restore and extract knowledge from the information. The result is a semantic network of information resources, or a graph of nodes and edges, representing the individual resources and their relations.

The set of object classes and relation types that are applicable in an information network is formally appointed by an ontology. Classes and properties that are defined in this ontology are instantiated in the network, forming a concrete representation of the abstract definition of types and relations. Through this formal representation of concepts, instances, and relations, information networks can be automatically queried to derive and infer knowledge.

In a design context, information networking can be a powerful mechanism to structure and manage knowledge that is acquired during the design process. Design processes can be very heterogeneous. Still, four iterating core activities can be generally identified on a very abstract level. In brief, these are: a fact finding and observation phase in which end-users are analyzed through contextual inquiries; a design phase to creatively unfold innovative opportunities; prototyping for the quick, iterative, and cost-effective testing of ideas with end-users. This testing makes up the last activity in the design cycle: evaluation. Figure 1 depicts this very abstract design process.

With this holistic view on the process, knowledge that is generated in these activities can be likewise grouped into four categories, or *knowledge domains*:

**Contextual Research.** Information originated from problem analysis and contextual end-user research (interviews, observations, contextual inquiries, etc.). Knowledge about user roles, use cases, workflows, user tasks, information flows, etc., is derived from these observations and structured by respective concepts. Prevailing items in this domain are e.g. interview notes, pictures, videos, sketches, or market studies.

**Software Specification.** The progress of the creative elaboration of specifications for the software design is captured in sketches, screen designs, or notes during collaborative activities such as brainstorming sessions. Emerged design decisions in

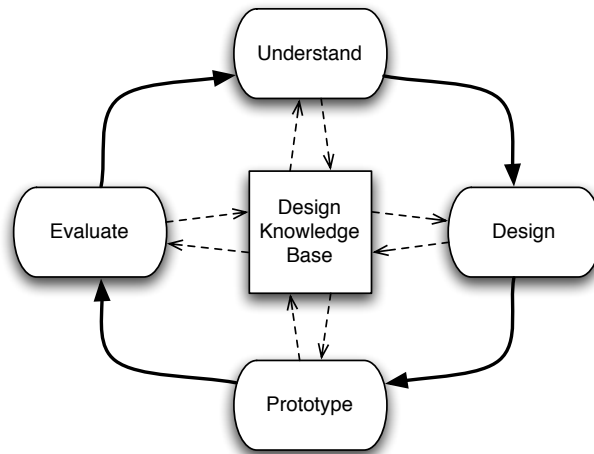


Figure 1: Core Activities in a Design Process

form of abstract and concrete user interface specifications, data models, or business processes are classified and related. Relevant information in this domain is provided e.g. as requirement specifications, task models, scenario descriptions, or stories to capture and communicate design ideas.

**Prototype Solution.** Information about prototypical solutions that emanated from design activities. For a software solution, this might include wireframes, mock-ups, code, web-pages, compiled libraries, or executables.

**Design Evaluation.** Feedback from end-users and other stakeholders, who evaluate the prototype solutions. The information that is acquired from test sessions is captured by the design community in order to serve as input for following design iterations and to refine the problem definition.

## 4 Resource-oriented Information Networks

Resource-oriented information networks classify the distributed information resources that are relevant for a specific project or community. They furthermore define implicit semantic relationships between resources that can not explicitly be represented by the resources itself via hyperlinks. Thus, resource-oriented information networks are knowledge networks which nodes represent Web resources that are identifiable through an unique resource identifier (URI). Edges in these networks express a semantic relation between two nodes. Note that the modeled graph is decoupled and independent from the graph that is spanned by the hyperlinked resources themselves. The resource-oriented information network is operating on a meta-level, providing a context-specific view on the Web and the relevant resources. Statements and relations

can be defined for arbitrary resources which are not under the control of the knowledge worker.

A short introduction into the core concepts of resource-orientation and their relationship to service-oriented systems engineering is meaningful at this point. Service-oriented architectures (SOA) are today's architecture of choice. A service is a mechanism to provide access to one or more remote capabilities to a service consumer. Latter may not be known to the service provider and *may demonstrate uses of the service beyond the scope originally conceived by the provider* [29]. If a provider may not know the actual use of a service, what makes a service a service? What minimum level of functionality must a service provide to be called a service? Resource orientation [17] solves this dilemma by making every entity explicit, not just services. Such explicit entity is called a resource. If one can find a noun for an entity, it qualifies as a potential resource. All restrictions declared for services still hold for resources, i.e. they have an independent life-cycle and a globally unique reference, their interaction style is stateless message exchange.

Resource orientation is driven by the fact that application semantics are very expensive to establish. Thus, in resource orientation semantics are pushed to the protocol level as much as possible, resulting in a universally comprehensible *uniform interface* [17]. To give an example, the uniform interface for requests of the Hypertext Transfer Protocol (HTTP) [16] is described in short:

**GET** Messages labeled as GET have an empty service request and are guaranteed to have no substantial effect within the receiver of such request, i.e. they are *safe* to call. GET responses are expected to be a description of the current state of the targeted resource. These attributes allow GET to act as a universal reflection mechanism, it can be issued without any prior knowledge of the resource. Also, as GET does not alter the state of the targeted resource, the response can be cached. This has great benefits to a distributed architecture and both aspects can be seized without prior semantic knowledge of the targeted resources.

**PUT** Messages labeled as PUT do cause an effect in the targeted resource, but do so in an *idempotent* fashion. An idempotent interaction is defined as replayable, i.e. the effect of  $N$  messages is the same as that of 1. Again, this assumption can be made without any prior semantic knowledge of the resource involved.

**DELETE** Messages labeled as DELETE have the same characteristics as PUT, but imply a negative connotation. Again, the interpretation is solely the responsibility of the receiver, i.e. the effect of a DELETE is not necessarily the deletion of a resource.

**POST** All other types of messages are labeled as POST, i.e. they cause an effect in the receiver and they are not safe to replay. This is a catch-all mechanism for all messages that can not be described by the prior verbs. Without a uniform interface, all messages would be treated like this, losing context free reflection, caching and replayability.

Finally, resource orientation proposes content type negotiation, i.e. sender and receiver can ideally represent and understand the content of a message in several formats thereby increasing the likelihood of finding a format they can agree on.

All put together, globally unique references, the uniform interface, and content type negotiation lower the barrier of entry for any participant to the semantic understanding of the protocol. One of the prime technical reason why the World Wide Web is such a success.

It is the same success that nowadays leads to a massive increase of information that is stored and processed online in the Web. Today's Web application provide the functionality and comfort that was accredited to desktop applications not too long ago. Many Web services exist to provide and share a diverse set of media and information stored therein, ranging from rich text documents, videos, images, and audio. This information is easily accessible and referable by everyone due to the concepts of resource orientation mentioned above. It is only too natural that communities leverage this possibilities for their collaborative practices.

## 4.1 Related Work

Several preceding activities in the area of Web-based knowledge management and collaboration support have been documented. Considered as closely related to the approach presented in this article are the following works.

Davies, Duke, and Sure present *OntoShare*, a knowledge management environment for virtual communities of practice [12]. The ontology-based knowledge sharing environment models the interests of community members in the form of a user profile. Information that is shared by an user is mapped to ontological concepts and transferred to other users whose profiles predict interest in the classified information. *OntoShare* relies on a Java desktop client for its user interface.

*OntoWiki* [22] targets community-driven ontology engineering and ontology usage based on Wikis. It allows the collaborative editing of ontology structures and the creation of instances and relations in a Web-based interface but does not directly imply external information resources. Also, providing parallel functionality for the design and application of ontologies is questionable from a user point of view.

*SOBOLEO* [47] is a tool for social bookmarking and lightweight engineering of ontologies. It is specialized and restricted to a simple subset of the Simple Knowledge Organisation Systems (SKOS) taxonomy. Hence, applicability and semantic expressiveness is limited. However, it allows the annotation and tagging of Web resources based on a structured vocabulary.

Another Wiki-based approach to knowledge management in communities is *Platypus Wiki* [8]. This tool uses ontology models to represent metadata and relations between Wiki pages. Thus, it spans a semantic graph of collaboratively editable Web pages and relationships among them. This graph is restricted to HTML pages generated by the *Platypus Wiki* instance. Arbitrary resources outside the scope of the platform can not be integrated into the graph structure.

The social bookmarking platform *del.icio.us* (<http://del.icio.us>) is also closely related to this work. Its functionality to classify Web resources by means of a community de-

finer vocabulary constructed out of freely definable tags is able to provide and share information based on personal topics and interests. A semantically defined meaning of a tag-based resource classification however is lacking. Furthermore, the context preservation of a bookmarked resource is weak due to the absence of displayed relationships to other resources. Therefore, the construction of a semantic resource network is not possible with del.icio.us.

## 5 The d.store Platform

This chapter presents the functional concepts of the d.store platform. It aims to provide services for the collaborative construction of information networks by using arbitrary resources on the Web as nodes, and community-specific ontologies for the classification and relation of these nodes.

### 5.1 Data Model

The d.store data model can be split up into two major categories. One part of the model is statically provided by the platform through a data schema that has been predefined during design time. Hence, the structure of this part is known beforehand and can be described here in detail. Its purpose is to define fundamental and universal concepts of the d.store application and includes concepts and relations for user and rights management, project information and general configuration parameters. The second category of data models contains a non pre-determinable, usage-specific set of descriptions for concepts and relations for certain domains of expertise. Those data models are utilized by installed projects, yielding in a 1-to-many relationship between custom data models and d.store projects: a domain-specific data model can be used by many projects, whereas each project is related to exactly one domain model. Each usage of a model, meaning the project-specific instantiation with concrete values and state, is managed again on individual project level to isolate general concepts from their project-specific instances.

One of the distinctive characteristics of the d.store platform is that its data model is being fully described in terms of ontologies. With that, it differentiates from such semantic applications, which are falling back to relational data models to implement domain-independent parts of the data structure in a rather traditional approach. The benefit of a pure ontology-based solution is that the application is instantly able to reason over the full set of encoded information, including and across project and application specific data.

A graphical representation of concepts and relations of the ontologies constituting the data model is shown in Fig. 2. Ontologies (respectively namespaces) are visualized by rounded boxes. The d.store ontology is accentuated with bold edges to distinguish from the second category mentioned above. Concepts (i.e. classes) are represented as concave boxes inside of the respective namespaces. Properties are shown as labeled, directed edges connecting two concepts, indicating the particular value domain and range (note that for the sake of simplicity, contingent inverse relationships are

omitted in the graphic). The dotted edges represent context-specific relations and are included here for explanatory reasons. They should be read as 'can have' or 'will have' relationships.

A detailed explanation of the data model and its entities follows next.

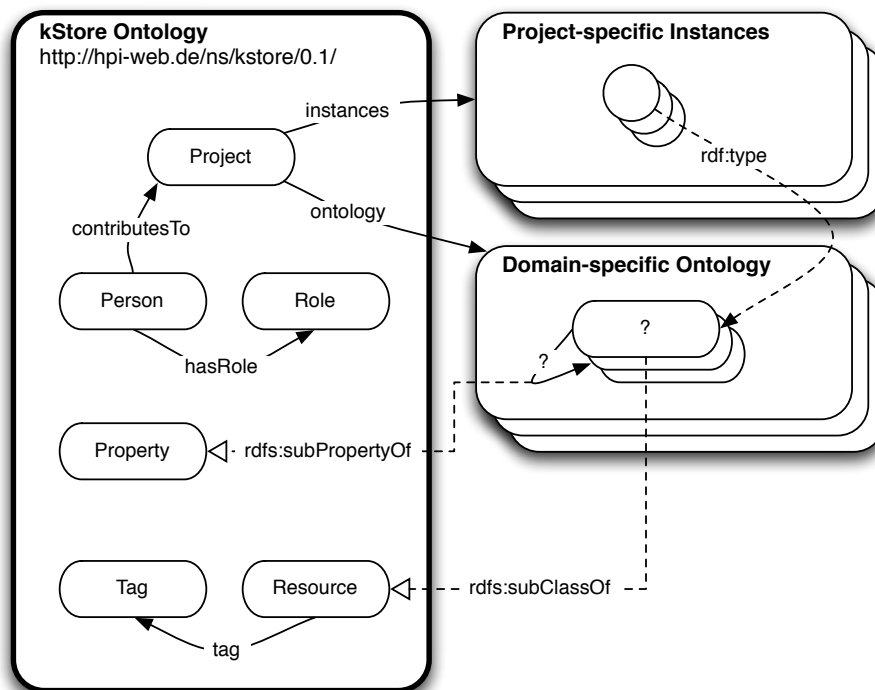


Figure 2: d.store Ontologies

### 5.1.1 The d.store Ontology

A central notion of the platform is the concept of a *Project*. A project comprises the collaboration process of people forming a community with the purpose to achieve a common goal. Hence, *Persons* (tantamount to users of the system) are related to projects, indicating that an individual is a member of this community and contributes data, information or knowledge to the particular project. Each person is assigned to certain *Roles* to allow for different permissions and rights for platform interaction.

As pointed out, every project is assigned to a custom data model, that defines concepts and properties for a particular domain of expertise. This domain describes the thematic area of the project, with the data model providing all relevant entities and dependencies for team communication and collaboration. These usage-specific concepts and relations are defined by domain-specific ontologies (see next section). This relationship between a project and a domain ontology is established via the *ontology* property. Likewise, the individual instantiation of a domain ontology, i.e. the set of concrete individuals and relations identified during the course of a project, is handled in a project-internal namespace, pointed to by the *instances* property.



*Properties* and *Resources* are other first-class entities of the platform. A resource instance resembles a node in the information network of a project. A property defines a class of relationships between two resources. The employment of these two concepts is further discussed in the following section. The platform allows any resource in an information network to be annotated with a set of community-defined tags, represented by the *Tag* concept and property respectively.

### 5.1.2 Domain-specific Project Ontologies

In terms of sharing knowledge, each community and each project has distinct interests, which cannot be fully captured during the design of a system. To address this issue, the d.store platform is designed to work on custom data models that have been defined independently and externally for individual projects and thematic domains. Those models are expressed through ontologies and describe the concepts and dependencies that have relevance for the community from a knowledge sharing perspective. Existing ontologies can be re-used and integrated into the data model at any time to be used as a basis for community collaboration.

Thus, the platform imports and organizes custom, domain-specific ontologies and assigns them to concrete project instances within the d.store ontology. By this means, arbitrary concepts and properties defined by custom ontologies are integrated into the platform and provided to projects. For each project, the community can employ the structure pointed to via the *ontology* property and individually instantiate concepts and define relations.

In order to support re-use and tailoring of existing ontologies, the platform does not rigidly publish the full set of concept definitions to the project community. In many cases, only a subset of an ontology is actually required, or some definitions do not exactly match with existing naming conventions or project peculiarities. To address the issue, the data model considers only those concepts to be employed in a project context, which are subclasses of the *Resource* concept, defined in the d.store ontology. Instances of those concepts hence become nodes of a projects' information network. Using RDFS for example, this subclass relationship can be established through the *subClassOf* property, as indicated in Fig. 2. A synonymic mechanism holds true for properties. In order to mark existing property definitions as utilizable in projects, they have to be defined subclasses of the provided *Property* concept.

Custom subclasses of the *Resource* concept are further annotated with properties provided by the d.store ontology in order to configure their appearance in the platform. The *label* property is used to define a custom d.store text representation for a concept. Similar, the *pluralLabel* property terms a plural representation, used by the platform for a more natural usage of concept names.

On instance level, each individual resource is assigned a unique numeric *instanceId*, unambiguously identifying it among the resources of a project. The type of a resource, meaning the concept it represents, is specified through basic ontology properties such as the RDF *type* property. Note that a resource instance can be of multiple resource types simultaneously. This instance collection is hold in a project-owned namespace, referenced to via the *instances* property.

## 5.2 Platform Services

The d.store platform provides a REST-ful service interface [17] to extract and manipulate information stored in the information networks of the individual projects. In the following, those services are explained in detail.

### 5.2.1 Index Resource

The index resource provides the entry point the d.store platform. It is not related to any project or project ontology and may serve to provide general information about the platform, a list of available projects, or a personalized home page of a user who is logged in. The index resource is returned when the path of the request is empty or equals `"/index"`. Thus, assuming that the platform server is accessible at `http://example.com`, the url to request the index resource can be defined in Backus-Naur-Form (BNF) as:

```
<index-url> ::= http://example.com [ "/index" ]
```

Only *GET* requests are supported by this URL. Note that from now on, the request schema and server domain are omitted and only the request path of the service locations is described using BNF.

### 5.2.2 Login Resource

In order for the system to identify the user and to grant access rights to particular project information, users have to log in with a username and a password. This login functionality is provided by the platform through the resource path

```
<login-path> ::= "/login"
```

A *GET* request for this resource is answered with a HTML form to enter a username and password. *POST*ing these values to the same resource will trigger the authentication procedure, which compares the request parameters with the values stored in the d.store data model. In the case where the user wants to terminate a session the platform accepts requests to the following resource:

```
<login-path> ::= "/logoff"
```

After dispatching a *GET* request to this resource, the user is no longer identified to the system and has no access to any restricted resource.

### 5.2.3 Project Resources

Project resources serve as an abstract or a description of the specific project structure. A HTML representation of the resource might provide statistical figures, information about the community involved, and the ontology that defines the project vocabulary, i.e. concepts and relations of information objects. Resource representations should include hyperlinks to allow easy navigation to all relevant resources of the project that make up the specific information network.

A project resource path is of the form

```
<project-path> ::= "/" <project-id>  
  <project-id> ::= <alphanum>
```

Hence, projects are accessible by appending an alphanumeric project identifier to the server address. *GET*ting these resources results in a representation outlined above. If the Atom format is requested, this resource can serve as a project-specific Atom service document, as defined by the Atom Publishing Protocol [21]. If the project ID can not be resolved by the platform to a known project, a *404 Not Found* response [16] is sent back to the client.

#### 5.2.4 Domain-specific Ontology Concepts

Access to domain-specific concepts (classes) defined in an ontology allows browsing and determination of the project-related list of individuals in the information network that belong to certain concept types. Consequently, concept resources are identified by appending a list of (at least one) concept identifiers to the project resource locator. The general syntax for concept resources is defined as

```
<concept-path> ::= "/" <project-id> "/" <concept-list> [ "/" <tag-list> ]  
<concept-list> ::= <concept-id> { "+" <concept-id> }  
  <tag-list> ::= [ <tag-id> { "+" <tag-id> } ]  
  <concept-id> ::= <alphanum>  
  <tag-id> ::= <alphanum>
```

The reason for accepting requests for more than one concept lies in the nature of ontologies: any individual can be an instance of an arbitrary number of types. Thus, the appendage of further concepts separated by plus signs acts as a logical AND operation. The resource located through the resulting path represents all individuals in a project information network that have *all* given concepts as direct or indirect type definitions. If any of the concepts in the path for a requested resource is unknown and can not be mapped to class of the project ontology, the platform replies with a *404 Not Found* message. Note that in order to be identifiable via this URL schema, an ontology class must be a subtype of the *Resource* concept of the d.store ontology and must have the *label* property of the same namespace set to an alphanumeric value matching the requested concept identifier. Additionally, the value of the optional *pluralLabel* property can also be used to identify the concept.

An optional list of tags can be appended to the resource path and to the list of concept identifiers. Thus, this list further acts as a filter for the set of individuals represented by the requested resource. Identical to the list of concepts, the concatenation of multiple tags is a logical AND operation. Any node of the knowledge graph that is not assigned to all the tags in the list is not considered to belong to the set of individuals represented by the requested concept resource. Contrary to the constraints of a concept list, providing an unknown tag ID in the request of a resource does not result in an error message.

Two HTTP method types are significant for these resources. The dispatching of a *GET* request returns a representation of the set of individual information network

nodes of a project (i.e. resources on the Web) that fulfill the constraints encoded in the request path in terms of concept types and associated tags. This will happen either in form of an HTML document providing a listing of the meta data and relevant hyperlinks for each resource, or via an Atom feed that contains entries to represent the set of individuals.

Sending a POST request to a domain-specific concept resource triggers the affiliation of a specified resource to the set of indicated concepts and assigns to it the list of provided tags. For the information network, this means that a new node is created in the project domain and that appropriate relations to concept types and tags are established for this individual instance. A post request which has been processed successfully is answered with a *201 Created* response message.

### 5.2.5 Project-specific Instances

The information network of a project is built up by nodes that represent resource on the Web. The d.store platform provides meta data about this resource and presents semantic relationships to other nodes in the network. This information is accessible through instance resources. Instance resources can be accessed using the following path syntax:

```
<instance-path> ::= "/" <project-id> "/" <instance-id>  
<instance-id> ::= <numeric>
```

Instance identifiers are project-wide unique numeric identifiers, which are assigned to a resource when it is added to the network. If a requested ID is not assigned, the platform responds with a *404 Not Found* message.

The representation of an instance resource returned in response of a *GET* request contains annotated information and describes the node's proximity in the information network, i.e. its context and relationships to other resources. It presents the collaboratively defined relations and enables the navigation to linked resources, providing meaningful information about the connections.

A *PUT* to a instance resource features an update activity for this node and its relations, according to the message payload sent to the server. Sending a *DELETE* message to an instance resource results in the removal of the representing node and all of its relations in the information network.

## 5.3 Prototype Implementation

Central parts of d.store platform have been prototyped to test the technical feasibility of this approach. The Java 2 Platform (Standard Edition) [25] has been chosen to provide the runtime environment due to the availability and maturity of supporting frameworks. The prototype implementation demonstrates a deployable Web application that runs inside an Apache Tomcat [5] server instance. However, any arbitrary Java servlet container should comply with the requirements. Figure 3 shows a conceptual sketch of the prototype architecture.

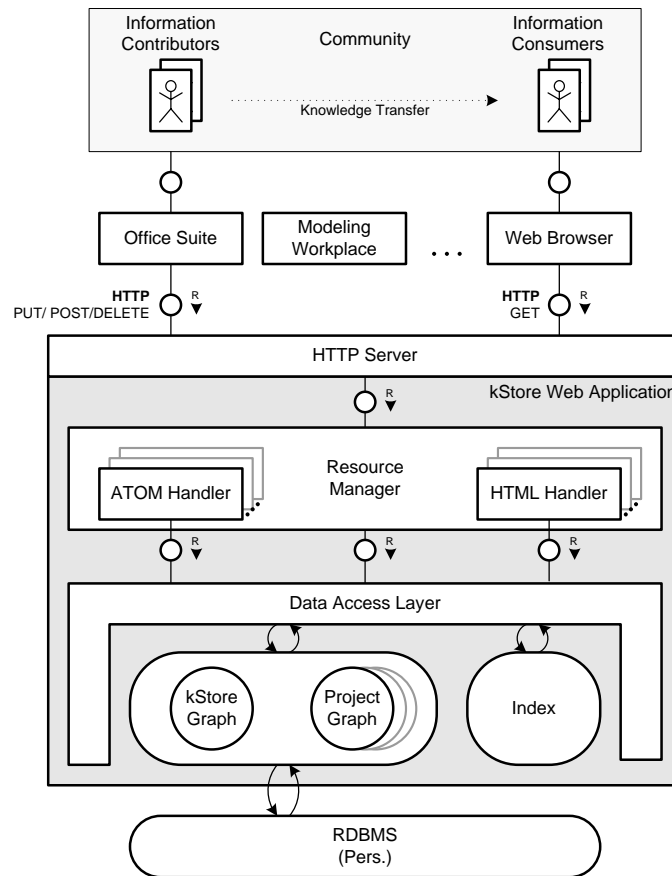


Figure 3: High-level View on the Architecture

In line with the 3-tier architectural schema, the server application can be described by dividing it into application layer and data layer. The data layer implements the data model and provides functionality for convenient access and manipulation to the application layer above. To decouple the application logic from concrete data layer implementations and modeling frameworks, the communication between those two layers is established via Java interfaces and data access objects. The data model interfaces are instantiated with concrete implementations using the dependency injection functionality provided by the Spring framework [18, 39]. Data access objects are available for each concept of the d.store ontology to pass information from and to the data model in an implementation-independent way.

### 5.3.1 Data Layer

The data layer of the prototype is built on top of the Jena Semantic Web Framework [26]. The formal foundation for the description of the ontologies is provided by the OWL Web Ontology Language [42]. More precisely, the sublanguage OWL DL was chosen to enable the application of sophisticated reasoning algorithms for automatic inference. Jena provides a programmatic environment for graph models that adhere to the structure of OWL and includes a rule-based inference engine. Latter one is pow-

ered by the open source OWL DL reasoner Pellet [32,38]. The d.store ontologies have been modeled in OWL using RDF/XML for a textual representation [43], and were imported into the Jena framework. For performance reasons, ontology models are kept and operated on in main memory during run-time. If required, changes to the models (e.g. insertion of new relations or individuals) are flushed to a persistent database in order to prevent data loss. A PostgreSQL [33] database was used for this purpose to provide this persistency layer and to store the ontology graphs in a uniform schema, denoting a list of statements in the form of subject-predicate-object triples.

To provide full-text search functionality on the knowledge network, Apache Lucene [4] is used to index meta data and representation of the resource nodes in the graphs. Note that none of the technologies, which make up the data layer causes dependencies in the implementation of the application logic that is presented next. Information between these two layers is transferred using data access objects, representing d.store concepts such as projects, users, resources, and properties.

### 5.3.2 Application Layer

As an integral part of the application, the resource manager component in Figure 3 handles all incoming HTTP requests that are sent to the server. When processing a request, the component starts with a syntactical interpretation of the resource path that has been requested in order to determine how to handle the request. Based on the URL syntax definition presented in the previous chapter, the component checks whether the path segment of the requested URL conforms to one of the valid targets. If so, the resource manager follows the Model-View-Controller paradigm (MVC) [7] and instantiates a designated controller, responsible to further handle the request. Until then, neither the method nor the payload of the HTTP request have been contemplated.

When instantiated by the resource manager, the controllers are initialized with all the relevant context variables that have been decoded from the requested URL path segment. For example, the controller type `ResourceClassController` holds references to data access objects that aggregate information about the specific project, requested ontology classes, and the appended tag list. Besides this type of controller, several other classes have been implemented to handle requests for projects, individual resource instances, user login, or the index resource respectively.

The controllers are based on classes provided by the Restlet framework [34], which provides lightweight functionality to create REST-ful Web services. Similar to ordinary Java servlets, the process is handled by these controllers through designated method, which are called by the framework depending on the HTTP request method type. To reside with the `ResourceClassController` as an example, this controller provides methods to process GET requests (for retrieving a list of instances for a given set of ontology classes), and POST requests (to assign a new instance to the specific classes).

After having successfully interpreted the syntax of the requested URL and delegated the processing of the request to an appropriate controller method, the payload of the requests finally gets analyzed. Depending on the encoding format of the transmitted data and the result of a server-driven content negotiation [16], format-specific handlers are charged with the final generation of the server response. This is done

with the support of the data layer interfaces discussed earlier, which by this means constitute the model component of the MVC pattern. Note that the message body of GET requests is non-existent, in which case only the content negotiation determines the appropriate handler.

For this prototype, two request/response formats were in the center of interest: The Atom Syndication Format [31] (messages of type `application/atom+xml`), and standard HTML messages of type `application/x-www-form-urlencoded` (request) and `text/html` (response). The decision to support these two formats is primarily grounded in their suitability for different usage scenarios. While client-server communication via HTML is required for human-readable in-browser interaction, Atom is well-qualified to enable the integration of the services in non-HTML clients such as feed readers, office suites or modeling environments. Furthermore, the manipulation of resources using the Atom format is currently in the process of being standardized in form of the Atom Publishing Protocol [21]. The creation and processing of those Atom feeds is done using the Apache Abdera [3] implementation, whereas HTML templates are processed by the FreeMarker template engine [19]. Each of those are therefore forming the basis of the handler-specific view component of the MVC structure.

While the actual presentation of the response is in the responsibility of the client, each representation of a requested instance resource contains all relevant information to describe the context of the resource as described in section 5.2. If the response message is HTML, the meta information and relations of a requested node are easily readable in a browser. In case of an Atom response, the information is encoded in Atom feed entries and is thus interpretable by any compliant client.

## 6 Conclusion

This report has presented d.store, a resource-oriented collaboration platform to capture and access community knowledge in domain-specific information networks. The need for such a platform has been motivated with the increased demand for communication and collaboration support in the design and implementation of service-based software systems. In this context, Communities of Design were introduced to describe the social aspects in global, design-intense processes. With the help of this platform, explicit community knowledge can be represented through the classification, relation, and context of arbitrary information resources in distributed hypertext networks such as the World Wide Web. Information networks have been described as a model to represent graph structures of classified information resources and their relations in a community-defined domain. The platform is based on formal representations of ontologies, defining concepts and relations relevant for a particular project or scenario.

The basic concepts of the platform have been presented and the core services to deploy semantic knowledge networks were outlined. The data model of the application is organized in such a way that it can handle multiple projects simultaneously, each with its own set of vocabulary and instances. This allows for a hosted or centralized management of multiple project information networks, providing one-stop access to organizational information and preserved context of several projects.

The application of this knowledge management approach in software design communities was taken into special consideration. Current efforts are concentrating on the design of such a domain ontology for design activities, in particular for prototyping and end-user evaluation. Prototyping, and evaluation of appropriate user interfaces and interactions is currently in progress. With this first attempt to deploy resource-oriented semantic networks in software design processes, new insights into the nature of information sharing, resource relationships, and used design vocabularies can be expected.

## References

- [1] R. L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989.
- [2] M. Alavi, T. R. Kayworth, and D. E. Leidner. An empirical examination of the influence of organizational culture on knowledge management practices. *Journal of Management Information Systems*, 22(3):191–224, 2006.
- [3] Apache Abdera. Apache Incubator. <http://incubator.apache.org/abdera/>, accessed Sep. 26th, 2007.
- [4] Apache Lucene. Apache Software Foundation. <http://lucene.apache.org/java/docs/index.html>, accessed Sep. 26th, 2007.
- [5] Apache Tomcat. Apache Software Foundation. <http://tomcat.apache.org/>, accessed Sep. 26th, 2007.
- [6] J. S. Brown and P. Duguid. Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation. *Organization Science*, 2(1):40–57, March 1991.
- [7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, volume 1. Wiley, 1st edition edition, 1996.
- [8] S. E. Campanini, P. Castagna, and R. Tazzoli. Platypus Wiki: a Semantic Wiki Wiki Web. In *Proceedings of the 1st Italian Semantic Web Workshop Semantic Web Applications and Perspectives (SWAP)*, pages 1–6, Ancona, Italy, 2004.
- [9] D. Clark. Performance, learning, leadership, & knowledge. <http://www.nwlink.com/~donclark/>. Accessed Sept. 9, 2007.
- [10] H. Cleveland. Information as resource. *The Futurist*, pages 34–39, 1982.
- [11] T. H. Davenport and L. Prusak. *Working Knowledge. How Organizations Manage What They Know*. McGraw-Hill Professional, 2nd rev. ed. edition, 2000.
- [12] J. Davies, A. Duke, and Y. Sure. OntoShare: A Knowledge Management Environment for Virtual Communities of Practice. In *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture*, pages 20–27, New York, NY, USA, 2003. ACM Press.
- [13] Merriam-Webster Online Dictionary. Definition of knowledge. <http://www.m-w.com/dictionary/knowledge>. Accessed Sept. 6, 2007.
- [14] R. Dieng, O. Corby, A. Giboin, and M. Ribiere. Methods and tools for corporate knowledge management. *Int. Journal of Human-Computer Studies*, 51(3):567–598, 1999.
- [15] T. S. Eliot. *The Rock*. Faber & Faber, 1934.
- [16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, The Internet Engineering Task Force, 1999. <http://www.ietf.org/rfc/rfc2616>.



- 
- [17] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair-Richard N. Taylor.
- [18] M. Fowler. Inversion of control containers and the dependency injection pattern. <http://martinfowler.com/articles/injection.html>, Accessed Sept. 20th, Jan. 2004.
- [19] FreeMarker Java Template Engine Library. <http://freemarker.org/>, accessed Sep. 26th, 2007.
- [20] A. Giddens. *Runaway World: How Globalisation Is Reshaping Our Lives*. Routledge; Revised edition, 2002.
- [21] J. Gregorio and B. de hOra. The Atom Publishing Protocol (Draft). Technical report, The Internet Engineering Task Force, 2007. <http://tools.ietf.org/wg/atompub/>.
- [22] M. Hepp, D. Bachlechner, and K. Siorpaes. OntoWiki: community-driven ontology engineering and ontology usage based on Wikis. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 143–144, New York, NY, USA, 2006. ACM Press.
- [23] J. Hey. The Data, Information, Knowledge, Wisdom Chain: The Metaphorical Link. [http://ioc.unesco.org/Oceanteacher/OceanTeacher2/02\\_InfTchSciCmm/DIKWchain.pdf](http://ioc.unesco.org/Oceanteacher/OceanTeacher2/02_InfTchSciCmm/DIKWchain.pdf), December 2004.
- [24] E. Hustad. Knowledge networking in global organizations: The transfer of knowledge. In *SIGMIS CPR '04: Proceedings of the 2004 SIGMIS Conference on Computer Personnel Research*, pages 55–64, New York, NY, USA, 2004. ACM Press.
- [25] Java Standard Edition. Sun. <http://java.sun.com/javase/>, Accessed Sept. 26th.
- [26] Jena Semantic Web Framework. <http://jena.sourceforge.net/>, accessed Sep. 26th, 2007.
- [27] J. K. Johansson. *Global Marketing: Foreign Entry, Local Marketing, & Global Management*. McGraw-Hill / Irwin, Boston, MA, 2000.
- [28] U. Kulkarni, S. Ravindran, and R. Freeze. A knowledge management success model: Theoretical development and empirical validation. *J. Manage. Inf. Syst.*, 23(3):309–347, 06-7.
- [29] C. Matthew, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report Committee Specification 1, OASIS Open, 2006. [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-rm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm).
- [30] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company. How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995.
- [31] M. Nottingham and R. Sayre. The Atom Syndication Format, RFC 4287. Technical report, The Internet Engineering Task Force, 2005. <http://www.ietf.org/rfc/rfc4287>.
- [32] Pellet Open Source OWL DL Reasoner. <http://pellet.owldl.com/>, accessed Sep. 26th, 2007.
- [33] PostgreSQL Open Source Database. <http://www.postgresql.org/>, accessed Sep. 26th, 2007.
- [34] Restlet. Lightweight REST Framework for Java. <http://www.restlet.org/>, Accessed Sept. 26th.
- [35] A. P. Sage and W. B. Rouse. Information systems frontiers in knowledge management. *Information Systems Frontiers*, 1(3):205–219, 1999.
- [36] V. W. Setzer. Data, Information, Knowledge and Competence. 3rd International Conference on Information Systems (CONTECSI). <http://www.ime.usp.br/~vwsetzer/data-info.html>, 2006.
- [37] N. Sharma. The origin of DIKW Hierarchy. [http://www-personal.si.umich.edu/~nsharma/dikw\\_origin.htm](http://www-personal.si.umich.edu/~nsharma/dikw_origin.htm), accessed Sep. 6th, 2007, December 2005.
- [38] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, 2007.
- [39] Spring Framework. <http://www.springframework.org>, accessed Sep. 26th, 2007.
- [40] York Sure, Steffen Staab, and Rudi Studer. Methodology for development and employment of ontology based knowledge management applications. *SIGMOD Rec.*, 31(4):18–23, 2002.

- [41] G. Toye, M. Cutkosky, L. J. Leifer, J. M. Tenenbaum, and J. Glicksman. SHARE: A Methodology and Environment for Collaborative Product Development. In *Proceedings of the 2nd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 33–47. IEEE Computer Society Press, 1993.
- [42] W3C. OWL Web Ontology Language. <http://www.w3.org/TR/owl-features/>, Accessed Sept. 26th.
- [43] W3C. RDF/XML Syntax Specification (Revised). <http://www.w3.org/TR/rdf-syntax-grammar/>, Accessed Sept. 26th.
- [44] E. Wenger. *Communities of Practice. Learning as a social system*. Systems Thinker, <http://www.coi-l.com/coil/knowledge-garden/cop/lss.shtml>, June 1998.
- [45] E. Wenger. Knowledge management as a doughnut: Shaping your knowledge strategy through communities of practice. *Ivey Business Journal*, Jan./Feb. 2004.
- [46] E. Wenger. Communities of practice - a brief introduction. <http://www.ewenger.com/theory/>, 2005.
- [47] V. Zacharias and S. Braun. SOBOLEO - Social Bookmarking and Lightweight Ontology Engineering. *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC), 16th International World Wide Web Conference (WWW 2007)*, 2007.

# Federation in SOA - Secure Service Invocation across Trust Domains

Michael Menzel

michael.menzel@hpi.uni-potsdam.de

Service Oriented Architectures promise an increased responsiveness to changing business requirements. The need to communicate with business partners and customers demands a seamless integration of services offered by foreign organisations. To prevent misuse, scalable security solutions are required to establish trust across all involved business partners. This report outlines approaches and open issues with regard to the composition of services across independent trust domains and introduces a solution to overcome current limitations.

## 1 Introduction

SOA facilitates the interoperable and seamless interaction of service consumer and service provider [14]. With the advent of Web Services as a foundation to realize SOA, additional standards emerged to secure SOAP-based message exchange [20]. However, the exchange of WS-Security messages encapsulating simple security credentials is insufficient when multiple trust domains are involved. Each domain may have a different understanding of security attributes (such as business roles), may support different security mechanisms and may require different information for access control. In addition, users may have multiple accounts registered with different service providers. Several solutions for federated identity management such as WS-Federation [17] or Liberty Alliance [1] have been developed to address these problems.

These specifications provide a good foundation to establish trust and security across independent domains to enable a seamless interaction between users and services. However, it is still challenging to manage these security mechanisms in an enterprise SOA that is based on complex service compositions due to the dynamic negotiation of security attributes at runtime. Automatic service compositions as well as the modeling of workflows under security constraints demand the prior evaluation and verification of security preconditions. The challenges and possible solutions to apply federated identity specifications are outlined in this report.

The rest of this report is structured as follows. The next section provides an overview about the basic concepts of federated identity management and introduces the two most important specifications Liberty Alliance and WS-Federation. Possibilities to apply these specification particularly with regard to service compositions are introduced in section 3. The proposed solutions are based on a general security ontology for SOA that is introduced in the final section.

## 2 Federated Identity Management

### 2.1 Federation Basics

A digital identity consists of several personal attributes that unambiguously represents a related subject. This identity information is usually subsumed by an account in a particular trust domain. Multiple accounts specifying identities in different trust domains can be assigned to one subject. For instance, one account can be related to the subject's company and another to an email provider or an online store. Different personal attributes are assigned to each account with different privacy requirements and security mechanisms. Traditionally, each account is managed independently in each trust domain. This requires the user to reauthenticate when he tries to access a service in another domain.

Federated Identity Management provides a solution for these problems by enabling the propagation of identity information through all trust domains in a federated environment to services and other relying parties. The key concept in a federation is the establishment of trust whereby all parties in a federation are willing to rely on assertions about a set of attributes representing a digital identity. Trust is usually stipulated by contracts specifying the business relationships and technically realized using security tokens that contain the assertions. Dedicated components (*Identity Providers*) in a federation are able to assert identity attributes that can be promoted to service providers acting as *Relying Parties (RP)*.

Based on these mechanisms, solutions for federated identity management are designed to support the following features.

#### 2.1.1 Single Sign On

Single Sign on (SSO) prevents that a user has to reauthenticate multiple times. Different approaches have been realized in different application domains, for instance portal solutions, ticketing systems or local solutions. Federated SSO realizes a ticketing system and promotes the idea of authentication and authorization as a service. Authentication and authorization mechanisms are decoupled from applications and services in the different trust domains by describing a set of claims using security assertions, which can be accepted by all services in the trust domain. For instance, these claims can represent authentication/authorization decisions, can state permissions such as 'the user is allowed to perform orders that are limited to 10.000 Euro' or additional information such as the authentication context. Security tokens are issued by a *Security Token Service (STS)* that will also be called *Identity Provider (IP)*, if this token service supports user authentication. The STS may provide a generic interface for different token types and can be instructed to generate a specific token for a particular service. This would allow the STS to encrypt specific attributes if there was a privacy requirement. Authenticated user can pass these security tokens to the desired services in order to gain access. The tokens are signed by the STS and usually include a time stamp to be valid for a specific period of time.

One possibility to define security assertions is specified by the Security Assertion Markup Language (SAML) that defines three types of security assertions:

- An *authentication assertion* states that an individual has proven his identity to an authentication authority using a particular authentication method.
- An *authorization assertion* states that an authorization instance has granted or denied access to a resource for a limited period of time.
- An *attribute assertion* is a statement signed by an issuing authority that includes claims about subject attributes such as its role.

In conclusion, SAML facilitates flexible authentication and authorization mechanisms realized by autonomous STS servers.

*Single Log Out Mechanisms* are closely related to SSO to enforce that information related to a session can be removed by the involved service provider.

### 2.1.2 Propagation of Identity Information

In addition to SSO/SLO, two other approaches can be determined to enable identity management in a federation.

1. Account Federation - Several accounts can already be assigned to one user in different trust domains within the federation. Account federation (also called Identity Federation) enables a subject to link his account in the identity provider domain with a configured account at the service provider. When a user tries to access a resource in another trust domain, a mapping to the identity in the service provider's domain must be performed automatically.
2. Identity Propagation - A subject may own one account solely that is managed by one identity provider. To access a service in another trust domain it might be necessary to automatically create an account at the relying party. The propagation of demanded attributes is performed automatically as well.

### 2.1.3 Attributes and Pseudonyms

Additional information about a subject can be received from an *Attribute Service*. However, it will depend on the concrete deployment, if the identity provider queries the attribute service to put all the information as claims in a single assertion or if the authorization instance in the service domain queries the attribute service to get additional authorization information.

Another optional service is the *Pseudonym Service* that is capable to map identities to facilitate account federation. Such a service enables scenarios in which a mapping is performed on behalf of the user due to privacy concerns. Another possibility is to perform the mapping automatically when a token is requested for a target service. In some scenarios, the target services are allowed to register their own mappings.

Random identifiers can also be used as pseudonyms to preserve the user's privacy and to prevent a correlation of interactions. In this approach the service has to query for attributes related to the identifier using the attribute service. Access control can be established to protect the attributes and finally the user's privacy.

### 2.1.4 Deployment

There are various possibilities how components facilitating federated identity management can be deployed and configured. For instance, services may need additional information that has not been included in the security token provided with the request. In general, two approaches can be distinguished.

The first possibility is to reject all requests that do not contain all required credentials or attributes. The service can return an error message including a policy constraint that enables the requester to query all the required credentials to access the service again (as described in [10]). Altogether, the requester is responsible to present all necessary information to the service.

An example is shown in Figure 1 that visualizes the interaction between a service requester, the STS and the resource based on a fictional use case comprising the companies Fabrikam and Contoso Systems. An employee of the company Fabrikam wants to bid for a request for proposal. In a first step the federation is established by exchanging federation documents. When the employee requests the service without any security tokens, a SOAP-fault is sent back containing a WS-Policy document. The policy requires a WS-Security token to be present, that is received from the employee's local STS and used for the final service invocation.

The second possibility for a service provider to get required information to authorize a requester is to obtain these attributes from a dedicated Attribute Service in the user domain as described above. Again, the entities that are allowed to interact with this service depend on the concrete federation model.

Moreover, there are different ways, how security tokens can be resolved that are needed to access a desired service. All the steps to resolve the right token can be done by the client or this work can be delegated to the STS. In the first case, the client has to ask the resource for a list of trusted STS, which results in a query to one of the resolved (resource) STS. This STS responds with a list of trusted STS in the clients domain. Finally the client can query the STS in his domain and the resource's STS to get the required token.

The second possibility is to enable the client's STS to automatically contact the resource's STS and to resolve the right security token.

## 2.2 Solutions for Web Service Federation

Several implementations and standards for Web Service Federation exist, but the two major approaches are WS-Federation and Liberty Alliance:

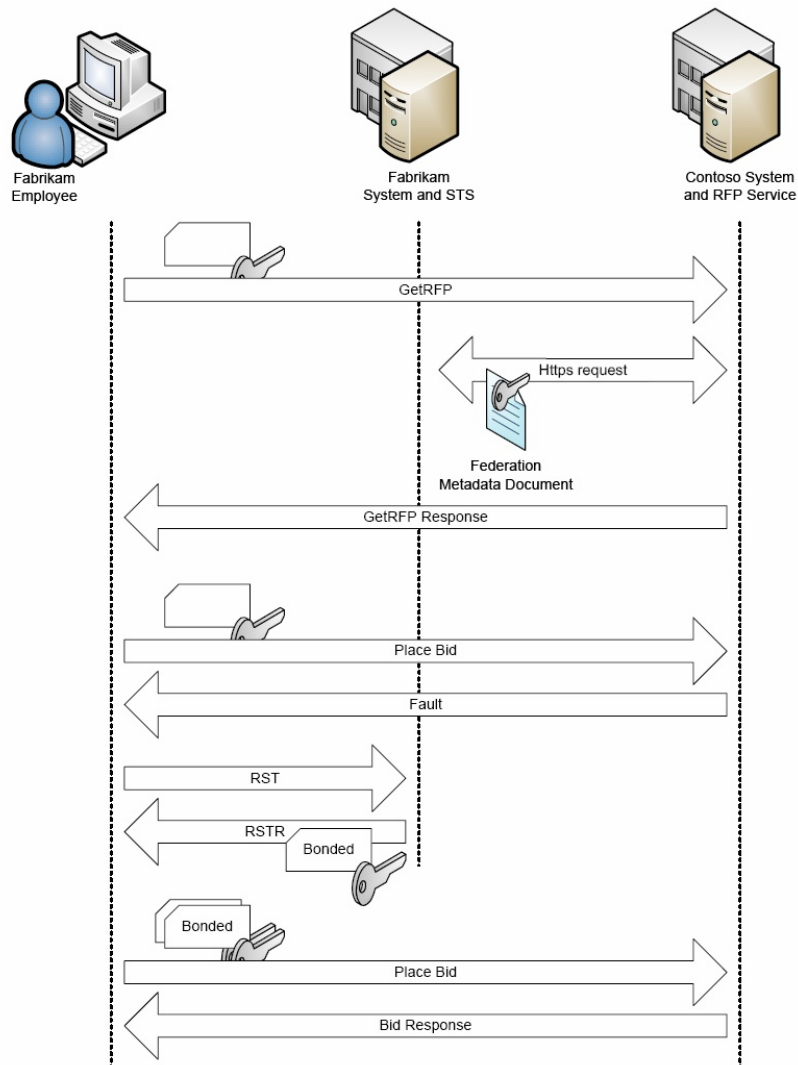


Figure 1: WS-Federation

### 2.2.1 WS-Federation

The *Web Service Federation language (WS-Federation)* [17] defines a framework to federate independent trust domains by leveraging WS-\* Standards such as WS-Security [18], WS-Trust [16] and WS-SecureConversation [11]. This specification provides a model for security token exchange to enable the brokering of identities, the discovery and retrieval of attributes, and the provision of security claims in a Web Service based architecture. The token exchange is based on generic Secure Token Services using WS-Trust, close to the concept introduced above. A meta-data model to describe and establish a federation is introduced as well [10]. Altogether, WS-Federation is designed to enable the use of identity attributes across trust domains to facilitate authorization decisions specified by WS-Policy.

### **2.2.2 Liberty Alliance**

Liberty Alliance provides specifications for federated network identity management that is not just limited to Web Services. This project has been supported by a broad range of companies (Sun Microsystems, Novell, Intel, Oracle, ...) acting in different business areas.

The specification defines a basic framework for federation including protocols, bindings and profiles to enable account federation and cross-domain authentication based on SAML 1.0 (specified in Liberty Identity Federation Foundation (ID-FF)). In addition, bindings for Web Service Federation are defined (Liberty Identity Web Service Framework (ID-WSF)) and a set of standard services (Liberty Identity Service Interface Specifications (IS-SIS)).

In contrast to WS-Federation that can be used to exchange any type of security token, Liberty Alliance is totally based on SAML. However, this federation specification has been merged in SAML 2.0.

### **2.2.3 Comparison**

Solutions for federated identity management enable the seamless interaction between businesses and consumers across organisations. Although the Liberty Alliance and WS-Federation are quite similar from a conceptual view, their implementation and application is different. Microsoft's solution is focused on the interaction between businesses and has already been integrated into their products. Active Directory in Windows Server 2008, Card Space and the Windows Communication Foundation support this specification.

In contrast to WS-Federation, Liberty Alliance is more comprehensive and is particularly focused on the interaction between users and services. The integration into SAML 2.0 will promote the adaption to various products, especially in java-based technologies.



### 3 Federation in SOA

In the previous section, several ways to deploy security components to enable a secure federation of Web Services have been introduced. However, a Service Oriented Architecture that facilitates dynamic service compositions has special requirements that have to be considered.

#### 3.1 Security in Service Compositions

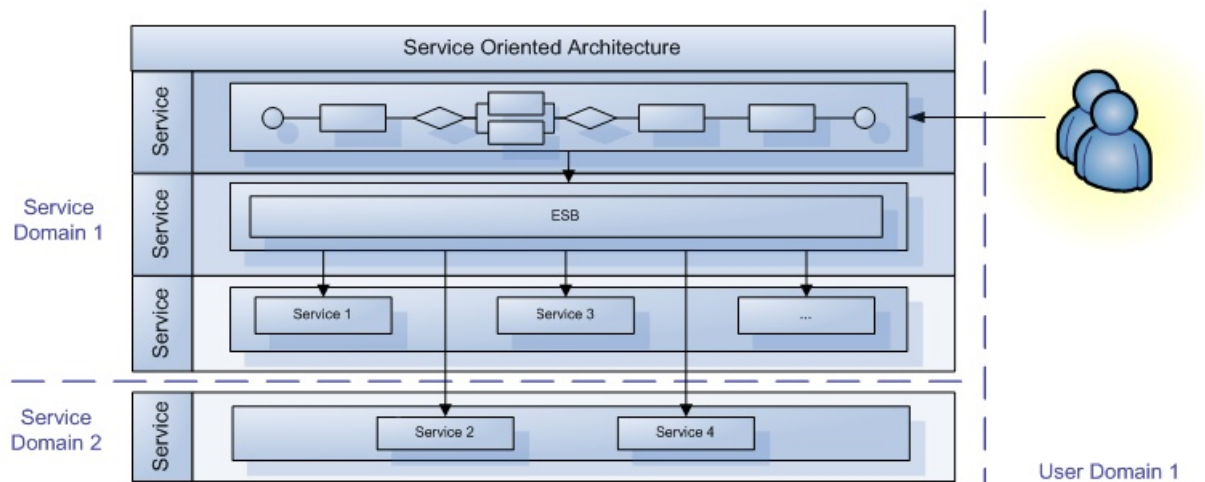


Figure 2: Layers in a Service Oriented Architecture

Service compositions in a Service Oriented Architecture are exposed as a service to the users. The abstract activities in these compositions are mapped to internal or external services that may have their own security requirements expressed as security policies, as shown in Figure 2. Therefore, the security requirement of the service representing the entire workflow depends on the security policies of the basic services. However, negotiating policies between the workflow service and the service consumer would demand the prior calculation of the workflow's security requirements. A straightforward solution is to pass the policy negotiation and to invoke the service providing just a pseudonym. The pseudonym of the service consumer can be resolved by the basic services using the clients attribute service, see section 1.3.

However, it is disadvantageous that each service is required to access the user's attribute service. Due to privacy requirements the negotiation of required attributes may be necessary, but the basic services located in the other trust domain may have no relationship to the client's domain.

The fact that the query to receive necessary attributes can just be performed at runtime is another drawback. The designer of the workflow would not be able to verify that the process can be successfully executed in advance. Dynamic service compositions may be an additional reason that requires the security preconditions of the workflow in advance to enable a proper matchmaking. The coexistence of different standards is a

further problem, since one trust domain might support WS-Federation while the other domain supports Liberty Alliance.

### 3.2 Decoupling federation protocols

The problems related to the integration of different federation technologies can easily be addressed by the introduction of a federation proxy. The proxy decouples federation mechanisms and encapsulate federation specific protocols, as shown in Figure 3.

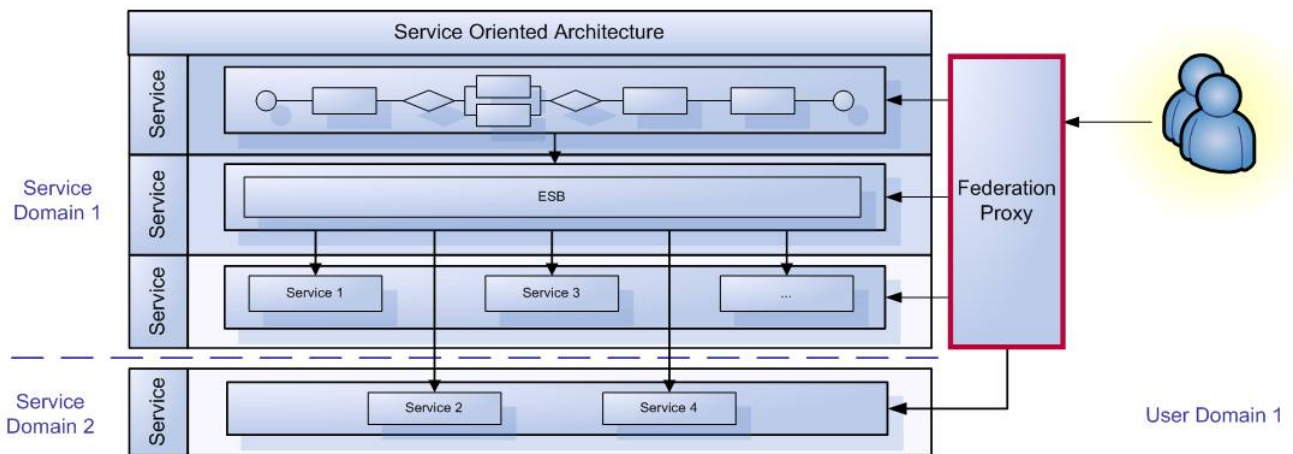


Figure 3: Integration of a Federation Proxy

Moreover, the proxy can act as a STS and attribute service to negotiate between external basic services and service consumers. In particular, the proxy should support all interfaces that are defined by the different frameworks such as WS-Trust. The transformation of security tokens is an important feature, since it might be required to convert proprietary tokens into SAML tokens that are understood by all services based on Liberty Alliance. In addition, a proxy can be used to perform all policy negotiations that might be necessary due to privacy considerations [23].

### 3.3 Workflow Security Preconditions

A further challenge is the prior determination of security preconditions that must be fulfilled to execute a workflow successfully. Security preconditions can describe the security mechanisms that must be supported in order to invoke a service, the required security tokens and claims that must be provided comprising several attributes. Therefore, a security ontology is needed to describe security information and their relationships that must be linked with service descriptions. In general, there exists a broad range of approaches and standards to describe the semantics of services, such as OWL-S [13], WSMO [12] or WSDL-S [2]. All standards use pre- and postconditions to enable services to express their requirements.

Based on these conditions several approaches have been described to calculate the preconditions of semantic workflows. Meyer [15] describes a formal workflow model

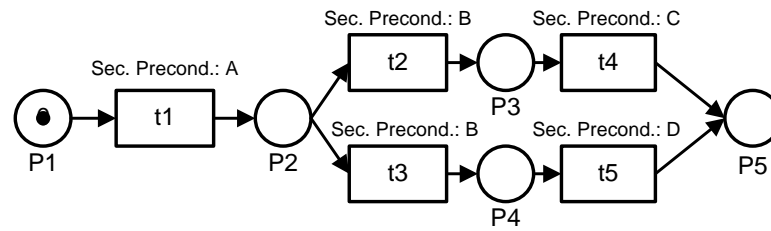


Figure 4: Calculating Security Preconditions

based on petri nets to calculate the preconditions and the effects. Security in dynamic service compositions has been introduced by Carminati, Ferrari and Hung [5]. Their work is focused on semantic matchmaking under security constraints to guarantee a selection of secured services that work in a particular service composition. So far, there is no work that investigates possibilities to calculate security preconditions. However, the approaches presented in former research work about the calculation of preconditions in semantic workflows provide a suitable foundation for the determination of workflow security requirements. An example workflow is shown in Figure 4. Based on the requirements of a single service, the requirements of the workflow can be calculated as  $A \wedge B \wedge (C \vee D)$ .

## 4 Security Ontology

A Security ontology is needed to express the security requirements of services and the relationship among these requirements. Several approaches have been described to define security in semantic web and Web Services [7, 9], but these work is based on simple security annotations for services. A comprehensive model that describes all security aspects including the relationship to policy definitions is missing. This section introduces such a model to identify the relationship between security requirements and specific attributes. This model has been defined in cooperation with Christian Wolter (SAP Research).

### 4.1 Specifying Security Goals

The abstract concept of security can be defined precisely by specifying a set of security goals [19]. Although these goals can be further specialized, subdivided or combined, we will focus on the basic goals in this paper solely:

1. *Confidentiality* provides protection against the unauthorized notice of stored, processed, or transferred information.
2. *Integrity* ensures the properness (intactness, correctness, and completeness) of information (data integrity) and the correct functioning of a system (system integrity) respectively. Transferred, processed, or stored data must not be modified with proper rights and - in economic terms - modifications must correspond to business values and expectations. A system must act in an expected and proper way at each point in time.
3. *Authentication* ensures the credibility of information - such as a claimed identity - by confirming this information as authentic.
4. *Authorization* is the process of granting rights to participants to perform an interaction, for instance to access a resource.
5. *Traceability and Auditing* provide verifiability regarding all performed actions in an information processing system. This can be related to simple logging mechanisms, but also to monitoring as real-time auditing e.g. in intrusion detection systems.
6. *Availability* ensures that data, resources and services, which are needed for the proper functioning of a system, are available at each point in time regarding the requested quality of service.

These goals can be related to various entities of a process-aware information system. These relations among security goals and affected entities are typically described by *Constraints* that are composed in a security *Policy* as indicated by Figure 5.

The basic entity in such a model is an *Object*. We define an object as an entity that is capable to participate in an *Interaction* with other objects. This interaction always

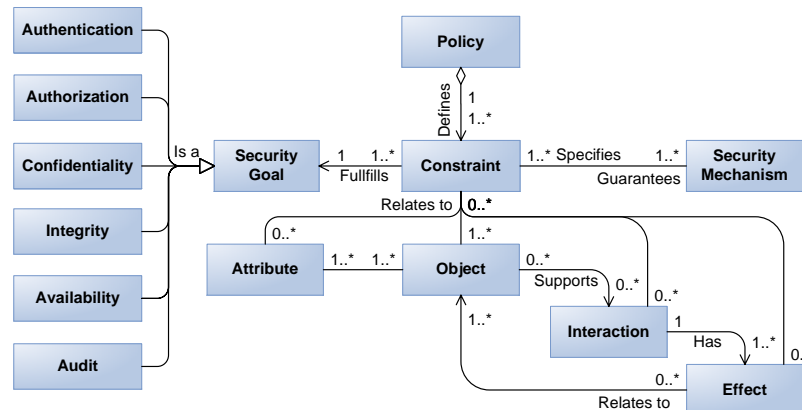


Figure 5: Security Policy Model

leads to an *Effect*, which can comprise the provision of information or the change of state in a system. The effect can, but does not need to be related to the object that initiated the interaction. For example, one object could be an application and another object could be a resource, such as a file. The process of accessing this file would be the interaction resulting in the effect that data in the file is changed or some information is returned to the application.

Each object is related to a set of attributes describing its meta information. For instance, if the object represents a user, attributes, such as name, email address, age, etc. will be assigned. Altogether, policy constraints always refer to a set of objects, a particular set of objects' attributes, and optionally a set of interactions and effects that are related to the objects. Based on these relations, specific constraints for particular security goals can be defined. These specific constraints define requirements for associations between the entities with regard to the particular security goals. In the course of this paper four of six basic security goals are modeled and described subsequently. Namely the security goals authorization, authentication, integrity, and confidentiality.

As shown in Figure 5, constraints specify security mechanisms that enforce or guarantee the defined constraint. For instance, a confidentiality policy usually specifies an algorithm (e.g. DES) that must be used to guarantee this requirement.

## 4.2 Security Mechanisms

In our model a *Security Mechanism* is designed to characterise techniques that are used to enforce security constraints (cf. Figure 6). In general, these mechanisms can be classified as algorithms (e.g. DES), protocols (e.g. WS-Security) or syntax (e.g. XML). The dependencies between these entities and their relationship to *Interaction* and *Effect* are not visualized in Figure 6. However, it provides the foundation to specify a comprehensive ontology for security mechanisms.

Besides security mechanisms, a *Credential* represents another important entity in our model that subsumes evidences used by security mechanisms. A detailed classification of security credentials was presented by Denker *et al.* [8]. In this work they

introduced an ontology that divides credentials in simple credentials (e.g. key, login, certificate) and composed credentials (e.g. Smart Card, SAML, WS-Security Token) that contain a set of simple credentials.

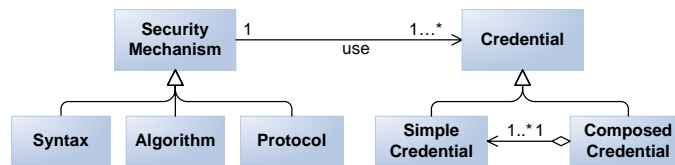


Figure 6: Security Mechanisms Model

### 4.3 Security Constraint Models

Based on the given security policy model (cf. Figure 5), we define specific types of *Constraints*, each guaranteeing one of the security goals listed above. Due to space limitations, the models introduced in this section are related to the security goals *Authorization*, *Authentication*, *Integrity*, and *Confidentiality*. Boxes with a dashed border refer to entities defined in the aforementioned policy and security mechanism models. Each constraint is related to a specific set of entities and define rules restricting particular associations between those entities. These rules must be enforced by security mechanisms and are visualized in our model using dashed arrows pointing to the restricted associations.

#### 4.3.1 Authorization Constraint

A broad range of access control models have been developed in the last decades, defining access control constraints based on particular security information such as the user's role (RBAC [21]) or the user's team affiliation (TBAC [22]). Since all these pieces of information can be considered as attributes of involved objects, the attribute-based access control model (ABAC) can be seen as the most comprehensive access control model, as described in [4].

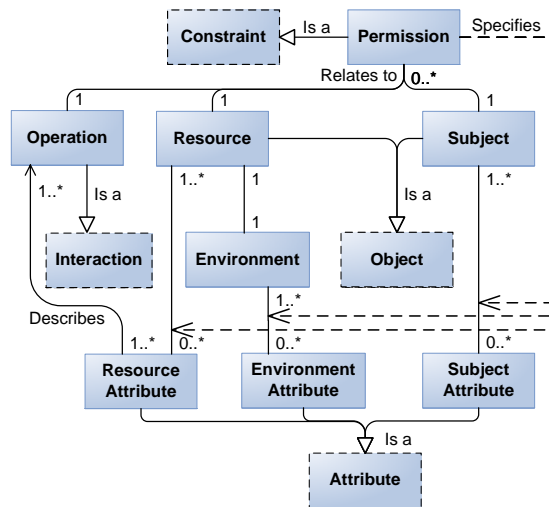


Figure 7: Authorization Constraint Model

In general, there are three entities involved in an access control decision: The subject that wants to access a resource, the resource itself, and an operation that can be performed on this resource. Subject and resource map to objects in our basic constraint model, while operation specifies the interaction. According to ABAC, the access control decision is made based on subject attributes, resource attributes, and attributes of the resource’s environment. Which attributes must be present, is specified by the policy constraint called *Permission* (cf. Figure 7).

**4.3.2 Authentication Constraint**

Authentication enables the credibility of information and is guaranteed by a credential that can be verified using security mechanisms. In our model, information is represented by a set of attributes and can be authenticated by one or more credentials. Since the credential must be assigned to a subject, it is a subject attribute as well. As shown in Figure 8, the authentication constraint *Claim* specifies the relationship between subject attributes and a set of credentials. For instance, the digital identity of a user can be authenticated by providing a name and password (i.e., the user’s credential).

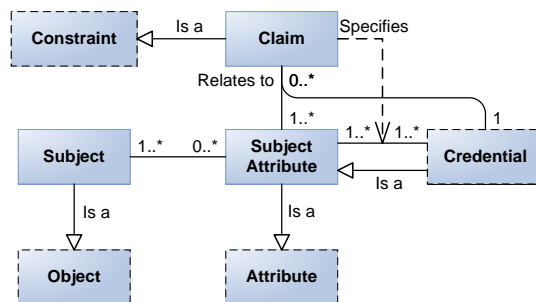


Figure 8: Authentication Constraint Model

### 4.3.3 Integrity Constraint

Integrity ensures that a system must act in an expected way at each point in time regarding transferred, processed, or stored data and the functioning of the whole system itself. In other words, an interaction must have exactly one effect that is specified by the integrity constraint. Such a constraint is called an *Assurance* in our model (cf. Figure 9). The security mechanisms that need to be defined in the assurance to guarantee the integrity depend on the concrete application. For example, WS-Policy [6] can be used as a policy language, specifying WS-Security [18] assurances to enforce the integrity of SOAP communication.

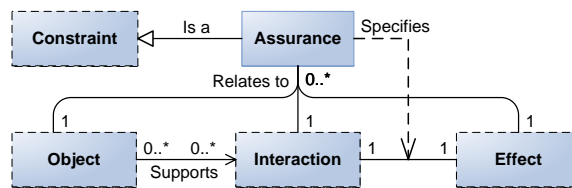


Figure 9: Integrity Constraint Model

### 4.3.4 Confidentiality Constraint

Since confidentiality ensures that authorized subjects are able to notice stored, processed, or transferred information solely, it is similar to authorization, but more specific. The access to the information depends on a credential, the *Shared Secret*, which is shared by all authorized subjects, as shown in Figure 10. Confidentiality is guaranteed by a security mechanism using the shared secret. In our model, the confidentiality constraint, named as *Privilege*, specifies the relationship between subject and shared secret.

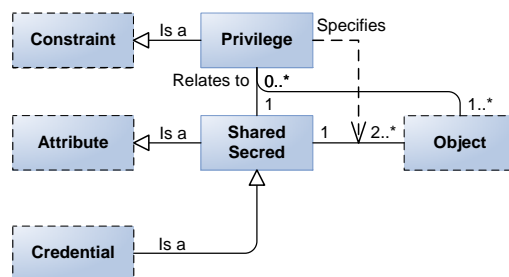


Figure 10: Confidentiality Constraint Model



## 5 Conclusion

The provision and exchange of identity information represents an essential aspect in an federated environment that comprise multiple independent trust domains. Several solutions for identity management already exist that are designed to be used with Web Services. However, it is challenging to apply these solutions in an SOA since current approaches do not consider security aspects at the business process level. The difficulty to manage security solutions independently, complicate the seamless integration of services and result in an increased error-proneness. The design of service compositions under security constraints and the enabling of automatic service compositions require a generic security model. In this report, a model has been introduced that specifies security goals, policies, and constraints based on a set of basic entities, such as *Objects*, *Attributes*, *Interactions*, and *Effects*. The strength of our model is that these entities can be mapped to an arbitrary application domain. For instance, our model can represent security in an operating system, database system, or service-oriented architecture.

This model constitutes the foundation to express security aspects at the business process level. Existing process models are extensible enough to provide sets of attributes and objects that can be mapped to our general security model. Finally, this model provide an ontology to calculate the security preconditions of a workflow, which can be used for policy negotiation with clients from other trust domains. This negotiation can be performed by a federation proxy that encapsulates the federation specific mechanisms.

### 5.1 Future Work

The introduced security model is a promising approach to represent security in different application domains. The relationship between our security model and the OASIS reference model for SOA [14] must be investigated. Since the definition of the SOA reference model is based on the concept of interaction and effects, a straightforward application should be possible. Based on the security model it is necessary to specify a formalized model to define the mapping between the business process layer and our security model to guarantee syntactical and semantical correct security policy implementations. Therefore, the projection to concrete policy languages (i.e. WS-Policy or XACML [3]) has to be defined and it must be proofed that our model can be used to translate concrete security policy implementations to semantic security information used by the business layers. Finally, the negotiation mechanisms in the proposed federated identity architecture need to be adapted to use semantic descriptions. These topics will be addressed by future work.

---

## References

- [1] The liberty alliance project page. Online, 2007.
- [2] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s. Public Draft Specification, November 2007.
- [3] Anne Anderson. Core and hierarchical role based access control (RBAC) profile of XACML v2.0. OASIS Standard, 2005.
- [4] Hai bo Shen and Fan Hong. An attribute-based access control model for web services. In *pdcat*, pages 74–79. IEEE Computer Society, 2006.
- [5] Barbara Carminati, Elena Ferrari, and Patrick C. K. Hung. Security conscious web service composition. *icws*, 0:489–496, 2006.
- [6] Giovanni Della-Libera, Martin Gudgin, and et al. Web services security policy language (ws-securitypolicy). Public Draft Specification, Juli 2005.
- [7] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, and Katia Sycara. Security for daml web services: Annotation and matchmaking. *The SemanticWeb - ISWC 2003*, 2870/2003:335–350, 2005.
- [8] Grit Denker, Lalana Kagal, Timothy W. Finin, Massimo Paolucci, and Katia P. Sycara. Security for daml web services: Annotation and matchmaking. In *International Semantic Web Conference*, pages 335–350, 2003.
- [9] Grit Denker, Son Nguyen, and Andrew Ton. Owl-s semantics of security web services: a case study. *ESWS 2004*, LNCS 3053:240–253, 2004.
- [10] Marc Goodner, Maryann Hondo, Anthony Nadalin, Michael McIntosh, and Don Schmidt. *Understanding WS-Federation*. Microsoft and IBM, 2007 May.
- [11] Martin Gudgin, Anthony Nadalin, and et al. Web services secure conversation language. public draft Specification, February 2005.
- [12] Holger Lausen, Axel Polleres, Dumitru Roman, and et al. Web service modeling ontology (wsmo). OASIS public draft specification, June 2005.
- [13] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing semantics to web services: The owl-s approach. *SWSWPC 2004*, LNCS 3387:26–42, 2005.

- [14] Francis McCabe Peter Brown Rebekah Metz Matthew MacKenzie, Ken Laskey. Reference model for service oriented architecture 1.0. OASIS Committee Specification, February 2006.
- [15] Harald Meyer. On the semantics of service compositions. In *Proceedings of The First International Conference on Web Reasoning and Rule Systems (RR 2007)*, 2007.
- [16] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. Ws-trust 1.3. OASIS Standard, March 2007.
- [17] Anthony Nadalin, Chris Kaler, and et al. Web services federation language (ws-federation) v 1.1. Specification public draft, December 2006.
- [18] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web services security: Soap message security 1.1. OASIS Standard Specification, February 2006.
- [19] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Prentice Hall Professional Technical Reference, 2002.
- [20] Jothy Rosenberg and David Remy. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Pearson Higher Education, 2004.
- [21] Ravi S. Sandhu and Edward J. Coyne. Role-based access control models. *IEEE Computer*, 29:38–47, 1996.
- [22] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented authorization management. In *DBSec*, pages 166–181, 1997.
- [23] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. *disceX*, 01:0088, 2000.



# KStruct: A Language for Kernel Runtime Inspection

Alexander Schmidt

alexander.schmidt@hpi.uni-potsdam.de

Modern operating systems are complex. The behavior of many resource management algorithms implemented in the kernel (memory management, working set management, CPU scheduling, etc.) depends on a multitude of data structures making up the state of the operating system kernel. Although this state is crucial to the performance of user applications, typically it is hidden from the application developer and system administrator

Within this paper, we present KStruct, a domain-specific language that allows for inspecting kernel data structures while the operating system is running. KStruct allows the specification of kernel data structures which is used by a code generator that generates a device driver that accesses these structures. We further demonstrate how consistent access to the kernel data is possible and present the KStruct framework that facilitates the generated device driver for accessing kernel data via a Web browser; this provides the opportunity for integrating other types of information, like documentation or source code, into the rendered contents of kernel data.

Although this paper addresses the Windows implementation of KStruct, the concepts presented herein are more general and applicable to a wide range of current operating systems.

## 1 Introduction

When a software developer tries to learn how an existing software system works, various sources of information might be available: typically, there is product documentation listing the functions of the system and their intended behavior, and possibly design documents explaining the architecture and the rationale for that architecture. In some cases, the source code of the system might be available for inspection. Also, it is typically possible to run the software in a black-box fashion, to see how it reacts to certain inputs.

For the detailed understanding of the system, developers sometimes prefer white-box inspection, *e.g.*, by running the system in a debugger. Using the debugger facilities such as break points and data display, the developer can learn what actual values certain variables take, and can then correlate that knowledge to the source code.

In the case of operating systems, that approach is limited. While some operating systems, and Microsoft Windows in particular, do support debugging of the operating system kernel, such debugging necessarily influences significantly the overall behavior

of the system. In particular, as the system is designed to react to all kinds of events in a timely manner, stopping execution of the system might cause failures which otherwise would not occur (e.g., operations, in particular network interactions, may time-out when the system runs under debugger control). In addition, using kernel debuggers is often a tedious task requiring a already deep understanding of the system as even “simple” interactions may involve large amounts of kernel code.

We present a supplementary approach to studying the behavior of a complex system like an operating system kernel, one which focuses on the data structures only. In our system, inspection of kernel data structures is possible in a running system, using a standard web browser. For example, to find out the list of all processes in the system, the user points the web browser to `http://<machine>/Processes`. This brings up a page (Figure 1) giving a list of all addresses, allowing the user then to navigate to `http://<machine>/Processes/81EB5D88`. That page will display all information about the process (specifically, all fields of the `EPROCESS` structure). Some of these are pointers to other data structures, which again can be inspected by following a hyperlink.

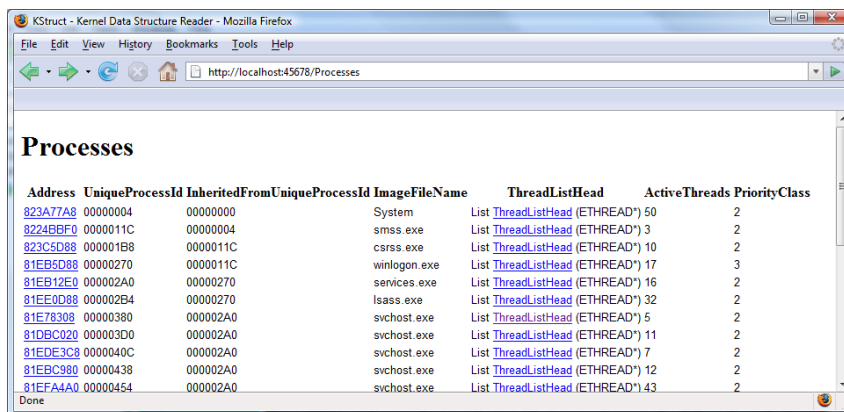


Figure 1: Screenshot of a process list.

In order to get such a display of a kernel data structure like the list of active processes in the system, meta-data is required to gather the desired data. As the main contribution of this paper, we present *KStruct*, a language for defining data structure specific meta-data which is used by a code generator, *KGen*, to generate a device driver capable of locating and accessing the desired data structure and to generate a rendering library that is capable to render contents of a particular data structure to some user interface.

In the remainder of the paper, we first present programming idioms that we identified in the Windows kernel and their representation in *KStruct*, second, we present the grammar of the *KStruct* language, and third, we present the *KStruct* framework the facilitates the *KGen* generated device driver and the rendering library.

## 2 The KStruct Language

The goal of this language is to allow the fully automatic generation of both a kernel-mode device driver and of a user-mode helper library to access arbitrary data structures in the kernel.

Using this language, a specification of kernel data structures can be made that allows a legible rendering of the contents of a given structure at some point in time. To really interpret the structure, the meaning of the various fields must be investigated. In many cases, this investigation just requires inspection of the declaration of the data structure in the kernel. However, for advanced data structures, additional inspection of kernel implementation code must be performed, to clarify the meaning of certain integer values, or to understand how polymorphic pointers need to be interpreted.

For such an approach, the language needs to support constructs to denote the following information:

- For a record-based data structure: what data fields are present, and what names and types do they have?
- For a variable-sized container structure: what is the element type, and how can the complete collection of elements be determined?
- For a primitive value: how should it be rendered in the user interface?
- For the entire state of kernel structures: how can graphs of data structures be traversed, and what are the roots of such traversal?
- How can access to the data be achieved in a consistent manner?

The specification above is deliberately very broad. We found that when studying a specific operating system kernel (such as the Windows kernel) these requirements translate into more precise concepts, as the kernel will have a set of reoccurring programming idioms that allow programmers to better understand the kernel source code. Relying on these idioms also allows to retrieve information from the kernel automatically.

In the following sections, we will present these idioms and how they are expressed in terms of the KStruct language.

### 2.1 Structures

On a first inspection, we find that many data structures in the kernel are defined using the C language, so using the C `struct` definitions is a natural choice for defining the layout of data. Indeed, the KStruct language is an extension to the subset of C which allows the definition of `structs` and `enums`. We will motivate the usage of KStruct in the context of the `EPROCESS` data structure that denotes a process control block in Windows. The specification is shown in Figure 2.

While this has exactly the syntax of a `struct` definition, the semantics of the KStruct definition is slightly different to that of the actual kernel: the KStruct definition may

```
struct EPROCESS {
    KPROCESS Pcb;
    EX_PUSH_LOCK ProcessLock;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER ExitTime;
    HANDLE UniqueProcessId;
    EX_FAST_REF Token;
    PFN_NUMBER WorkingSetPage;
    KGUARDED_MUTEX AddressCreationLock;
    PFN_NUMBER NumberOfPrivatePages;
    PFN_NUMBER NumberOfLockedPages;
    PVOID Win32Process;
    HANDLE InheritedFromUniqueProcessId;
    UCHAR ImageFileName[ 16 ];
    LIST_ENTRY ThreadListHead;
    ULONG ActiveThreads;
    ULONG JobStatus;
    union {
        ULONG Flags;
        struct {
            ULONG CreateReported           :1;
            ULONG NoDebugInherit           :1;
            ULONG ProcessExiting            :1;
            ULONG ProcessDelete             :1;
            ULONG Wow64SplitPages           :1;
            ULONG VmDeleted                  :1;
            ULONG OutswapEnabled             :1;
            ULONG Outswapped                 :1;
            ULONG ForkFailed                 :1;
            ULONG Wow64VaSpace4Gb           :1;
            ULONG AddressSpaceInitialized   :2;
            ULONG SetTimerResolution         :1;
            ULONG BreakOnTermination         :1;
            ULONG SessionCreationUnderway   :1;
            ULONG WriteWatch                 :1;
            ULONG ProcessInSession           :1;
            // further fields omitted here
        };
    };
    UCHAR PriorityClass;
    MM_AVL_TABLE VadRoot;
    ULONG Cookie;
};
```

Figure 2: The KStruct specification of the EPROCESS data structure.



silently omit fields which are considered irrelevant for display, or whose exact interpretation has not been yet discovered. Our implementation will then map the fields given in the KStruct definition to the fields in the kernel declaration of the type.

In addition to these basic constructs, we found that a number of extensions (compared to C) are necessary to capture the idioms used in the Windows kernel. Discovering these idioms is still work-in-progress.

## 2.2 Enumerations

Also borrowed from C is the concept of `enum` type definitions which allow to use symbolic names for integer values. Unlike the usage in C, where the source code specifies the symbolic enumerator, and the compiler converts that to an integer, the KStruct implementation finds the numeric value at run-time, and then looks up the corresponding symbolic name according to the `enum` definition. As an example we present the `NT_PRODUCT_TYPE` enumeration:

```
enum NT_PRODUCT_TYPE {
    NtProductWinNt = 1,
    NtProductLanManNt,
    NtProductServer
};
```

Whenever a field is declared to be of that type, the KStruct framework will resolve the numerical value of that field to one of the symbolic names given above. This greatly improves the readability of a displayed data structure.

## 2.3 Locking

For the device driver generated by KGen, consistency of data is major concern. In particular, when accessing pointers, it has to be ensured that the pointer actually points to a structure of the specified type (*i.e.*, it is not a dangling pointer); else an infamous *blue screen* may occur. Additional requirements for consistency are that atomic values really are displayed with the value that they had at a certain point in time, or that all fields of a struct are displayed with the values that they had at the same point in time.

In the Windows kernel, we found that consistency is achieved through locking. While we are not certain whether this was an explicit design principle of the Windows kernel, so far, in all cases (with one exception) we found that all data available to different threads or processors are protected with some synchronization mechanism, and that the kernel itself consistently acquires the necessary locks before accessing the fields of a structure.

However, there is no uniform name or data type in use that allows to determine what lock needs to be held when locking a certain structure, nor does every structure incorporate a lock. Instead, we found that structures without locks are assumed to be protected by the lock to the container in which that structure lives (either directly embedded, or referred-to from a pointer of that container). To access such a structure, one must first find the container, lock it, and then traverse to the embedded data structure.

These observations led to the introduction of the *lock* and *mlock* keywords. The *lock* attribute must be used for locks that must be acquired before the data structure that contains the lock is accessed. In the `EPROCESS` data structure that lock is the `ProcessLock` field (Figure 2). The *mlock* attribute is used to denote a field that guards another field in the same structure. This is necessary for data structures that are used to implement lists or binary trees and that have only a single lock protecting the whole list or tree. In Figure 2, the field `VadRoot` represents a binary tree which is guarded by the field `AddressCreationLock`. To express the relation between both fields, *mlock* requires the name of the field that is protected by the attributed lock field. The modified specification is given below:

```
struct EPROCESS {
    KPROCESS Pcb;
    lock EX_PUSH_LOCK ProcessLock;
    // ...
    mlock(VadRoot) KGUARDED_MUTEX AddressCreationLock;
    // ...
    MM_AVL_TABLE VadRoot;
    ULONG Cookie;
};
```

The KStruct framework is responsible for invoking the appropriate operating system API to acquire and release the lock. The code generated by KGen will only contain generic synchronization functions following the following naming pattern:

```
BOOLEAN KStructAcquire_LOCK_OBJECT_TYPE(LOCK_OBJECT_TYPE *lock);

VOID KStructRelease_LOCK_OBJECT_TYPE(LOCK_OBJECT_TYPE *lock);
```

`LOCK_OBJECT_TYPE` is replaced by KGen with the appropriate lock type. In the example above, `KStructAcquire_EX_PUSH_LOCK` and `KStructAcquire_KGUARDED_MUTEX`, respectively, are invoked by the generated device driver.

## 2.4 Fast References

To access a field that is declared with a pointer type, it is usually sufficient to just dereference the pointer in the structure using a regular C pointer dereference operation. However, in some kernel structures, pointers are declared with the `EX_FAST_REF` type. This is a Windows kernel mechanism where the three least-significant bits of a pointer hold a count of spare references to an object, so that the reference counter in the object itself does not need to be adjusted as frequently as it would without that mechanism.

As a consequence, a field of type `EX_FAST_REF` cannot be dereferenced directly, because neither the type of the structure the pointer points to is known, nor is can the address be used directly from the field. Instead, the 3 least-significant bits must be masked out from the pointer address.

In KStruct, such a field is declared with the *fastref* keyword, and the type of the field is the actual target type of the pointer. In Figure 2, the field `Token` is such a “fast reference” pointer. Actually, this pointer points to a data structure of type `TOKEN`, thus the pointer type is `PTOKEN`. To express that fact, in KStruct the `Token` field is defined as follows:

```
struct EPROCESS {
    // ...
    fastref PTOKEN Token;
    //..
};
```

## 2.5 Bitfield Unions

In general, interpreting the data in a union type requires information that cannot be inferred from the type declaration, as it is not clear what union alternative is active at any point in time. Different design patterns have been established, *e.g.*, the use of tagged unions, where a separate field in a surrounding union indicates what union alternative is valid. Identifying the specific set of patterns used in the Windows kernel is still a subject of ongoing research.

One specific case that we already identified is a union between a `ULONG` (32-bit) value and a `struct` whose fields are all bitfields. One example is the `Flags` field in Figure 2. This pattern is used quite a significant number of times in the kernel and the reason for that is mainly to ease debugging. In KStruct such unions can be indicated by the *bitfield union* keywords.

The specification of a bitfield union results in a different rendering of the contents of that union. Instead of displaying the numerical value of each bitfield, *e.g.*, `CreateReported` or `VmDeleted`, the code generated by KGen will evaluate each bitfield and renders its symbolic name only if that bitfield evaluates to the boolean value `true`. In KStruct, the specification of a bitfield union is given below:

```
bitfield union {
    ULONG Flags;
    struct {
        ULONG CreateReported           :1;
        ULONG NoDebugInherit           :1;
        ULONG ProcessExiting            :1;
        ULONG ProcessDelete             :1;
        ULONG Wow64SplitPages           :1;
        ULONG VmDeleted                  :1;
        ULONG OutswapEnabled             :1;
        ULONG Outswapped                 :1;
        ULONG ForkFailed                 :1;
        // ... further fields omitted here
        ULONG Spare2                     :1;
    };
};
```

```
        ULONG Spare1                :1;
    };
};
```

## 2.6 Variable-sized Arrays

There are two kinds of dynamic containers in the Windows kernel: arrays and lists. To interpret an array, the size of the array and its element type needs to be known. While the element type is readily available, the size is only declared for an array if the array has a fixed size.

A variable-sized array comes in two forms: as the last element of a structure (a form which standard C calls “flexible array member”), and as a pointer which is to be interpreted as pointing to the first element. In either case, the size must be determined somehow.

As a special case of dynamic arrays, strings are commonly null-terminated in C; the length of the string is nowhere stored. KStruct supports this concept through an explicit *string* keyword (the example is again from the EPROCESS struct):

```
string UCHAR ImageFileName[ 16 ];
```

In this example, a fixed-size storage of 16 UCHAR elements is provided, yet the field is still understood as variable-sized, terminated by a null string.

In the general case of arbitrary arrays, Windows typically follows the idiom of storing the array size in a different field of the same struct. In KStruct, this can be expressed through the *item\_count* keyword; the example is taken from the TOKEN structure:

```
struct TOKEN {
    ULONG UserAndGroupCount;
    ULONG RestrictedSidCount;
    ULONG PrivilegeCount;
    ULONG VariableLength;
    ULONG DynamicCharged;
    ULONG DynamicAvailable;
    ULONG DefaultOwnerIndex;
    item_count(UserAndGroupCount) PSID_AND_ATTRIBUTES UserAndGroups;
    item_count(RestrictedSidCount) PSID_AND_ATTRIBUTES RestrictedSids;
    PSID PrimaryGroup;
    item_count(PrivilegeCount) PLUID_AND_ATTRIBUTES Privileges;

    // ...
};
```

In this type definition, `UserAndGroups` is a pointer to an array of `SID_AND_ATTRIBUTES` structures with `UserAndGroupCount` elements.

## 2.7 Lists

The other dynamic container in the Windows kernel is the list; lists are typically double-linked. Rather than allocating separate memory blocks for the spine of the list and the elements, the fields for adding elements to the list (*i.e.*, the forward link and the backward link) are members of these elements.

In the C declarations of the structures, all link fields are declared with the type `LIST_ENTRY`. Such a field could either be used to make the structure element of a list for similar structures, or the field could act as the head of a list (where an empty list is indicated by the back and forward links pointing to the list entry itself).

What specific type of element is linked in a given list cannot be determined merely from the type definitions, nor can it be determined at run-time. Instead, inspection of the entire kernel source code is necessary to find out what kinds of elements are listed in a list. Fortunately, Windows follows the design principle of only linking like objects in a list, although polymorphic type variations may occur.

To declare the relationship between the list head and the list elements, the *lhead* keyword is used. The `EPROCESS` structure contains such a list, denoted by the field `ThreadListHead`. This doubly linked list is composed of `ETHREAD` structures and each element is linked via the `ThreadListEntry` field of the `ETHREAD` structure. KGen needs this information to appropriately compute the address of each `ETHREAD` component in the list. To specify this list, the KStruct specification is given below:

```
struct EPROCESS {
    // ...

    lhead(ETHREAD::ThreadListEntry) LIST_ENTRY ThreadListHead;

    // ...
};
```

## 2.8 Rendering

In addition to the declarations above which deal with the retrieval and interpretation of information, we found that it is also helpful if the user interface for rendering the data could be derived automatically from our data specification.

For regular structures, this is trivially possible: to render a struct, a two-column display can be used (possibly indented according to the nesting of structures), giving the field name and field value. At the end of the recursion, primitive values need to be displayed. As the list of primitive types that occur in data structures is small<sup>1</sup>, rendering can just enumerate all the types. As there is a larger number of type aliases for integral types, it might be necessary to extend KStruct to support the declaration of type aliases (*i.e.*, to support the `typedef` keyword).

For variable-sized structures, the same approach is possible in principle as well: the array or list could be rendered by displaying all its elements, recursively then displaying

<sup>1</sup>We currently support 18 primitive types

the structures by enumerating all their fields. However, such a display gets very large if the structures are large (e.g., for the `ETHREAD` or `EPROCESS` structures, each having more than 80 fields), or if the number of elements gets large (which, again, happens for threads and processes).

Therefore, we offer rendering hints in the type declaration, allowing the user interface to make an educated choice of what information to display in what manner. So far, we found the need for one such declaration, but we anticipate that we may need more declarations solely for rendering in the future.

The keyword that we defined so far is *listing*; it indicates that a field of a structure should get a column in a tabular view displaying lists of the structure. For example, for `EPROCESS`, we declare that the fields `UniqueProcessId`, `InheritedFromUniqueProcessId`, `ImageFileName`, `ThreadListHead`, `ActiveThreads`, and `PriorityClass` should be included in the listing. As an example, the declaration of `ActiveThreads` then reads

```
listing ULONG ActiveThreads;
```

The choice of listing fields was fairly arbitrary, and user interfaces may decide to ignore these hints.

## 2.9 Grammar

To summarize the previously presented elements of the KStruct language, we present the grammar thereof in this section. Figure 3 contains the grammar for the KStruct language in EBNF notation. As the number of idioms supported by KStruct has not been completed yet, the grammar also might change when KStruct evolves.

## 3 How to obtain KStruct descriptions

In order to inspect kernel data structures, KStruct descriptions of the fields in the data structure must be available. In our experiments, we found that an iterative, systematic process for defining these data structures is possible.

In order to create KStruct descriptions systematically, one needs to start with the C declarations of the same data structures. In many cases, it is possible to copy them literally into the KStruct source file. In translating the new declaration, it might be that certain identifiers are not known, in particular the type names of some fields. In these cases, the following options are available to produce a correct specification:

- the missing field type can be added as well, thus extending the specification recursively, or
- the field with the missing type can be commented-out, in which case it will be suppressed from access, or
- if the field type is a pointer type, its type can be changed to `PVOID` (i.e., `void*`), thus temporarily disabling access to the data pointed-to, until a proper type definition can be provided

```
specification =
    {structdef | enum};

enum = 'enum' ID '{'
    enumerator {',' enumerator}
    '}' ';'
    ;

enumerator =
    ID [EQUALS INT];

structdef =
    'struct' ID '{' {field} '}' ';'

kstruct_attribute = locks | fastref | string | itemcount | lhead;

locks = 'lock' | mlock

mlock = 'mlock' '(' ID ')';

fastref = 'fastref';

string = 'string';

itemcount =
    'item_count' '(' ID ')';

lhead = 'lhead' '(' ID ':' ID ')';

field =
    ['listing', kstruct_attribute | kstruct_attribute]
    type ID [bitfield | {array}] ';'
    | bitfield_union
    ;

bitfield_union =
    'bitfield' 'union' '{'
    field
    'struct' '{' {field} '}' ';'
    '}' [ID] ';'
    ;

bitfield = ':' INT;

array = '[' INT ']';

type = ID | ID '*';
```

Figure 3: The syntax of the KStruct language in EBNF notation.

Once the layout of a structure has been declared, annotations must be added to fields. The set of annotations and their meaning has been elaborated above; authors need to systematically inspect each field to determine whether an annotation must be added.

Particular care must be made in applying locking attributes correctly; if they are incorrect, kernel data consistency might be at risk. In order to determine the proper locking mechanism for a data structure, it is, in principle, necessary to study all uses of that data structure in the kernel. However, we found that the Windows kernel architects followed a convention of including the substring `Lock` in field names used for locking. With that convention, it should be possible to find out what fields need to be locked for access.

While the C type definitions will be sufficient for most cases, they do not work sufficiently for polymorphic fields. In particular, the Windows kernel sometimes uses `PVOID`; in these cases, assignments to the fields need to be studied to find out what kind of data structure the pointer points to.

The definition of the KStruct language is an ongoing process; as further relevant idioms in the kernel data definitions are discovered, the KStruct language itself may need to be extended.

## 4 KStruct Framework

The KStruct language can be used to specify data structures. This specification is used by KGen, a code generator, that generates automatically a device driver and a rendering library that are capable of both accessing a data structure and rendering the contents of the data structure. However, both components need runtime support to be executed and to be queried for certain data structures. The KStruct framework forms a basic execution environment providing the user interface as well as some basic API required by the device driver. This section will briefly introduce the KStruct framework, as shown in Figure 4

A client can connect over the Internet to the KStruct HTTP server specifying an object path for a particular kernel object. The server passes the object path to the KStruct framework that is implemented partly in user-mode and partly in kernel-mode. The framework starts parsing the object path invoking appropriate functions of the *KStruct driver*. If the object could be located successfully, the driver extracts those fields described in the KStruct specification and copies them to the *KStruct buffer*. Finally the *KStruct renderer* renders gathered data into a human readable format. The standard output proposed by this paper is HTML which allows for hyper-linking fields of the actual object with either external information like source code or documentation or another KStruct invocation to follow a chain of objects into the kernel.

### 4.1 Object Path

Object paths are essential for using the KStruct runtime system. An object path is similar to a URI, that uniquely identifies an object in the address space of the kernel.



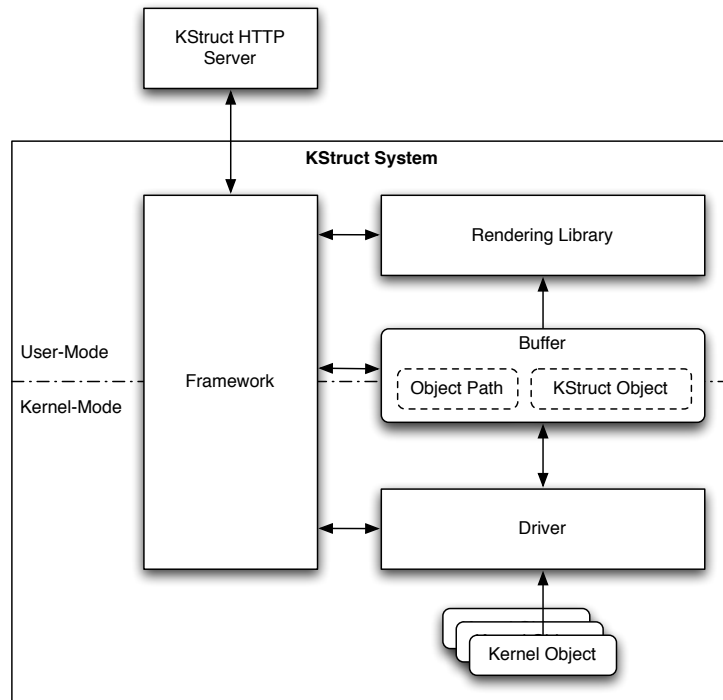


Figure 4: Architecture of the KStruct runtime system.

The object path has a similar structure as the path component of a URL:

```
/<root>/<member1>/<member2>/.../<membern>
```

The *root* item is a special item in the object path: it identifies no instance of a kernel data structure; it rather indicates the function that starts the traversal process through the object graph of the kernel. We discuss root objects later in this section. All other items in the path point to an instance of a kernel data structure. The item name can be either the object ID, *i.e.*, the address of the instance, or the field name that points to or contains the object. For example, refer to the following object path:

```
/Processes/87654321/Pcb/ThreadListHead
```

The root object, `Processes`, denotes the list of `EPROCESS` objects. The next member is the address of a particular `EPROCESS` object that contains a field `Pcb`, denoting a `KPROCESS` object that finally contains a field named `ThreadListHead`. This object is displayed to the user. The type of field name is resolved by the KStruct specification file: Each data structure contains both primitive types, like an `int`, and other structure types identified by their name.

The KStruct system expects such an object path as input for all operations. The object path is parsed item wise; before accessing the next item, it is verified that (1) the item correctly denominates a field contained by the current object, and (2), if an address is specified, that address is valid and reachable. This prevents the KStruct system from crashing the system caused by an invalid memory reference and makes it more robust against erroneous input.

## 4.2 The KStruct Framework

Although we have designed and implemented a generator which allows to generate code in order to access kernel data structures, there are some basic tasks that we moved from the generator to the *KStruct framework* in order to keep the generator as simple and maintainable as possible.

### 4.2.1 Driver Management

Kernel objects can only be accessed when the processor is in kernel-mode as they reside in the kernel's address space. This is an inherent feature common to most modern operating systems. To access them anyway, the running application must also run in kernel-mode, *i.e.*, a device driver. Our generator generates a device driver responsible for the access to kernel objects. However, the OS requires some basic management for the driver in order to properly load it.

### 4.2.2 Root Objects

Root objects are methods provided by the framework serving as entry points into an object graph. They require implementation efforts as their handling is too specific to generate appropriate code. However, we assume that the kernel provides only a very limited number of data structures appropriate to be a root object. Indeed, the number of objects that can be reached from a root object and that can be generated by KStruct Access is far greater than the overhead necessary to manually implement root object handling.

Actually, the framework supports the process list (`/Processes`) as the only root object, but we are currently working on a root object that allows access to objects of the Windows Object Manager.

### 4.2.3 Locking Synchronization Primitives

We decided to remove implementation details from the KStruct code generator. The Kernel Runtime module provides a generic API for well known kernel synchronization primitives. The API has the following signature:

```
BOOLEAN KStructAcquire_LOCK_OBJECT_TYPE  
    (LOCK_OBJECT_TYPE *lock);
```

```
VOID KStructRelease_LOCK_OBJECT_TYPE  
    (LOCK_OBJECT_TYPE *lock);
```

`LOCK_OBJECT_TYPE` is a place holder for a well known kernel synchronization primitive. So far, we discovered the following three types, but we expect to find more:

- `KGUARDED_MUTEX`

- EX\_PUSH\_LOCK
- ERESOURCE

Although locking kernel objects is important for consistently accessing this object, the implementation must guarantee that no deadlock is injected to the system. To prevent deadlocks, the implementation of our acquire and release API calls checks for contention before blocking. If a lock cannot be acquired, `KStructAcquire_XXX` returns `FALSE`. The code generated by `KStruct` checks for that return value and if that is the case, all acquired locks will be released in the reverse order they were acquired. Afterwards, the acquisition process is started again. This might result in increasing response time of the `KStruct` application, but we consider this a fair price for the guaranty to be deadlock free.

#### 4.2.4 Iterators for Collections of Objects

Collections of objects are typically arrays or lists. While it is simple to iterate through a list, iterating through a list of arbitrary objects requires special handling. In Windows, lists are organized as either singly linked lists or doubly linked lists. For these variants an API exists that allows for inserting, removing, and traversing items. The driver framework provides an API for iterating through such a list, and, if necessary, invokes a supplied callback function, generated by `KStruct`. The API for iterating is shown below:

```
NTSTATUS KStructIter_LIST_TYPE(  
    LIST_TYPE *Head,  
    PKSTRUCT_CONTEXT Context);
```

`LIST_TYPE` is either `LIST_ENTRY`, if a doubly linked list should be traversed, or `SLIST_ENTRY`, if an singly linked list should be traversed. The second parameter, `Context`, defines a context for the iterator that contains, for example, the callback function, and the constraints for invoking the callback function.

## 5 Correctness of Data

`KStruct` allows for an insight into the Windows kernel by displaying contents of kernel objects (instances of kernel data structures). By providing this facility it helps understand and derive design decisions of the kernel and, in particular, helps motivate the purpose of certain fields or data structures. It is essential that displayed kernel objects contain correct data, *i.e.*, the kernel object's state is valid in the system context, so that `KStruct` users may not be confused by seeing contradictory information. For example, we consider a process control object – an instance of an `EPROCESS` data structure – that contains both a thread list (field `ThreadListHead`) containing all threads belonging to that thread and an integer field (`ActiveThreads`) denoting the number of threads being in the thread list. It seems obvious that such a displayed kernel object is invalid in the system context, if the number in the field `ActiveThreads` differs from the number of

threads in list denoted by `ThreadListHead`. Although it is trivial to verify the correctness, or *consistency*, of that structure in that particular case, it is the very opposite regarding all fields of the structure or even of all kernel objects. There are a number of publications [2, 6–8, 13, 14] that allow for verifying consistency constraints in the aftermath of an object retrieval. However, we consider a more general approach to assure consistency of retrieved kernel objects, as we consider defining consistency constraints for all data structures, at least for those discovered yet, an unsolvable task because of the maturity of the Windows kernel.

Let us assume the Windows kernel being a shared-memory system and all threads/processes composing the Windows kernel being virtual processors of that system. The shared-memory is composed of kernel objects that are jointly used of all kernel modules. Then there must be an agreement of all processors on the order of accesses to the shared-memory in order to assure the proper functioning of the kernel. We call this agreement on the order of shared-memory accesses the *consistency model* of the kernel, because shared-memory access executed in the order defined by the consistency model transfer a kernel object from a consistent state to another.

In the past, memory consistency models have been well discussed in a number of publications. They can be classified by their strictness as done by Mosberger [11]. We examined several of those trying to identify the most appropriate one. We left out the strictest consistency model, *sequential consistency* proposed by Lamport [9], from our examination as implementing the sequential consistency in software results in unnecessary concurrency penalties. Furthermore, we concentrated on weaker, more relaxed consistency models, like *weak consistency* proposed by Dubois *et al.* [3], *release consistency* proposed by Gharachorloo *et al.* [4], and *Entry consistency* proposed by Bershad and Zekauskas [1]. We found the entry consistency model to be the most appropriate model to be implemented by the Windows kernel. There are a number of reasons for that:

- The Windows kernel distinguishes special accesses, *i.e.*, (synchronizing) accesses to acquire and release a kernel object, and ordinary accesses, *i.e.*, read/write accesses, to the kernel object. Special accesses can be identified after a certain naming pattern, like `XxAcquireYyy`, or `XxReleaseYyy`, where `Xx` denotes the kernel module the synchronizing access belongs to (in most cases “Ex” for the Executive or “Ke” for the Kernel module) and `Yyy` denotes the name of the lock object type.
- The entry consistency model requires explicit association of synchronization objects and kernel objects. We found that most major kernel data structures include such an object or, if the kernel object is a root object, there is an associated global synchronization object. For example, each `EPROCESS` structure contains a field `ProcessLock` representing the lock to be acquired before accessing other fields of the structure. Similarly, the list of processes, denoted by the global symbol `PspActiveProcessHead`, is guarded by the global synchronization object `PspActiveProcessMutex`. To be precise, we also found structures that have no explicit lock object, but they are accessed by instructions of the processor that

guarantee mutual exclusion, *e.g.*, `CMPXCHG` on an IA. Such special processor instructions are a sequence of acquire, modify, and release operations all in one instruction.

- Ordinary accesses, *i.e.*, non-synchronizing, read/write accesses, to shared kernel objects are executed only after the associated lock has been acquired. In this paper, all accesses to the presented structures apply that rule.
- The thread owning a lock receives the most recent update of the associated kernel object from the previous owner. This rule follows implicitly because of the usage of a non-distributed global shared-memory.

The weak consistency model [3] does not match the programming model of the Windows kernel, as it is only the generalization of the entry consistency and release consistency model. The release consistency model [4] does not match, as its programming model does not allow for explicitly associating a synchronization object to a kernel object. Instead, the library implementing the consistency model automatically assigns a synchronization object that is considered appropriate. In a kernel environment, selecting the right synchronization primitive is a question of the design of the kernel as it directly influences the performance of the system.

We designed `KStruct` to implement the entry consistency model and faced a problem that may not occur in the kernel: the danger of deadlock injection by introducing lock dependencies. In general, each lock object guards its associated kernel object for a certain purpose. For example, the lock of the process control object list guards insertion, traversal and deletion of list members, regardless of their particular type. On the other hand, the `ProcessLock` field in the `EPROCESS` structure guards the fields inside the structure, regardless of the circumstances the process control object might be used in. In `KStruct`, we define a dependency between both formerly independent lock objects. If we display a list of `EPROCESS` objects, we need to guarantee that the displayed list has existed at a certain point in time in the history of the OS. That is, the consistency constraints of both the process list and each `EPROCESS` object must be satisfied. Figure 5 shows a simplified object graph of the process list and its `EPROCESS` objects. Circles denote structures, rectangles denote base or primitive types, and hexagons denote lock objects. A directed edge between structures or between a structure and a primitive denotes a “contains” relation. (The process list “contains” `EPROCESS` structures.) A double-line directed edge between a structure and a lock denotes a “guards” relation, *i.e.*, the lock indicated by the rectangle besides the process list “guards” the list. The dotted line is just for grouping purposes; it groups all components that belong to the same structure. To display the whole structure of Figure 5, it is necessary to acquire first the process list lock and then all `EPROCESS` object locks before accessing any `EPROCESS` structure. Thus, the acquire operation for the list of processes requires acquisition of a bunch of locks. Similarly, a release operation on the list of processes requires several release operations on the lock objects of participating objects.

The hierarchy in which the locks must be acquired is derived from the `KStruct` definition file. The order in which the lock must be acquired is the same as if the object graph is traversed in level-order. However, this discovery process must be executed on-line, *i.e.*, at run-time of the system. Each time a new lock object is discovered, `KStruct`

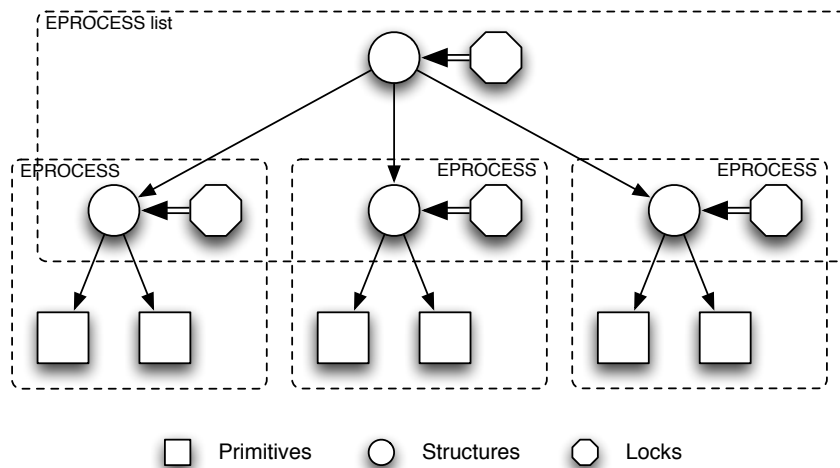


Figure 5: Lock hierarchy of the process list and its contained EPROCESS objects

attempts to acquire the lock. If the acquisition fails, all previously acquired locks will be released and the acquisition process is started again. This is to guarantee the absence of any deadlock injected by KStruct. This, however, might lead to a situation where KStruct is unable to display a certain object because other threads are permanently holding lock objects. Though we never discovered such a situation yet, we are currently working on other ways to achieve the consistency of displayed data structures in a more relaxed manner, for example, by observing the structure over a period of time and determine whether its state has changed. If not the state may be considered as stable and the contents of the structure may be considered as consistent with respect to the OS.

## 6 Related Work

Although we find the KStruct approach unique with respect to how data structures can be examined at run-time of the operating system and to link displayed data structures with other kinds of information, several key aspects were separately discussed in research publications.

### 6.1 Debuggers

In the Windows domain, there are several kernel debuggers available [10, 16]: KD is a command line oriented kernel debugger, while WinDbg also provides a graphical user interface. Both require two PCs for debugging: the host that runs the kernel debugger and the target that is being debugged. Both machines must be end-to-end connected, either via a null modem cable, or an IEEE 1394 connection. In contrast, a KStruct client only requires a Web browser and an active Internet connection.

Also, KD, and WinDbg, respectively, stops the execution of all threads except the debugger controller thread on the target machine which tremendously affects the target

machine with respect to timing and network IO. KStruct, on the other, only affects those threads that try to access the specified kernel object and only for a very limited time.

Finally, debuggers do not guarantee the consistency of any kernel object revealed. The KD, for example, simply forces all threads, except its own, to suspend while it accesses a specified kernel object [16]. This approach prevents the kernel object from changes being done while KD is accessing; however, if one thread was suspended before it was able to complete all modifications, the kernel object is left in an inconsistent state. KStruct implements the consistency model of the operating system. Thus, only threads that are competing for an access to a kernel object are affected while other threads can continue running. Also, if KStruct gains access rights to a kernel object, the consistency model guarantees for the consistency of the kernel object.

## 6.2 Monitoring Frameworks

Monitoring frameworks like Windows Management Instrumentation (WMI) [15] or the /proc file system [5] in UNIX, to name just a few, also allow for an insight into the running kernel. These frameworks are designed to remotely monitor user-mode or kernel-mode applications including some details about the kernel itself. Both lack the opportunity of adding data structures without re-booting or re-building the kernel. While the latter might be possible for the UNIX domain, it is definitely not in the Windows domain, except the Windows Research Kernel [12].

WMI and /proc file system, respectively, do not consider consistency models as they only provide a generic interface to information providers. The way this information is gathered is no matter of interest for the framework; the gathering process must be implemented by a developer. Due to the maturity and complexity of operating system kernels, hand-coding the gathering process is error prone and elaborate. KStruct relieves the developer from implementation details, it only requires a data structure definition. The generated code inherently applies to the proper consistency model of the OS.

## 7 Conclusion and Outlook

In this paper we presented KStruct, a system for consistently inspecting kernel data structures, while the kernel is in use. We also presented idioms supported by KStruct Access, our domain-specific language to describe kernel data structures which is used to generate both a device driver and a user-mode rendering library being able to access and inspect instances of these structures. Although our implementation is based on the Windows Research Kernel, we find most of our presented idioms applicable to other kernels as well; however, applicability to other kernels is still subject to ongoing research.

KStruct allows for accessing kernel objects via a Web browser. Using HTML for rendering kernel objects provides the opportunity for hyper-linking structures with their definitions in source code and other kinds of information, like documentation. We are still evaluating impacts on performance of the operating system while KStruct is in use

to inspect certain kernel objects, but we estimate a far less affection as is caused by kernel debuggers. We also work on enhanced rendering methodologies of data structures that facilitate comprehending data structures and their use throughout the kernel.

As we have demonstrated, the codification of the kernel data structure idioms in KStruct definitions is an ongoing process. We encourage the community to contribute KStruct definitions in an ongoing manner, to support access to data in kernel areas not currently supported in our research prototype.

The KStruct generated code allows for accessing data structure without halting the whole system, a major advantage over kernel debuggers. This is implemented by applying locking schemes or consistency protocols used by the kernel itself. However, accessing arbitrary data structures may introduce dependencies between former independent data structures. To circumvent these inter-dependencies, future research will concentrate on more relaxed consistency protocols that achieve the same correctness of data as ordinary locking does but with only a minimum of locking or even completely without locking.

## 8 Acknowledgments

Part of this work on the “Windows Research Kernel” was funded by Microsoft Grant No. 15899.

## References

- [1] Brian N. Bershad and Matthew J. Zekauskas. Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU-CA-91-170, Carnegie-Mellon University, 1991.
- [2] Harold W. Cain and Mikko H. Lipasti. Verifying sequential consistency using vector clocks. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 153–154, New York, NY, USA, 2002. ACM Press.
- [3] M. Dubois, C. Scheurich, and F. Briggs. Memory access buffering in multiprocessors. volume 14, pages 434–442, New York, NY, USA, 1986. ACM Press.
- [4] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *ISCA '98: 25 years of the international symposium on Computer architecture (selected papers)*, pages 376–387, New York, NY, USA, 1998. ACM Press.
- [5] T. J. Killian. Processes as files. In *Proceedings of the USENIX Summer Conference*, Salt Lake City, 1984.



- [6] V. Kuncak, P. Lam, K. Zee, and M. Rinard. Implications of a data structure consistency checking system, 2005.
- [7] Viktor Kuncak, Patrick Lam, Karen Zee, and Martin C. Rinard. Modular pluggable analyses for data structure consistency. *IEEE Transactions on Software Engineering*, 32(12):988–1005, 2006.
- [8] Patrick Lam, Viktor Kuncak, and Martin Rinard. Hob: A tool for verifying data structure consistency. In *14th International Conference on Compiler Construction*, volume 3443/2005 of *Lecture Notes on Computer Science*, pages 237–241, April 2005.
- [9] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, September 1979.
- [10] Microsoft. Debugging Tools for Windows. <http://www.microsoft.com/whdc/devtools/debugging/default.aspx>.
- [11] David Mosberger. Memory consistency models. *SIGOPS Oper. Syst. Rev.*, 27(1):18–26, 1993.
- [12] Andreas Polze and Dave Probert. Teaching operating systems: the Windows case. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 298–302, New York, NY, USA, 2006. ACM Press.
- [13] Shaz Qadeer. Verifying sequential consistency on shared-memory multiprocessors by model checking. *IEEE Transactions on Parallel and Distributed Systems*, 14(8):730–741, 2003.
- [14] A. Silberschatz and R. B. Kieburtz. The external consistency of abstract data types. *SIGPLAN Not.*, 15(2):64–73, 1980.
- [15] Craig Tunstall and Gwyn Cole. *Developing WMI Solutions*. Addison-Wesley, 2002.
- [16] Richard A. Willems. System and method for collecting system data using record based requests with tag lists and pausing all but one thread of a computer system. United States Patent 6,983,452 B1, January 2006.



# Deconstructing Resources

Hagen Overdick

hagen.overdick@hpi.uni-potsdam.de

## 1 Introduction

There has been a lot of discussion about service-orientated architectures (SOA) [4], lately. A service is a mechanism to enable access to one or more capabilities. The eventual consumers of the service may not be known to the service provider and *may demonstrate uses of the service beyond the scope originally conceived by the provider* [16]. If a provider may not know the actual use of a service, what makes a service a service? What minimum level of functionality must a service provide to be called a service? Furthermore, according to recent studies [9], about two-thirds of all services deployed today are data-centric. Is a memory cell already a service? Resource orientation [8] solves this dilemma by making every entity explicit, not just services. Such explicit entity is called a resource. If one can find a noun for an entity, it qualifies as a potential resource. All restrictions declared for services still hold for resources, i.e. they have an independent life-cycle, a globally unique reference, and their interaction style is stateless message exchange.

Resource orientation is the dominant architectural style on the Internet, as it is the scientific foundation of the World Wide Web [3]. Resources have a globally shared request message classification system, confusingly called *uniform interface*. The idea is, that even without semantic understanding of the messages exchanged, the classification provides additional benefits to the overall architecture. However, up to now, the World Wide Web favors an informal, ad-hoc description of complex resource behaviors. Roy Fielding coined the term "hypertext as the engine of application state" [8], upgrading this ad-hoc fashion from bug to feature; quite a successful feature indeed measured by the success of the World Wide Web itself.

In the course of this paper, a meta model for resource orientation is introduced and design strategies for implementing resources based upon this meta model are outlined. The remainder of this paper is structured as follows: In section 2, an introduction to resource orientation is given. In section 3, an example of a complex resource behavior is shown to illustrate the requirements of a resource-oriented process language. In section 4, a meta model for resource orientation is given to aid the design of a resource-oriented process language. In section 5, BPEL is introduced as a viable candidate for such a language and the necessary extensions are outlined. Section 6 discusses related work and section 7 concludes this paper with a summary and outlook.

## 2 Resource Orientation

As already described in the introduction, resource orientation is a subset of service orientation. As such, it can be regarded as a modeling strategy for services. Instead of a few "gateway" services, with carefully crafted custom interfaces, all entities of the modelled system expose a uniform interface. To illustrate this, let us look at a concrete example. The Hypertext Transfer Protocol (HTTP) [7] defines its uniform interface for requests as:

**GET** : Messages labeled as GET have an empty service request and are guaranteed to have no substantial effect within the receiver of such request, i.e. they are *safe* to call. GET responses are expected to be a description of the current state of the targeted resource. These attributes allow GET to act as a universal reflection mechanism, it can be issued without any prior knowledge of the resource. Also, as GET does not alter the state of the targeted resource, the response can be cached. This has great benefits to a distributed architecture and both aspects can be seized without prior semantic knowledge of the targeted resources. In the physical world, GET request can be correlated to looking.

**PUT** : Messages labeled as PUT do cause an effect in the targeted resource, but do so in an *idempotent* fashion. An idempotent interaction is defined as replayable, i.e. the effect of  $N$  messages is the same as that of 1. In a distributed system, where transactions may not be readily available, this is a great help to error recovery. Again, this assumption can be made without any prior semantic knowledge of the resource involved. In the physical world, this correlates to physical interaction, although replaying the exact same "message" is only a theoretical mind exercise.

**DELETE** : Messages labeled as DELETE do cause an effect in the targeted resource, where that effect is expected to be a termination. Just as PUT, DELETE is defined as *idempotent*. However, as with all messages, the interpretation is solely the responsibility of the receiver, i.e. a DELETE has to be regarded as "please terminate". In the physical world, this correlates to sending a notice of cancelation.

**POST** : All other types of messages are labeled as POST, i.e. they cause an effect in the receiver and they are not safe to replay. This is a catch all mechanism for all messages that can not be described by the prior verbs. Without a uniform interface, all messages would be treated like this, loosing context free reflection, caching and replayability.

The uniform interface tries to lower the barrier of entry to a client and it also includes a characterization of response messages. Thus, interaction with a resource can start purely on the basis of semantic understanding of the uniform interface. If one obtains a resource identifier, the uniform interface provides a minimum level of shared semantics

to start. To increase the likelihood of understanding, both client and resource perform content type negotiation on each request. Content type negotiation honors the fact that there are many ways of encoding information.

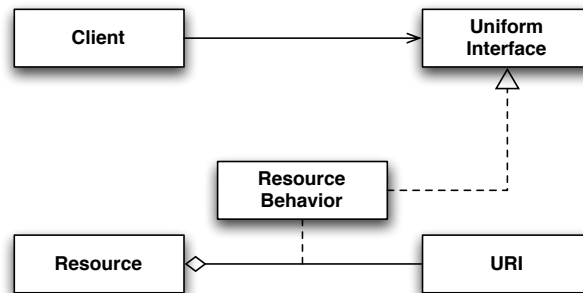


Figure 1: Exposing behavior in resource-oriented architectures

Conceptually, a resource is outside the architecture, a client can only communicate with it via the uniform interface it addresses by a globally unique identifier, e.g. a URI [18]. The relationship between a resource and a URI may change over time. Yet, resource orientation today spends very little effort on describing the *underlying process* defining the relationship between a resource and its URIs.

### 3 Example of a complex resource

As an illustration, let us now introduce a complex resource most people should be familiar with: an online ordering process. In figure 2, a very simple version is illustrated. A shopping cart is created by the user by adding an initial item. Adding items can be repeated as many times as the user likes. If the user simply stops interacting with the shopping cart, it may time out or the user decided to check out by choosing a payment method. At this point the user is presented with the content of the shopping cart, the chosen payment method, and the total bill to confirm before actually committing the order.

The first step towards a good resource-oriented design is to identify the relevant resources. Is a shopping cart actually a resource on its own, or just a state of an order? By modeling the later—the shopping cart to be just a state of an order resource—we can uniquely identify the order in all stages, e.g. shopping cart, check-out, assembly, in-delivery, and post-delivery. The user is given a single URI, something to bookmark in a browser. Clicking on such a bookmark will issue a GET request. A GET request in a resource-oriented view is nothing else than introspecting the current state of a resource. In the true spirit of *hypermedia as the engine of application state* the returned representation of the current resource should include all relevant links and currently possible interactions. By doing so, the client is never forced to understand the process as such, being able to browse and post is the only requirement to participate

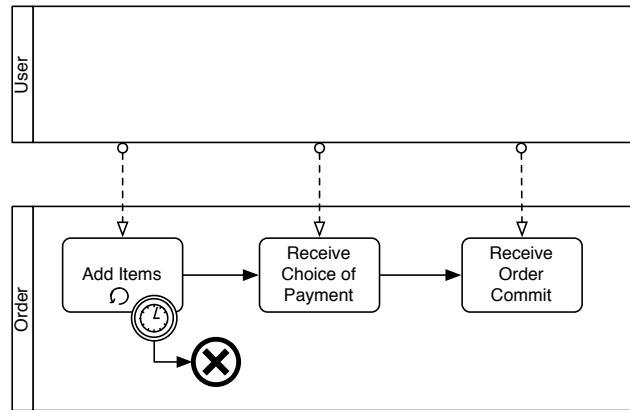


Figure 2: Shopping cart as a complex resource

as a client. This simplicity is the true strength of a resource-oriented design and the foundation of the World Wide Web's success story. At the same time, this motivates the resistance against formal descriptions of interfaces and processes as practiced in a Web Services environment. While resource orientation does not conflict with formal interface descriptions and in fact would benefit from it, any attempt to introduce such formalism to resource orientation must honor the fact that resource orientation can and will work without such formalism.

Nevertheless, it should have become apparent that resources are indeed complex state machine and that such state machines can be expressed as processes, matching the business concepts used to motivate the system in the first place. We already identified a shopping cart to be just a state or dependent sub-resource of an order. However, this opens the question of how to choose resource boundaries. Is the order a resource in itself or is it a sub-resource of the store? In [14] a very pragmatic answer is given: Breaking down an application into as many resources as possible benefits scalability and flexibility, but at the same time the resource is the scope of serializability, i.e. there may not be transactions across resource boundaries. I.e. the order is not dependent on the store (at least in a transactional view), but the order items probably are dependent on the order, as an order item may only be changed as long as the order has not been committed.

Before we evolve these concepts into a meta model in the next chapter, let us summarize our findings: A resource may consist of several complex states, each able to expose a set of URIs. Each of these URIs expose a certain behavior of the resource. Interaction with any of the resource's URIs is classified into *safe* (one interaction has the same effect as zero interactions), *idempotent* (one interaction has the same effect as  $n$  interactions), or *unrestricted*, i.e. such interaction is able to produce an uncontrollable side-effect and/or change the internal state of the resource. Also, a resource must be able to extract URIs from representations received via interaction and be able to then interact with the extracted URIs, as this is a fundamental aspect of resource orientation.

## 4 A Resource Meta Model

Resource orientation is an instance of a client-server architecture. However, neither client nor server are monolithic. In the last chapter, a typical example of a resource was given. Based on these findings, the following meta model can be deduced:

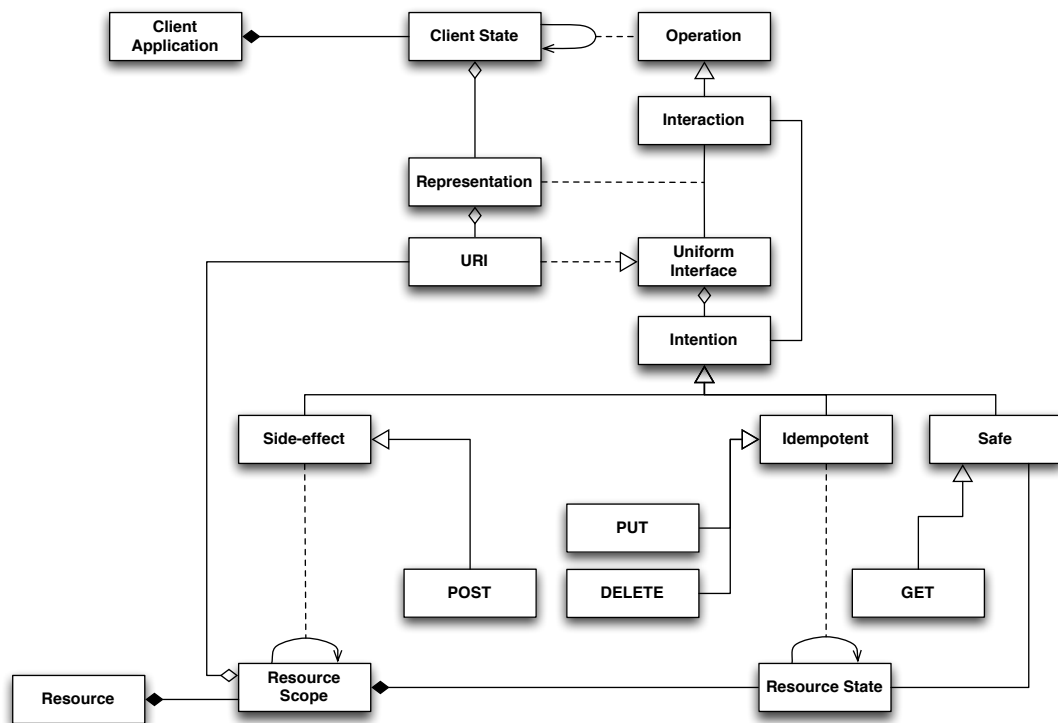


Figure 3: A Resource Meta Model

**Client Application** The best-known resource-oriented client application is a common browser. All applications can be broken down into distinct states, with operations moving the application from state to state. However, in a resource-oriented application, these states and the operations acting as transitions are not fixed. Instead they are dynamically altered by interaction.

**Interaction** Resource orientation has direct support for only one interaction style: Stateless request-response cycles. The client starts the interaction by sending a message, the server responds, and all necessary state at the protocol level is released. Hence, any interaction will conceptually look like any other, including the very first one.

**Unified Resource Identifier (URI)** As interactions are stateless, the target of a request-response cycle needs to be directly addressable. Various schemes do exist to encode such identifiers. To ease interaction, resource orientation introduces a uniform

mechanism as a superset to existing schemes. A resource-oriented client thus only needs to handle the uniform resource identifiers, the handling of the concrete scheme is moved to the protocol implementation.

**Representation** Client and server are conceptually completely separated, they can only exchange *representations*, i.e. documents describing the intended request and response. Once transmitted, a representation has no connection to its sender at all, it is self-contained and detached. Just as the URI decouples the identification, this approach decouples the client application from the transport layer. This communication style is sometimes called document-oriented.

**Client State** A client state is the sum of all then available representations. In the case of a browser, the initial representation available is the bookmark list and the default home page. Representations may contain URIs to resources. Any interaction with a URI will result in a new representation, thus a new client state. As representations are dynamically created upon request, this in fact turns the client application into a dynamically changing process, where all possible transitions are represented by URIs.

**Uniform Interface** All URIs expose the same interface, i.e. a client only needs to be able to interact with one, uniform interface. As described before, interaction with the uniform interface is always a synchronous request response cycle. Both request and response are representations. The encoding of the representation is negotiated upon each interaction. As the resource does not need to keep track of the resource, this simplifies support for different content types and increases the likelihood of successful interaction.

**Intention** Part of every request is the intention of the interaction. Resource orientation does not dictate which intentions are to be supported, just that the overall interface should be uniform, i.e. any intention should be meaningful to every possible resource. Having an explicit intention at the protocol level allows for seizing certain side conditions of an interaction without a semantic understanding of the content. The most relevant sub-classes of intention are *safe*, *idempotent*, and *side-effect*. A *safe* intention guarantees the requester to cause no side-effect in the interaction target. The seizable side condition of a *safe* intention is caching. If no side-effect is caused, the interaction does not actually have to be delivered to the receiver, as the response will be the same. *Idempotent* interactions guarantee  $N > 0$  identical requests to cause the same side-effect as a single request. The seizable side condition of an *idempotent* interaction is replayability. All other intention can be thought of as causing a *side-effect* each time when called. In figure 3 the relevant intentions of HTTP (GET, PUT, DELETE, and POST) are shown.

**Resource** Resource orientation is a client-server architecture and a resource is the abstraction used for the server side. Any information that can be named can be a



resource, both virtual (e.g. computer-based) and non-virtual (e.g. a person):

A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.

More precisely, a resource  $R$  is a temporally varying membership function  $M_R(t)$ , which for time  $t$  maps to a set of entities, or values, which are equivalent. The values in the set may be *resource representations* and/or *resource identifiers*. A resource can map to the empty set, which allows references to be made to a concept before any realization of that concept exists—a notion that was foreign to most hypertext systems prior to the Web [11]. Some resources are static in the sense that, when examined at any time after their creation, they always correspond to the same value set. Others have a high degree of variance in their value over time. The only thing that is required to be static for a resource is the semantics of the mapping, since the semantics is what distinguishes one resource from another. [8]

The meta model introduced here tries to capture the membership function  $M_R(t)$  and the side condition of the three intention classes in the concepts *resource scope* and *resource state*.

**Resource Scope** Given the above definition of a resource, there is a one to many relationship between resource and URI. This relationship is hidden to the outside—the URI completely decouples the communication system from the resource—and may change over time. A *resource scope* defines a fixed relationship between a resource and its bound URIs. Changing the set of URIs is only possible by a scope transition. In a truly resource-oriented environment, we can assume all side-effects to be addressable by an URI. Hence, POSTing to a URI can safely be assumed to produce a fresh URI and consequently to cause a scope transition within the resource.

**Resource State** Within a resource scope, idempotent interactions may transfer the resource into distinct *resource state*. The given resource state determines the response to *safe* interactions, but a resource state transition must not cause a change to the set of bound URIs of the resource, as this is limited to resource scope transitions.

## 4.1 Comparison to Remote Procedure Calls

While at first glance remote procedure calls (RPC) and resource orientation look similar, they are not. In RPC concepts are defined in terms of language APIs, not network-based applications. In resource orientation, the target of any interaction is a concept, the request uses a uniform interface with standard semantics. This allows any intermediary to process the request with the same effort as the final destination of the interaction. The result is an infrastructure that allows for layers of transformation and indirection that are independent of the information origin, tailored to the heterogeneous and multi-organizational reality for the Internet.

## 4.2 Comparison to Object-Oriented Programming

There are many programming languages today being called object-oriented. However, the original conception thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages:

OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things [15]

In this sense, true object orientation and resource orientation share a lot of features. However, just as with RPC, resource orientation puts a strong focus on the message itself and its intention, allowing intermediaries to transparently influence the interaction.

## 5 Process-oriented Resources

Given from the example and the deduced meta model, modeling resources as processes exposing URIs seems viable. In this section, BPEL is introduced and extensions for modeling complex resources are outlined.

### 5.1 BPEL

BPEL is arguably the de facto standard for specifying processes in a Web Services environment. BPEL provides *structured activities* that allow the description of the control flow between the *interaction activities*. BPEL does not support explicit data flow, but rather relies on shared variables referenced and accessed by interaction activities and manipulation activities. The control flow between activities can be structured either block-based by nesting structured activities like `< sequence >` and `< flow >`, or graph-based by defining directed edges (called `< links >`) between activities inside `< flow >` activities. Both styles can be used at the same time, making BPEL a hybrid language.

Beyond control flow and data manipulation, BPEL also supports the notion of scopes and allow for compensation handlers and fault handlers to be defined for specific scopes. Hence, scopes represent units of works with compensation-based recovery semantics. Fault handlers define how to proceed when faults occur, compensation handlers define how to compensate already completed activities, as processes not transactional and consequently must be rolled back explicitly. Further more, scopes allow for event handlers which can be regarded as repeatable, attached sub-processes [12] triggered by events.

### 5.2 BPEL without Web Services

The wide-spread acceptance and the sophistication of the control flow constructs, make BPEL a strong candidate when trying to formally express the process governing the relationship between a resource and its URIs. Both the interaction activities and the grouping mechanism that allows modeling complex message exchanges depend on WSDL. However, in [17] BPEL light is introduced, a WSDL-less version of BPEL.

While BPEL light itself still is not a good match for resource orientation, a clear path on how to remove the dependency on WSDL from BPEL and adding new interaction models in a compatible way is shown clearly. In essence, the elements *< receive >*, *< reply >*, *< invoke >*, *< onMessage >* within a *< pick >*, as well as *< onEvent >* within an *< eventHandler >* need to be replaced by constructs not relying on WSDL.

### 5.3 Using BPEL to model resource states

BPEL does not have an explicit state modeling, but an implicit via the *< scope >* construct. Generally speaking, a POST message or an event may cause a state transition. However, while in a state, as many GET, PUT, and/or DELETE messages may arrive, as they are safe and/or idempotent.

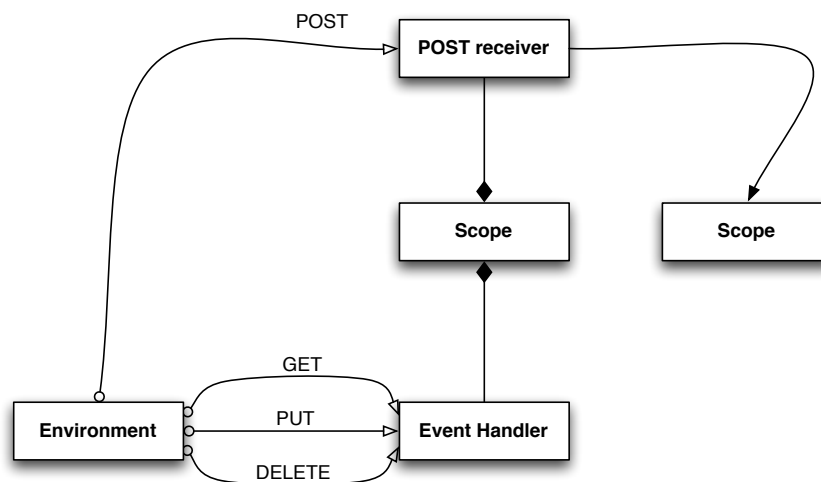


Figure 4: Using BPEL to model resource states

As shown in figure 4, BPEL provides the concept of event handlers to model GET, PUT, and DELETE interaction as attached, repeatable subprocesses. Enforcing *safe* and *idempotent* characteristics of those interactions is beyond the scope of this paper. However, a straight forward solution may be disallowing write access to any variable during a GET interaction to ensure safeness. PUT and DELETE can be enforced idempotent by disallowing write access to any variable read, i.e. overwriting a variable is ok, computing a new value based on the old one is not. Such interaction may be executed several times and in parallel, while POST interaction or events move the BPEL process into a new scope. Relating these concepts to the meta model from chapter 4, a BPEL process instance directly maps to a resource, a BPEL scope maps to a resource scope and the activities within each scope and all interaction in the event handlers map to resource state transitions.

## 5.4 Resource interaction in BPEL

Web Services try to abstract from the communication protocol, providing support for a wide range of interaction models, such as asynchronous or one-way interaction. Resource orientation on the other hand puts much effort into the core protocol as the lowest level of shared semantics. The dominate resource-oriented protocol is HTTP. Consequently there is no point in abstracting away from it when modeling interaction in BPEL. In fact many proponents of resource orientation have major concerns with any attempt to hide the protocol layer behind an abstraction.

All interaction in HTTP is based on synchronous request-response. Asynchronous communication is supported by identifying either the asynchronous sender or receiver by an explicit URI and sending it along in the initial request. I.e. at the protocol level, there will be a synchronous request and then an independent synchronous response push or pull. This design makes the interaction much simpler, but requires an easy mechanism to construct URIs. There is currently one attempt to standardized URI templating [10] applicable to creation, matching, and selection of URIs. Within WADL, URI templates are already used for matching and selection of URIs. To a resource itself, creation of URIs must be available, too. Coming back to our example process, upon receiving a shopping item, it must be added to the shopping cart, in turn generating one or more URIs for the newly created item.

```
<assign>
  <copy>
    <from>rbpel:generate-uri("./item/{itemNumber}")</from>
    <to variable="newItemURI" />
  </copy>
</assign>
```

Figure 5: URI creation by XPath-method

The easiest way to provide such functionality is to offer an XPath function. Figure 5 shows how the regular `< assign >` construct is used to create a new URI using such XPath function. URIs themselves do not need a special construct and can be kept in normal variables.

With URIs introduced to BPEL, let us look at URI interaction again, as shown in figure 6. Any URI interaction is synchronous and the tuple of *request* and *response* is grouped into a *message exchange*. Both request and response contain a header and a body, where the header includes the content type of the body. The response also includes a *status*, which is part of the uniform interface of HTTP and encodes a general indication of how the request was processed.

Remember, a message exchange is always synchronous. This reduced the possible interaction patterns to *send-receive* and *receive-reply*. While it is tempting to simply the BPEL constructs into a `< send >` and `< receive >` element with the complete handling of the request as child elements, the BPEL specification does not appear to allow extension activities with child elements, hence we refrain from doing so and stick with the traditional `< send >`, `< receive >`, and `< reply >` constructs without children. How-

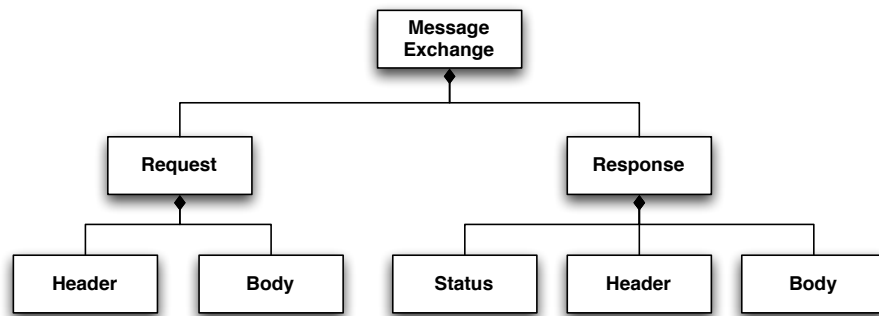


Figure 6: URI interaction

ever, the newly introduced activities all have a *messageExchange* attribute by which the required data structures—as shown in figure 6 are referenced.

```

<rbpel:onMessage path="./item/new" messageExchange="createItem">
  <wabl:method name="POST" />
  <sequence>
    <assign>
      <copy>
        <from>rbpel:generate-uri("./item/{itemNumber}")</from>
        <to variable="newItemURI" />
      </copy>
    </assign>
    <rbpel:reply messageExchange="createItem">
      <rbpel:status>201</rbpel:status>
      <rbpel:param name="Location" style="header">$newItemURI</rbpel:param>
    </rbpel:reply>
  </sequence>
</rbpel:onMessage>

```

Figure 7: Creating a shopping cart item

In figure 7 the fragment from figure 5 is completed to a complete *< onMessage >* block. Notice the *path* attribute containing a relative URI template. The given template is relative to the BPEL process, as each instance of the process is assigned a URI itself. The exact details of the message the *< onMessage >* activity is waiting for is described by reusing the *< method >* element of WADL [13]. Here, the only criteria is that the message is send as a POST. WADL itself is quite descriptive and this descriptive power can be used to model pattern matching on request, i.e. several *< onMessage >* activities waiting on the same URIs with the same verb but different contents. The *< reply >* activity again references the *messageExchange* data structure by attribute. Here, some convenience elements are shown (*< status >* and *< param >*), there functionality could be simply mapped to *< assign >* working on the data structure. However, this fragment shows how a new URI is generated by template and returned

to the requester in the *Location Header* as outlined in the HTTP specification.

The complete BPEL for all functionality hidden in the *Add Items* activity of figure 2 is shown in figure 8.

The loop—depicted by a curved arrow on the “Add Items” activity in figure 2—is mapped to a `< repeatUntil >` block. Upon receive a POST to the *checkout* URI the loop is left by setting the *commitRequest* variable to *true*. Also, the new internal state of the resource modeled by BPEL process outlined has a URI by itself. The requesting client is redirect to that URI by issuing a 303 status, again as outlined by the HTTP specification.

## 6 Related work

There are many other language available as a foundation to modeling resource behavior, such as Web Service Choreography Interface (WSCI) [1] or the Web Service Conversation Language (WSCL) [2]. The Bite language [6] follows a similar approach, but does not appear to focus on the intentional aspect of the uniform interface. Also, Bite is influenced but not based on BPEL. However, mind share is vital to language selection and BPEL seems to be able to form a common ground for various interest groups. Also, even though some constructs may be expressed more elegantly, BPEL is designed as an open, extensible language laying a clear track of how to integrate the required functionality, as shown in the course of this paper. Describing static resource interfaces, the author is unaware of any alternative to the Web Application Description Language. On the other hand, WADL can be seen as a mashup of HTTP, RelaxNG [5], and XML Schema [19], so these standards should be mentioned here as well.

## 7 Summary and Outlook

In the course of this paper, resource orientation was introduced as a viable subset of service orientation. Resource as such are complex state machines, exposing one or more uniform interfaces over time. This can be formally expressed as a complex state machine. The main contribution of this paper is to introduce a meta model for resource orientation clearly relating the intentions exposed via a uniform interface to the inner structure of resources. This inner structure may be described as a complex state machine. BPEL was identified as a suitable candidate for modeling such state machines and the necessary modifications to BPEL were outlined. All of these modifications are in the scope of the extension mechanisms of the BPEL specification.

However, at this point, the application of BPEL to describe resources is still regarded by the author as just an illustration of the core concepts of resource orientation. While an understanding of these concepts is stabilizing, research of how to best describe the inner structure of resources is just beginning. There are many web frameworks available, but they are lacking a process aspect. Also, there are many process languages available, but none seem to honor the intention aspect of the uniform interface. The

```

<repeatUntil>
  <scope xmlns:rbspel="http://bpt.hpi.uni-potsdam.de/ns/rbspel"
        xmlns:wadl="http://research.sun.com/wadl/2006/10">
    <eventHandlers>
      <rbspel:onMessage path="/item/{itemNumber}" messageExchange="itemShow">
        <wadl:method name="GET" />
        <!-- return representation of item $itemShow.itemNumber -->
      </rbspel:onMessage>
      <rbspel:onMessage path="/item/{itemNumber}" messageExchange="itemUpdate">
        <wadl:method name="PUT" />
        <!-- update and return item $itemUpdate.itemNumber -->
      </rbspel:onMessage>
      <rbspel:onMessage path="/item/{itemNumber}" messageExchange="itemDelete">
        <!-- delete item $itemDelete.itemNumber -->
      </rbspel:onMessage>
      <rbspel:onMessage path="/item/new" messageExchange="createItem">
        <wadl:method name="POST" />
        <sequence>
          <assign><copy>
            <from>rbspel:generate-uri("/item/{itemNumber}")</from>
            <to variable="newItemURI" />
          </copy></assign>
          <rbspel:reply messageExchange="createItem">
            <rbspel:status>201</rbspel:status>
            <rbspel:param name="Location" style="header">${newItemURI}</rbspel:param>
          </rbspel:reply>
        </sequence>
      </rbspel:onMessage>
    </eventHandlers>
    <pick>
      <rbspel:onMessage path="/checkout" messageExchange="transfer_to_payment">
        <wadl:method href="/wadl/post/method/definition" />
        <sequence>
          <assign>
            <copy><from>true</from><to>$commitRequest</to></copy>
          </assign>
          <rbspel:reply messageExchange="transfer_to_payment">
            <rbspel:status>303</rbspel:status>
            <rbspel:param name="Location" style="header">"/checkout"</rbspel:param>
          </rbspel:reply>
        </sequence>
      </rbspel:onMessage>
      <onAlarm><for>'2h'</for><exit/></onAlarm>
    </pick>
  </scope>
  <condition>$commitRequest</condition>
</repeatUntil>

```

Figure 8: Complete example of "Add Items" activity

author believes such a language to be an important contribution to web application development and will put his future focus towards shaping such a language.

## References

- [1] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek. Web service choreography interface (wsci). Technical report, W3C, 2002.
- [2] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web services conversation language (wscl). Technical report, W3C, 2002.
- [3] T. Berners-Lee. Www: Past, present, and future. *IEEE Computer*, 29(10):69–77, Oct. 1996.
- [4] S. Burbeck. The tao of e-business services, 2000. <http://www-128.ibm.com/developerworks/library/ws-tao/>.
- [5] Jim Clark and Murata Makoto. Relax ng specification. Technical report, OASIS Open, 2001.
- [6] Francisco Curbera, Matthew Duftler, Rania Khalaf, and Douglas Lovell. Bite: Workflow composition for the web. In *Service-Oriented Computing – ICSOC 2007*, 2007. <http://www.springerlink.com/content/0575313210780121/>.
- [7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. Technical report, The Internet Engineering Task Force, 1999. <http://www.ietf.org/rfc/rfc2616>.
- [8] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair-Richard N. Taylor, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [9] Dana Gardner. Soa wikis, soa for saas, and the future of business applications. Technical report, Interarbor Solutions, 2007. <http://blogs.zdnet.com/Gardner/?p=2395>.
- [10] J.C. Gregorio, M.H. Hadley, M.N. Nottingham, and D.O. Orchard. Uri template. Technical report, IETF, 2008. <http://bitworking.org/projects/URI-Templates/>.
- [11] K. Grønbaek and R. H. Trigg. Design issues for a dexter-based hypermedia system. *Communications of the ACM*, 37(2), p. 41-49, Feb. 1994.



- [12] Alexander Großkopf. *xbpmn. formal control flow specification of a bpmn-based process execution language*. Master's thesis, Hasso Plattner Institut and SAP Research Brisbane, 2007.
- [13] Marc Hadley. *Web application description language*, November 2006. <https://wadl.dev.java.net/>.
- [14] Pat Helland. *Life beyond distributed transactions: an apostate's opinion*. In *Third Biennial Conference on Innovative Data Systems Research*, 2007. <http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p15.pdf>.
- [15] Alan Kay. *On the meaning of "object-oriented programming"*, July 2003. [http://www.purl.org/stefan\\_ram/pub/doc\\_kay\\_oop\\_en](http://www.purl.org/stefan_ram/pub/doc_kay_oop_en).
- [16] C. Matthew, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz. *Reference Model for Service Oriented Architecture 1.0. Technical Report Committee Specification 1, OASIS Open*, 2006. [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-rm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm).
- [17] Joerg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann. *Bpel light*. In *5th International Conference on Business Process Management (BPM 2007)*. Springer, September 2007.
- [18] L.Masinter T.Berners-Lee, R.Fielding. *Uniform resource identifiers (uri): Generic syntax*. Technical report, The Internet Engineering Task Force, 1998. <http://www.ietf.org/rfc/rfc2396.txt>.
- [19] Henry S. Thompson, C. M. Sperberg-McQueen, Shudi Gao, Noah Mendelsohn, David Beech, and Murray Maloney. *Xml schema 1.1*. Technical report, W3C, 2006.



# FMC-QE - Case Studies

Stephan Kluth

stephan.kluth@hpi.uni-potsdam.de

This report summarizes two case studies, analyzed with the new modeling and evaluation calculus FMC-QE. Both case studies could be seen as a Proof-of-Concept for FMC-QE. The first case study is a comparison of the performance behavior of a real system, its simulation and the FMC-QE model of this system. With this SAP-based case study it could be seen, that the difference between the estimated and the measured performance values is small and so the different techniques (measurements, simulation and modeling) complete each other in the different development phases of a system. In the second case study, the Axis2 Web service framework is structured analyzed, operationally measured and modeled for quantitative evaluation. This case study shows, that FMC-QE is suitable to model and evaluate complex Service-oriented Architectures.

## 1 Introduction

In order to achieve a good scalability and the ability to preserve Service Level Agreements, performance modeling of Service-oriented Systems is important in all phases of the development. From this point of view, the performance modeling and analysis calculus FMC-QE (Fundamental Modeling Concepts for Quantitative Evaluation) is proposed. After an introduction into the basic concepts and definitions, as well as the presentation of the graphical notations of FMC-QE at the Spring 2007 Research School Workshop [5], this paper now presents two case studies, analytically modeled with FMC-QE.

The first case study<sup>1</sup>, summarized in section 3, compares the performance values of a real system, its simulation and the model of this system, as a Proof-of-Concept for FMC-QE and the simulation framework *Perfact*. This work was based on the results of the “*Perfact, too*” Bachelor’s Project, that the author co-supervised. In the paper [8], written together with Marcel Seelig, Flavius Copaciu, Tomasz Porzucek, Nico Naumann and Steffen Kuehn, the author was responsible for the modeling part of the case study on the basis of the Bachelor’s Project Report. Because of this fact, the performance modeling, not the simulation, is in the scope of this report.

<sup>1</sup>Parts of the work presented in section 3 will also be published in [8]: Marcel Seelig, Stephan Kluth, Flavius Copaciu, Tomasz Porzucek, Nico Naumann, and Steffen Kuehn. Comparison of performance modeling and simulation - a case study. Accepted at the 15th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2008), Belfast, Northern Ireland.

The second case study<sup>2</sup>, examined in cooperation with Flavius Copaciu, Tomasz Porzucek and Werner Zorn, summarized in section 4, presents the modeling of the Axis2 Web Service Framework. This work was another FMC-QE Proof-of-Concept. The main contribution of the author was the performance modeling and because of that, this part of the paper [1] is in the scope of this report.

The outline of this report is defined as followed: In order to support the reader in the understanding of the further sections, section 2 introduces FMC-QE, its diagram types and the FMC-QE Tableau. After that introduction, the case studies are summarized in section 3 and section 4. Finally conclusions and an outlook are given in section 5.

## 2 FMC-QE

### 2.1 Introduction

The Fundamental Modeling Concepts for Quantitative Evaluation (FMC-QE) [10, 11] is a modeling and evaluation calculus that originates from and extends the modeling technique Fundamental Modeling Concepts (FMC) [6, 9], by quantitative evaluation paradigms. In FMC-QE, the hierarchical service request is in the main focus of the modeling. Coming from the service request structures, the server structures and the dynamic behavior are also modeled. In order to adapt the different diagram types of the three dimensional description space, FMC-QE integrates ideas, results and advantages of the Queueing Theory and the Time Augmented Petri Nets.

Within the Queueing Theory mathematical approaches and rules to determine the performance measurements of systems are defined. This methodology mainly focuses on modeling of traffic flows within static structures, while the dynamic behavior is only implicitly modeled. The modeling of dynamic behavior using Queueing Theory is sub-optimal, because control flows are normally not considered. FMC-QE uses and extends the results of the Queueing Theory in order to define the static structures. Also the mathematical rules and laws of the Queueing Theory are used and adapted for the hierarchical calculations in FMC-QE.

State discrete Time Augmented Petri Nets, especially colored Time Augmented Petri Nets, are another foundation of FMC-QE. With this methodology, the performance analyst is able to describe complex problems in a detailed manner. The calculation of the performance measurements with Time Augmented Petri Net models is algorithmically complex and the modeling often abstracts from server structure beneath. In comparison to Time Augmented Petri Nets, the Petri Nets in FMC-QE are only one view within the three-dimensional FMC-QE view on the performance behavior of the systems. Here the systems are modeled within the scope of the hierarchical service request structures and so the dynamic behavior and the control flows, defined in the the Petri Nets complete the whole model together with the Entity Relationship Diagrams

<sup>2</sup>Parts of the work presented in section 4 have also been published in [1]: Flavius Copaciu, Stephan Kluth, Tomasz Porzucek, and Werner Zorn. Hierarchical modeling of the Axis2 web services framework with FMC-QE. In Third International Conference on COMmunication Systems softWARE and middlewaRE (COMSWARE 2008), Bangalore, India, January 2008.

and the Block Diagrams.

After modeling the performance aspects of the systems in the FMC-QE diagrams, different performance parameters are used as input for the FMC-QE Tableau. In this hierarchical balance sheet, based on Little's Law [7] and the Forced Traffic Flow Law [4], the performance values of the system are then derived on the assumption of stationary processes.

A main feature of FMC-QE is the three-dimensional hierarchical view on the performance behavior of a system. These three views are:

- The service request structures in Entity Relationship Diagrams,
- The dynamic behavior in Petri Nets and
- The static (server) structures in Block Diagrams.

While the main modeling focus of the Queueing Theory is on the static structures and the dynamic behavior is implicitly defined and in Time Augmented Petri Nets vice versa, in FMC-QE both aspects are modeled from the view of service requests. In opposite to Queueing Theory or Time Augmented Petri Nets, FMC-QE consists of not only one but three diagram types in order to separate different aspects of the system. The different diagrams are derived from FMC and described in section 2.2.

FMC-QE is designed to model the steady state of a system, from the view of the service request. As such it assumes, that the analyzed system fulfills all the steady state conditions.

## 2.2 Diagram Types

**FMC-QE Entity Relationship Diagrams** The modeling of the service request and the hierarchical structure of the service request is the main focus in FMC-QE. This structure and the mappings between the three diagrams are defined in Entity Relationship Diagrams. Beside this, the traffic flow coefficients and quantitative parameters of the service requests, are also defined in this diagram.

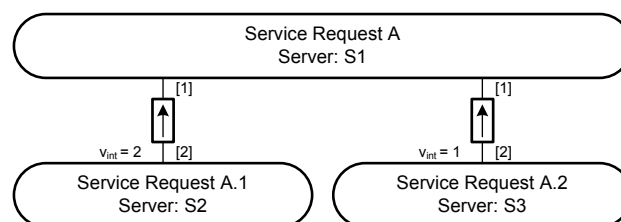


Figure 1: FMC-QE Entity-Relationship Diagram Example

Figure 1 shows the structure of a simple service request. In this example, a service request  $A$ , executed by server  $S1$  is hierarchically decomposed into two ( $v_{int} = 2$ ) sub-requests  $A.1$ , served by server  $S1$  and one subrequest  $A.2$ , computed by server  $S2$ . The service request  $A$  is positioned at the hierarchical level [1] (the topmost) and the service requests  $A.1$  and  $A.2$  are at the hierarchical level [2]. In order to provide the

mappings to the other diagrams, in the Petri Net, the action has the same index as in the Entity Relationship Diagram and the mappings to the static structures in the Block Diagram are done via the *Server* attribute.

**FMC-QE Block Diagrams** The static structures are described in Block Diagrams, which are a time extended version of the FMC Block Diagrams. In this diagram type, aspects and ideas of the Queueing Theory are modeled. In comparison to Queueing Theory, the views on the static structures in FMC-QE Block diagrams are an additional view on the system. Also the hierarchical structure of the systems is considered in the Block Diagrams and there is a strict distinction of control and service stations, as well as active and passive components.

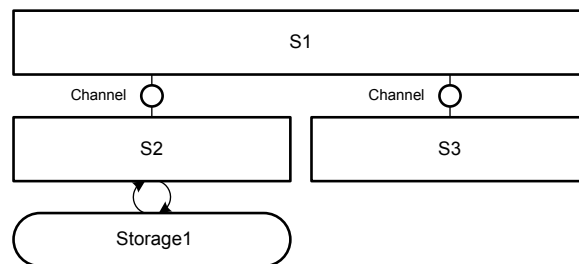


Figure 2: FMC-QE Block Diagram Example

Figure 2 describes a system of three active components (servers). Server  $S_1$  is connected to server  $S_2$  and  $S_3$  through a channel (volatile passive components). Additionally server  $S_2$  is connected with a storage (non volatile passive component) *Storage1*. With the help of this diagram, service request independent service times, like speed of a processor, could be defined.

**FMC-QE Petri Nets** The dynamic behavior is modeled in Petri Nets. These Petri Nets also have their origin in the FMC Petri Nets, but are extended by the aspects of performance evaluation. As a specialty, in the Petri Net, the gray places carry operational tokens (the service requests and responses) and the white places carry control tokens.

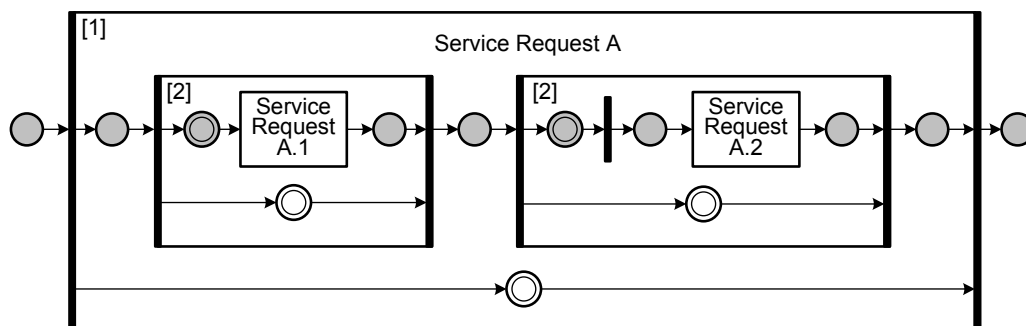


Figure 3: FMC-QE Petri Net Example

In figure 3 a service request  $A$  with two inner service requests  $A.1$  and  $A.2$  is shown.

In the above example, the parallelism of service request *A.1* is infinite and the parallelism of service request *A.2* is one with an infinite queue.

## 2.3 FMC-QE Tableau

Table 1: FMC-QE Tableau

Experimental Parameters:			
$N_{ges}$	20		
$\lambda_{bott} = \min(B_j)$	20,0000		
$1 \geq f = \lambda_{bott}/\lambda$	0,9000		
$\lambda$	18,0000		

Hierarchy Level [bb]	Service Request Section					Mapping		Dynamic Evaluation Section						
	Service Request $SRq_i$	$p_{[bb-1],i}$	$v_{i,ext}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	Server <sub>i</sub>	$X_i$	$\mu_i = m_j/X_i$	$\lambda_i$	$n_{i,q}$	$n_{i,s}$	$n_i = n_{i,q} + n_{i,s}$	$R_i$	$\rho_i$
2	Service Request A.2	1	1	3	3	S3	0,0500	20,0000	54,0000	8,1000	0,9000	9,0000	0,17	0,9000
2	Service Request A.1	1	1	1	1	S2	0,0300	33,3333	18,0000	0,0000	1,6200	1,6200	0,09	
1	Service Request A	1	1	1	1		0,0800		18,0000	8,1000	2,5200	10,6200	0,59	
1	Generate Request	1	1	1	1	Ext.	0,5211	18,0000	18,0000	0,0000	9,3800	9,3800	0,52	1,0000

Server Section					M/M/m bzw. M/M/∞						
Server	$X_j$	$m_j$	$\mu_j \cdot m_j$	$\mu_j$	$\lambda_j$	$\rho_j$	$\rho_{qj}$	$n_{j,q}$	$n_{j,s}$	$n_j$	$R_j$
S3	0,0500	1	20	20,0000	18,0000	0,9000	0,1000	8,1000	0,9000	9,0000	0,5000
S2	0,0300	∞		33,3333	54,0000				1,6200	1,6200	0,0300

The FMC-QE Tableau, shown in table 1, is a hierarchical balance sheet used for the performance evaluation of FMC-QE models. The Tableau consists of three linked tables. The main mathematical support is based on Little's Law [7] and the Forced Traffic Flow Law [4].

The upper table in table 1 defines experimental parameters, like overall number of service requests  $N_{ges}$ , bottleneck throughput  $\lambda_{bott}$  (derived in tableau), desired bottleneck utilization  $f$  and the overall arrival rate  $\lambda$ .

The table in the middle of table 1 defines and evaluates the quantitative parameters of the service request. In this table, for every (sub)request the following parameters or values are defined or derived:

- $[bb]$  – Hierarchy level - the hierarchical level of the request as shown in the FMC-QE diagrams;
- $SRq_i$  – Service request - the non-ambiguous name of the request;
- $p_{[bb-1],i}$  – Routing probability from the hierarchical higher request to this request;
- $v_{i,ext}^{[bb-1]}$  – Absolute value of the traffic flow coefficient on the next hierarchical level;
- $v_{i,int}^{[bb]}$  – Value of the traffic flow coefficient relative to the next hierarchical level;
- $v_i^{[bb]}$  – Absolute value of the traffic flow coefficient;
- $Server_i$  Corresponding server;
- $X_i$  – Measured service time for every request;

- $\mu_i$  – Maximal possible service rate;
- $\lambda_i$  – Arrival rate for this request - hierarchically derived -  $\lambda_i = \lambda * v_i^{[bb]}$ ;
- $n_{i,q} = n_{j,q} * (X_i/X_j)$  – Average number of queued requests;
- $n_{i,s} = n_{j,s} * (X_i/X_j)$  – Average number of requests in service;
- $n_i = n_{i,q} + n_{i,s}$  – Average number of requests;
- $R_i = n_i/\lambda_i$  – Average response time;

The lower table is used to describe the performance parameters of the static structures (the servers). The following parameters and values are used or derived in the table:

- Server – labels every server;
- $X_j = \sum_i(X_i)$  – Calculated service time ;
- $m_j$  – Multiplicity of the *server<sub>j</sub>*;
- $\mu_j * m_j$  – Service rate of *server<sub>j</sub>* ;
- $\mu_j = 1/X_j$  – Service rate for one server in *server<sub>j</sub>*;
- $\lambda_j$  – Arrival rate at *server<sub>j</sub>*;
- $\rho_j = \lambda_j/(\mu_j * m_j)$  – Utilization of *server<sub>j</sub>*;
- $p_{0,j}$  – Probability that no service request is in processing;
- $n_{j,q}$  – Average number of queued requests in *server<sub>j</sub>*;
- $n_{j,s}$  – Average number of requests in service in *server<sub>j</sub>*;
- $n_j$  – Average number of requests in *server<sub>j</sub>*;
- $R_j$  – Average response time.

The last five parameters are calculated according to the M/M/m and M/M/∞ formulas [3].

After setting up that Tableau, it is easy to generate “What if” scenarios, in order to predict the behavior of the system.



## 3 ERMF Case Study

### 3.1 Overview

The system modeled in paper [8] is a dynamic, Web service based, alert generation system called Emergency Risk Management Framework (ERMF). Originally the system was developed as case study of SAP Research France. Within a Bachelor's Project of the HPI, this case study was modeled, measured and simulated.

The sketch of the flow in the system is as follows: Due to dynamic defined rules, some Web services are called from the system. These Web services deliver environment data with which alerts could be generated.

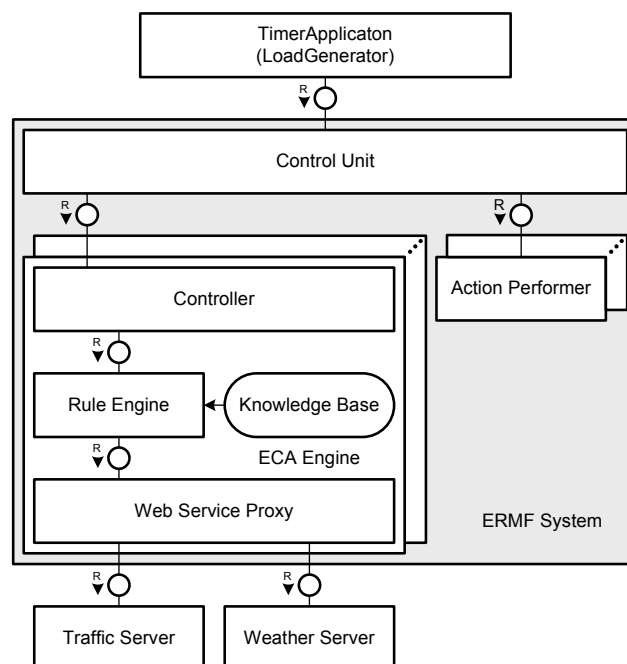


Figure 4: ERMF - Static Structure

Figure 4 shows the static structures of the system and its environment. The *Timer Application* generates the service requests for the system. The different service requests are then received by the *Control Unit* and dispatched for further processing. The main part of the system is the *ECA Engine* (Event-Condition-Action Engine). This component is responsible for applying the right rules in order to generate the risk estimation. This is done by choosing the most relevant Web services and evaluating their responses together with the rules. In this case study, the used public domain Web services are a weather forecast and a traffic information service. The responses of the *ECA Engine* are passed to the *Action Performer* through the *Control Unit*. Finally the alert is generated in the *Action Performer*.

The test platform was provided by the SAP AG and consists of the SAP NetWeaver Developer Studio 2004s as a development environment and the SAP Web Application Server 6.40 as server.

### 3.2 Service Request Structure and Dynamic Behavior

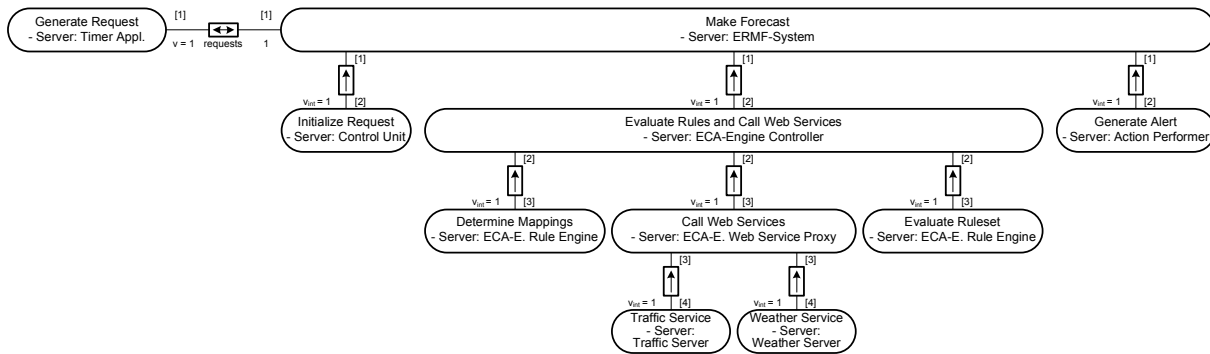


Figure 5: ERMF - Service Request Structure

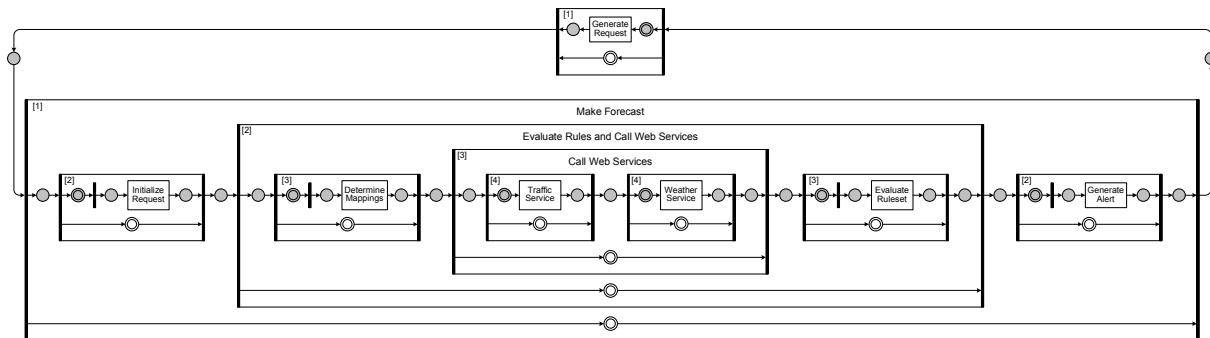


Figure 6: ERMF - Dynamic Behavior

Figure 5 and figure 6 illustrate the service structure and the dynamic behavior of the system. These two diagrams have the same hierarchies and complement each other. While the whole structure with traffic flow coefficients and the corresponding servers is shown in the Entity-Relationship Diagram, the dynamic flow, with parallelism and the right order is defined in the Petri Net.

As presented in figure 5 the *Make Forecast* request is the topmost request, which is then hierarchically decomposed into *Initialize Request*, *Evaluate Rules and Call Web Services* and *Generate Alert* requests. Accordingly the *Make Forecast* transition in the dynamic structure (figure 6) is decomposed into a sequence of these three transitions. Furthermore the *Evaluate Rules and Call Web Services* request was decomposed as shown in figure 5 and figure 6.

### 3.3 Measurements

The original ERMF source code was modified by including a measurement mechanism in order to provide measurements as a basis for the simulation and analysis, as well as for the comparison of the obtained results. The events handled by the ERMF framework are artificially generated by a so called *Timer Application*. These events are traced

throughout the system on a number of measurement points where the trace data is asynchronously written to a database for later evaluation. For measurements on the real system and during the simulation, the same measurement points were used. Thereby, any influence on the systems performance through the tracing is equal on both results.

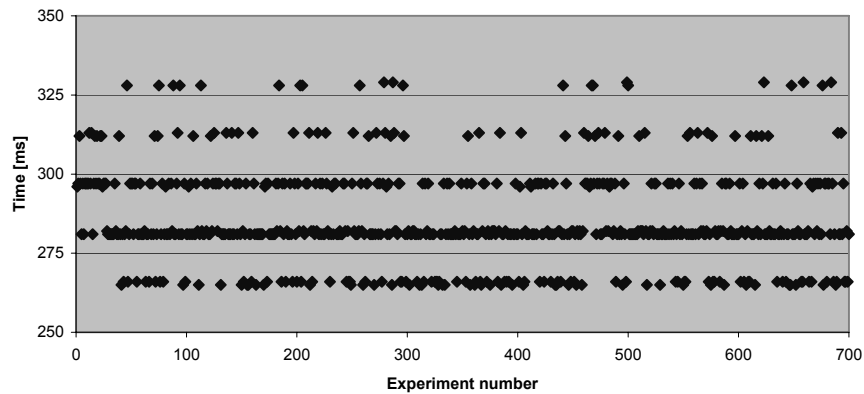


Figure 7: ERMF - Response Times - Traffic Service

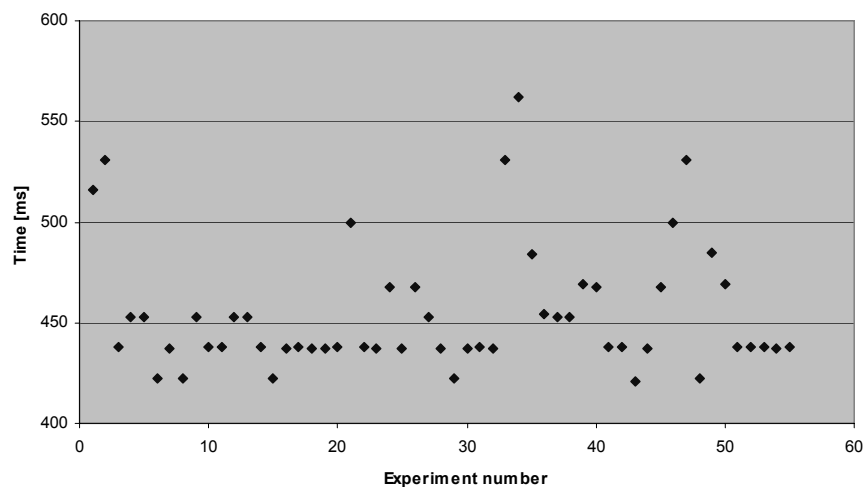


Figure 8: ERMF - Response Times - Weather Service

The charts in figures 7 and 8 show the response times of the traffic and weather services with the corresponding confidence intervals (95%) for average of  $285.8 \pm 1.09ms$  and  $454.91 \pm 8.21ms$ . The discrete values are a result of measurement resolution.

### 3.4 Analysis

The mean values obtained from the experimental data were used as input parameters for model evaluation.

Table 2: ERMF - Tableau

Experimental Parameters:			
$N_{ges}$	3	$\lambda$	1,0000

Service Request Section						Mapping		Dynamic Evaluation Section					
Hierarchy level [bb]	Service Request $SRq_i$	$p_{[bb-1],i}$	$v_{i,ext}^{[bb-1]}$	$v_{i,int}^{[bb]}$	$v_i^{[bb]}$	Server <sub>i</sub>	$X_i$	$\mu_i = m_j/X_i$	$\lambda_i$	$n_{i,q}$	$n_{i,s}$	$n_i = n_{i,q} + n_{i,s}$	$R_i$
2	Generate Alert	1,0	1,0	1,0	1,0	ERMF System (Action Performer)	0,010	100,000	1,000	0,001	0,010	0,011	0,011
3	Evaluate Ruleset	1,0	1,0	1,0	1,0	ERMF System (ECA Engine Rule Engine)	0,150	6,667	1,000	0,022	0,150	0,172	0,172
4	Weather Service	1,0	1,0	1,0	1,0	Weather Server	0,470		1,000	0,000	0,470	0,470	0,470
4	Traffic Service	1,0	1,0	1,0	1,0	Traffic Server	0,300		1,000	0,000	0,300	0,300	0,300
3	Call Web Services	1,0	1,0	1,0	1,0		0,770	1,299	1,000	0,000	0,770	0,770	0,770
3	Determine Mappings	1,0	1,0	1,0	1,0	ERMF System (ECA Engine Rule Engine)	0,200	5,000	1,000	0,029	0,200	0,229	0,229
2	Evaluate Rules and Call WS	1,0	1,0	1,0	1,0		1,120		1,000	0,050	1,120	1,170	1,170
2	Initialize Request	1,0	1,0	1,0	1,0	ERMF System (Control Unit)	0,350	2,857	1,000	0,050	0,350	0,400	0,400
1	Make Forecast	1,0	1,0	1,0	1,0		1,480		1,000	0,102	1,480	1,582	1,582
1	Generate Request	1,0	1,0	1,0	1,0	Timer Application	1,418	1,000	1,000	0,000	1,418	1,418	1,418

Server Section					M/M/m resp. M/M/∞						
Server	$X_i$	$m_j$	$\mu_j^* m_j$	$\mu_j$	$\lambda_j$	$\rho_j$	$p_{0,j}$	$n_{j,q}$	$n_{j,s}$	$n_j$	$R_j$
Traffic Server		0,300	∞	3,333	1,000			0,000	0,300	0,300	0,300
Weather Server		0,470	∞	2,128	1,000			0,000	0,470	0,470	0,470
ERMF System		0,710	2	2,817	1,000	0,355	0,476	0,102	0,710	0,812	0,812

Table 2, shows the FMC-QE Tableau of ERMF case study. In comparison to table 1, no bottleneck throughput  $\lambda_{bott}$  and no desired bottleneck utilization  $f$  is defined in this Tableau, because, in order to adjust the arrival rate to the arrival rates of the simulation and the real system, it is easier to define  $\lambda$  directly instead of defining it via  $f$  - with the risk to define illegal values ( $\lambda > \lambda_{bott}$ ).

### 3.5 Simulation

For the simulation, the Perfect framework, developed in the Bachelor's Project *Perfect* and *Perfect, too*, is used. More informations of this simulation framework can be found in [2] and [8]

### 3.6 Comparison of the Performance Values

Figure 9 shows a comparison between simulated results, the calculated results and measurements of the real system. The values are computed by changing experimental data in the FMC-QE Tableau and running experiments in the real system and the Perfect simulation. The calculated results are comparable as long as the system is operating under normal conditions ( $\rho < 1$ ). Running at its limits ( $\rho \geq 1$ ), the calculation predicts infinite response times due to the mathematical formulas of Queueing Theory, whereas the real system crashes because of buffer overflows. Similarly the simulation predicts invalid values. In this case, the calculated results of FMC-QE helped in understanding the system crashes due to overload and in finding an error in the implementation of Perfect.

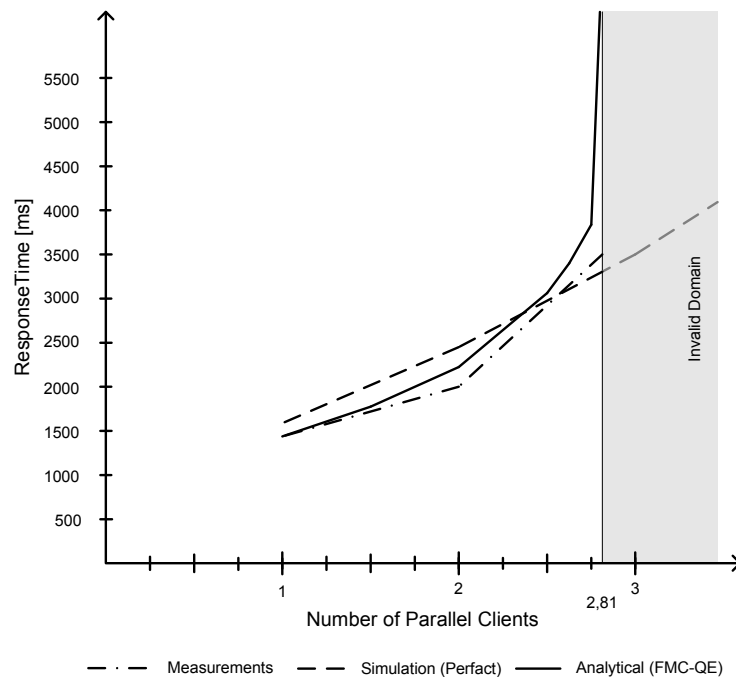


Figure 9: ERMF - Result Comparison

## 4 Axis2 Case Study

### 4.1 Axis2 Model Overview

As seen in section 2, a FMC-QE model consists of three diagrams: the Block Diagram, the Petri Net and the Entity Relationship Diagram. Using these diagrams types, a model of an Axis2 Web service Framework instance was developed and analyzed.

Axis2 can be used in two different ways, a standalone mode or deployed inside an application server. Due to the limitations of the standalone implementation (e.g. threading problems) it is usually deployed on application servers as a web application. This second case is shown in Fig. 10.

The diagram describes the components used, when new service requests are received. After the requests pass the *Admission Control*, they are queued in the *Request Queue*. Then the *Dispatcher* is responsible for allocating available threads from the *Thread Pool* for processing the service requests. The allocated thread performs all the processing required by the service request and is returned to the pool as soon as the processing has ended. The size of the thread pool is not static, it grows during the start phase, then it stabilizes between a minimum and a maximum thread count. New threads can be spawned to deal with usage peaks and idle threads are removed from the pool after a certain period of time. As soon as the processing of a service request has been completed, the service response is sent. The *Departure Control* shows the processing performed upon outgoing service responses.

The dynamic behavior of the system is presented in Fig. 11. The external world, representing the clients generating service requests, is modeled via the *Generate Re-*

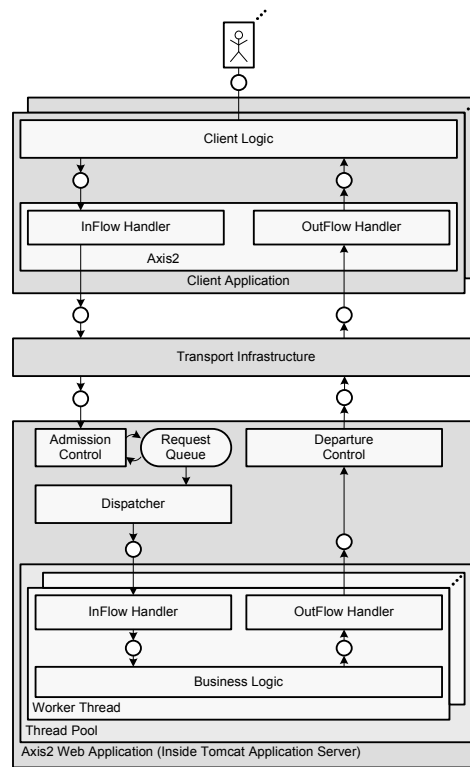


Figure 10: Axis2 Based System - Block Diagram

quests transition. This transition is at hierarchical level [1] just as the *Supervise and Execute Service Request* transition, used to model the whole Axis2 request processing. The processing threads are represented using the hierarchical level [2] transition *Execute Service*, while the processing flows and the business logic associated with the service are represented via the corresponding level [3] transitions. In section 4.2, the two processing flows are presented in detail. The service requests queue is represented by the infinite place between the level [1] and level [2] transitions. By default Axis2 uses an infinite queue, as shown in figure 11. If a specific queue size is set, the model has to be adapted by replacing the infinite place with finite, multi-token one.

The service request structure is presented in Fig. 12. This figure describes the hierarchical structure of the modeled Axis2 Service Request. This diagram describes among others the mapping between Petri Net transitions and agents in the Block Diagram, for example the *Execute Service* is handled by the *Worker Thread*. Besides this, the hierarchical levels and traffic flow coefficients are presented.

## 4.2 Axis2 Flows

In the Axis2 framework, the most important parts are the four SOAP processing flows that are part of the framework core: InFlow, OutFlow, InFaultFlow and OutFaultFlow. The first two are responsible for input and respectively output processing, while the last two are responsible for input and output processing in the case that errors have appeared. The error processing flows are quite similar to the regular ones and no

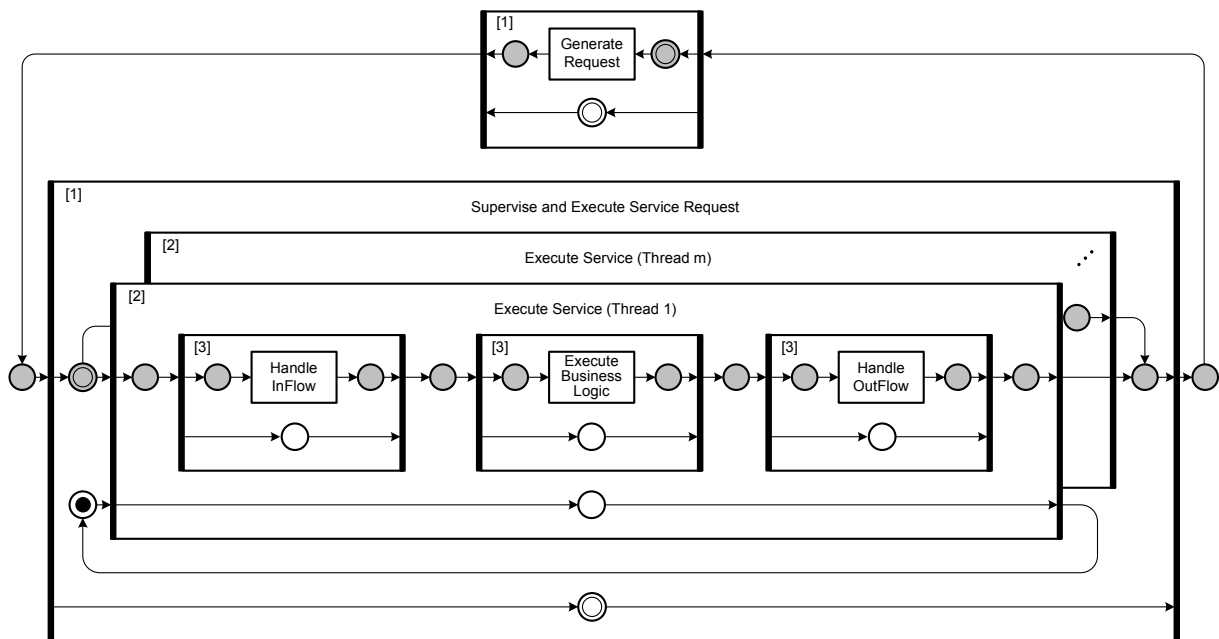


Figure 11: Axis2 Dynamic Behavior - Petri Net

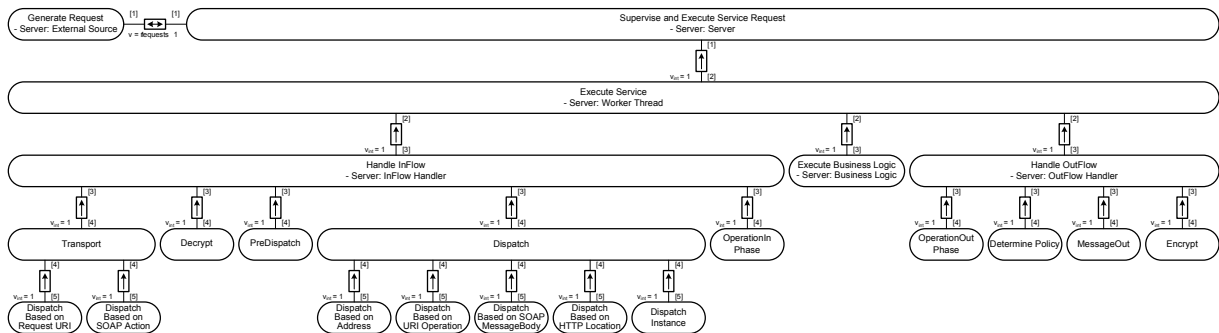


Figure 12: Axis2 Hierarchical Service Request - Entity Relationship Diagram

longer considered.

The Petri Net, detailing the input flow is presented in figure 13. This flow is the most complex one, spawning over three hierarchy levels, from level [3] to level [5].

The output flow, presented in figure 14, is simpler than the input flow and has only two hierarchical levels. This simplification is explained by the fact that the output flow has to take into account parameters that have been set during the input flow and by this the range of possible changes that can appear has been reduced. For example, if during the input flow the transport protocol has been decided as HTTP, the output flow will use this information and will not have to do any extra processing in order to determine a transport stack.

When comparing the graphical representation from Fig. 11 and 13 with the ones found in [5], [10] or [11], it can be noticed that the transitions have been simplified. This has been done by removing the place representing the queued service requests waiting to be served. This simplification is possible because there are no queues in the

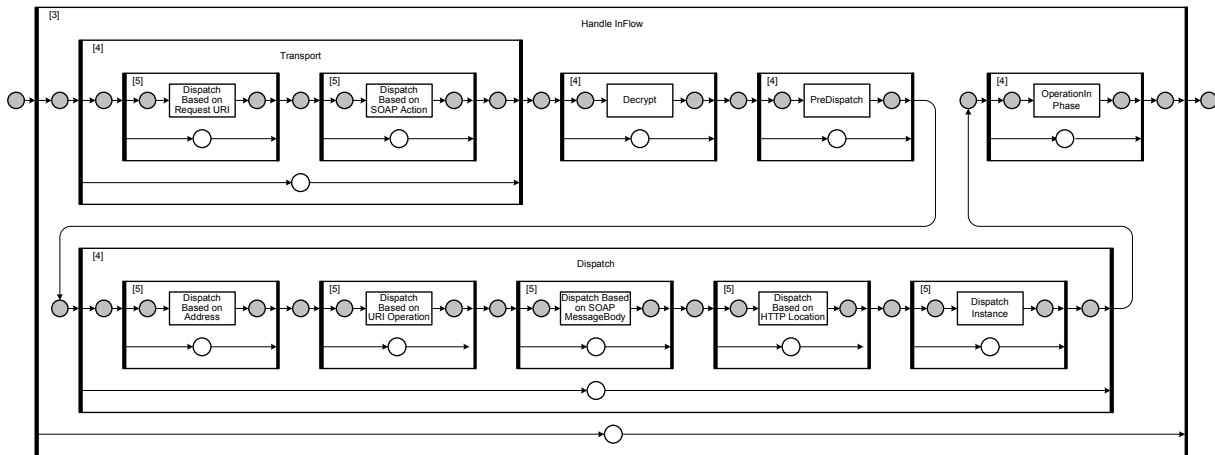


Figure 13: Axis2 Input Flow - Petri Net

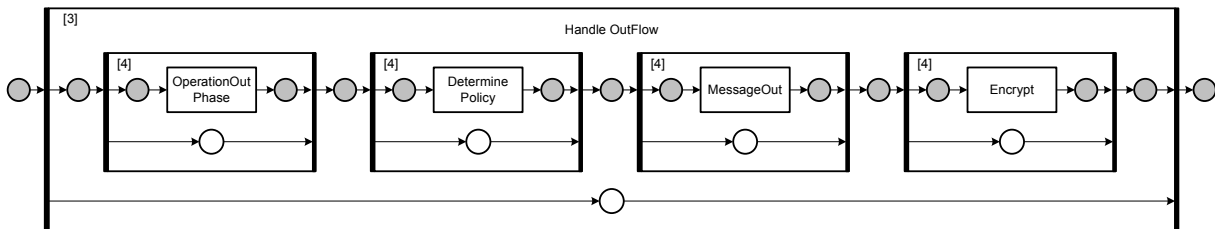


Figure 14: Axis2 Output Flow - Petri Net

processing flows. All the transitions belonging to the same thread will start executing as soon as the operation corresponding to the previous transition has finished. Inside Axis2 the only place where service requests are being queued is the queue in front of the *Thread Pool*. Another simplification was done by removing the traffic flow coefficients. This is possible since all the transformations are done one to one, that means that the coefficients are always 1 and have been removed from the graphs. However, the coefficients are used when building the FMC-QE Tableau, as shown in section 4.5.

### 4.3 Extending the Axis2 Model

One of the strengths of Axis is its flexibility and the possibilities to customize its behavior. The core of the system, presented in section 4.2 can be easily extended in order to incorporate new WS-\* extension, deployed as modules that can be enabled or disabled individually for each service or group of services.

When a new module is added to the handler chain and all services make use of that module, extending the model is a trivial task. Such changes can be easily reflected in the dynamic structure of the model by adding new handlers or phases in the existing handler chains, according to the specification of the newly activated module.

However, it is also possible to have specific modules activated and used only by some of the deployed Web services, as mentioned before. In order to cover such a case, a decision-making part has to be integrated in the model, as illustrated in Fig. 15,



where the *Encrypt* handler is not mandatory but optional. If encryption is activated, the service responses will be processed via the *Encrypt* handler. If encryption is not used, no processing will be performed, as indicated by the no-operation (NOP) transition.

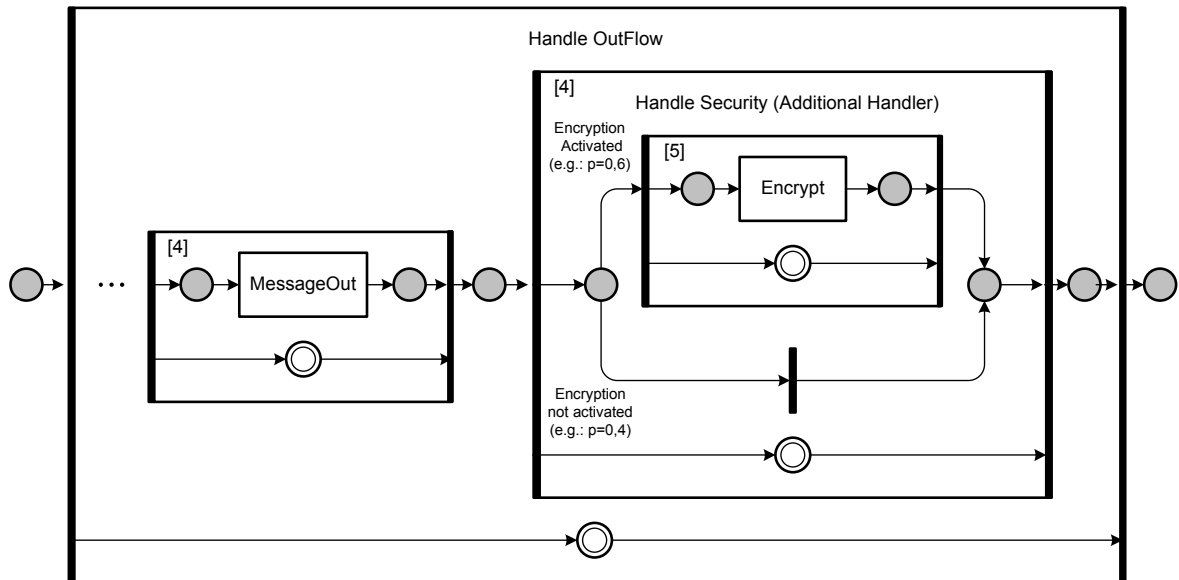


Figure 15: Axis2 - Optional handlers

For this situation it is necessary to determine the usage probability of the optional handler and to extend the model to incorporate this, just as in Fig. 15. This solution is appropriate if most of the services follow one branch and only a small percent follow the other. The drawback of this solution is the fact that, when there are many optional handlers, treating them through averages leads to a loss of representativity of the final results. This can be considered acceptable when such cases account only for a small amount of the total service requests.

The second solution to this problem implies transforming the problem into a multi-class one. Services that use the same path or closely related paths through the handler chain can be grouped together in service classes and the modeling can be done for these classes. Both of these approaches can be used with FMC-QE.

#### 4.4 Testbed Description

In order to perform the performance evaluation and prediction using FMC-QE, a benchmark of a server running Axis2 has been developed. The benchmarking has been done by using the Java method call `System.getNano()`. According to JAVA API documentation the method returns the current value of the most precise available system timer, in nanoseconds and provides nanosecond precision, but not necessarily nanosecond accuracy. On Mac OS X, this method delivers results with micro second precision. The Axis2 code has been extended with measuring points connected to each handler.

The server instance runs on a MacBook Pro machine with a Intel Core 2 Duo CPU at 2.16 GHz and 2 GB RAM. The machine runs Axis2 version 1.2 inside a Tomcat

6.0.10 application server on a Mac OS X 10.4.10 with Java 1.5.0.07. The experiments were repeated 1.000 times and the mean value corresponding to each handler has been calculated. For the experiments, the service *Version* available by default with each Axis2 distribution, has been invoked.

### 4.5 Tableau

Table 3: Axis2 - Tableau

Experimental Parameters:	
$N_{ges}$	15
$\lambda_{botf} = \min(B_j)$	0,0053
$f \geq f = \lambda_{botf}/\lambda$	0,9000
$\lambda$	0,0047
CPU	1

Service Request Section						Mapping			Dynamic Evaluation Section					
Hierarchy level [bb]	Service Request SRq <sub>i</sub>	$P_{[bb-1]j}$	$V_{i,ext}^{[bb-1]}$	$V_{i,int}^{[bb]}$	$v_i^{[bb]}$	Server <sub>i</sub>	$X_i$	$\mu_i = m_i/X_i$	$\lambda_i$	$n_{i,q}$	$n_{i,s}$	$n_i = n_{i,q} + n_{i,s}$	$R_i$	$\rho_i$
4	Encrypt	1	1	1	1	Worker Thread	0,58	8,6207	0,0047	0,0000	0,0027	0,0027	0,58	0,0005
4	Message Out	1	1	1	1	Worker Thread	4,45	1,1236	0,0047	0,0000	0,0211	0,0211	4,45	0,0042
4	Determine Policy	1	1	1	1	Worker Thread	0,4	12,5000	0,0047	0,0000	0,0019	0,0019	0,40	0,0004
4	OperationOut Phase	1	1	1	1	Worker Thread	0,43	11,6279	0,0047	0,0000	0,0020	0,0020	0,43	0,0004
3	Handle OutFlow	1	1	1	1		5,8600		0,0047	0,0000	0,0278	0,0278	5,86	
3	Execute Business Logic	1	1	1	1	Worker Thread	32,45	0,1541	0,0047	0,0000	0,1537	0,1537	32,45	0,0307
4	OperationIn Phase	1	1	1	1	Worker Thread	0,62	8,0645	0,0047	0,0000	0,0029	0,0029	0,62	0,0006
5	Dispatch Instance	1	1	1	1	Worker Thread	32,93	0,1518	0,0047	0,0000	0,1560	0,1560	32,93	0,0312
5	Dispatch Based on HTTP Location	1	1	1	1	Worker Thread	0,27	18,5185	0,0047	0,0000	0,0013	0,0013	0,27	0,0003
5	Dispatch Based on SOAP Msg Body	1	1	1	1	Worker Thread	0,1	50,0000	0,0047	0,0000	0,0005	0,0005	0,10	0,0001
5	Dispatch Based on URI Operation	1	1	1	1	Worker Thread	0,1	50,0000	0,0047	0,0000	0,0027	0,0027	0,58	0,0001
5	Dispatch Based on Address	1	1	1	1	Worker Thread	28,71	0,1742	0,0047	0,0000	0,1360	0,1360	28,71	0,0272
4	Dispatch	1	1	1	1		62,1100		0,0047	0,0000	0,2965	0,2965	62,59	
4	PreDispatch	1	1	1	1	Worker Thread	5	1,0000	0,0047	0,0000	0,0237	0,0237	5,00	0,0047
4	Decrypt	1	1	1	1	Worker Thread	0,49	10,2041	0,0047	0,0000	0,0023	0,0023	0,49	0,0005
5	Dispatch Based on SOAP Action	1	1	1	1	Worker Thread	39,73	0,1258	0,0047	0,0000	0,1882	0,1882	39,73	0,0376
5	Dispatch Based on Request URI	1	1	1	1	Worker Thread	43,74	0,1143	0,0047	0,0000	0,2072	0,2072	43,74	0,0414
4	Transport	1	1	1	1		83,4700		0,0047	0,0000	0,3954	0,3954	83,47	
3	Handle InFlow	1	1	1	1		151,6900		0,0047	0,0000	0,7208	0,7208	152,17	
2	Execute Service	1	1	1	1		190,0000		0,0047	6,8624	0,9023	7,7647	1639,22	
1	Supervise and Execute	1	1	1	1		190,0000		0,0047	0,0000	7,7647	7,7647	1639,22	
1	Generate Request	1	1	1	1		1527,4495	0,0047	0,0047	0,0000	7,2353	7,2353	1527,45	1,0000

Server Section						M/M/m					
Server Name	$X_i$	$m_i$	$\mu_i \cdot m_i$	$\mu_i$	$\lambda_i$	$\rho_i$	$\rho_{0j}$	$n_{i,q}$	$n_{j,s}$	$n_i$	$R_i$
Worker Thread	950,0000	5	0,005263158	0,0011	0,0047	0,9000	0,0050	6,8624	4,5000	11,3624	2398,7371

The FMC-QE Tableau, shown in table 3, calculates the performance values of the Axis2 model. A description of the calculations and different parts of the Tableau can be found in section 2.3.

With the the help of the Tableau, performance predictions are easy to derive.

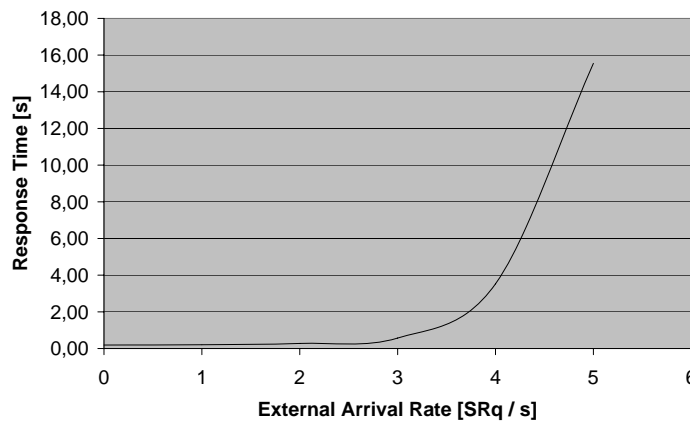


Figure 16: Axis2 - Response Times from FMC-QE Evaluation

Figure 16 shows the prediction of the response times of the whole system, evaluated with FMC-QE.

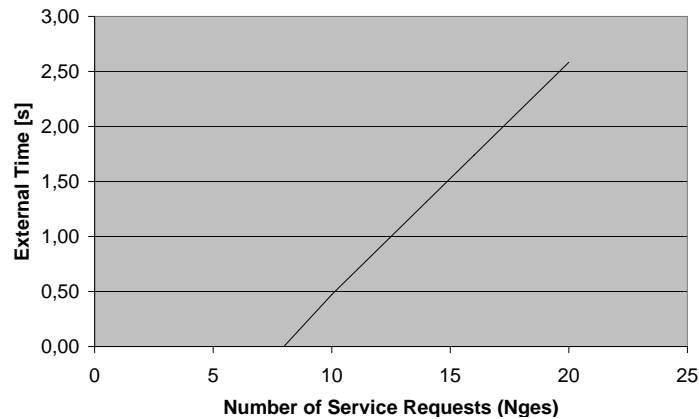


Figure 17: Axis2 - Increasing the Number of Service Requests

Figure 17 shows the dependence of  $N_{ges}$  (the total number of service requests) and  $X_{ext}$  (the external service time) as a result of the Response Time Law  $X_{ext} = (N_{ges}/A_i) - R_{sys}$ .

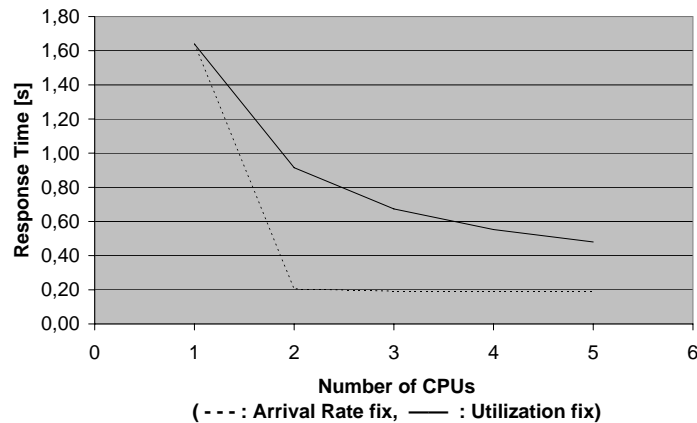


Figure 18: Axis2 - Increasing the Number of CPUs

In figure 18 the number of available CPUs is increased. Here, in the dashed line, the arrival rate is fixed to the value of table 3 ( $\lambda = 0,0047[ServiceRequests/ms]$ ) and in the solid line, the desired utilization of each server is fixed to the value in table table 3 ( $f = 0,9$ ).

During the evaluation differences have been noticed between the hierarchically estimated service times and some measurements done at the phase level. Our hypothesis is, that differences are due to Java object instantiation time, partially not taken into account in our benchmark. Work is currently under going to in order to minimize this differences.

## 4.6 Summary

In this case study a large and complex system has been modeled using FMC-QE and the methodology has proved suitable for modeling such systems and depicting their hierarchical structure. The performance modeling of Axis2 has been done using measurements from a test system. The results have confirmed the methodology and the usability of FMC-QE for performance estimations.

After setting up the FMC-QE model and Tableau of Axis2, performance predictions could be performed in a fast and simple way. A set of parameters, e.g probabilities, service time or number of CPUs, can be changed in the Tableau and allow to investigate the behavior of the system under a broad range of possible configurations.

## 5 Conclusions and Outlook

The two case studies show, that FMC-QE is suitable to describe complex service oriented systems in its quantitative behavior. Also the FMC-QE Tableau with its hierarchies and easy computations is qualified for this type of systems. With this modeling and evaluation technique developers and performance analysts are able to generate “What if” scenarios for a better understanding of the the performance behavior of the systems and developing of the architecture throughout the life cycle.

There is currently ongoing work focused on extending FMC-QE in order to increase the range of problems that it can addressed. This includes improvement of the diagrams, developing systems transformations and enhancing the mathematical support. It is of interest to see how other aspects of Service-oriented Computing, such as service composition, can be modeled using FMC-QE and how performance estimations can be done in such a context.

## 6 Acknowledgment

The author would like to thank Prof. Werner Zorn, Tomasz Porzucek and Flavius Copaciu for valuable discussions and cooperative work in FMC-QE. Especially for the paper [8] the author wants to thank Marcel Seelig, Nico Naumann and Steffen Kuehn and the other “*Perfact, too*” Bachelor’s Project members for the collaboration, as well as SAP Research France, especially Cedric Ulmer, Volker Gersabeck and Martin Grund for providing the case study, their support and valuable feedback during the development phases of the Bachelor’s Project.

## References

- [1] Flavius Copaciu, Stephan Kluth, Tomasz Porzucek, and Werner Zorn. Hierarchical modeling of the Axis2 web services framework with FMC-QE. In *Third International Conference on COMMunication Systems softWARE and middlewaRE (COMSWARE 2008)*, Bangalore, India, January 2008.

- 
- [2] Gero Decker, Volker Gersabeck, Jan Schaffner, and Marcel Seelig. Architecture-based performance simulation. Hong Kong, China, March 2007. IMECS.
- [3] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc., New York, 3rd edition, 1998.
- [4] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.
- [5] Stephan Kluth. FMC-QE - Positioning, Basic Definitions and Graphical Representation. Presented at the Spring 2007 Workshop of the HPI Research School on Service-Oriented Systems Engineering, Hasso Plattner Institute for Software Systems Engineering, Potsdam, Germany, April 2007.
- [6] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. John Wiley & Sons, March 2006.
- [7] John D. C. Little. A Proof of the Queueing Formula  $L = \lambda * W$ . *Operations Research*, 9:383–387, 1961.
- [8] Marcel Seelig, Stephan Kluth, Flavius Copaciu, Tomasz Porzucek, Nico Naumann, and Steffen Kühn. Comparison of performance modeling and simulation - a case study. Accepted at the 15th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2008), Belfast, Northern Ireland.
- [9] Peter Tabeling. *Softwaresysteme und ihre Modellierung*. Springer, Berlin, Heidelberg, 2006.
- [10] Werner Zorn. FMC-QE - A New Approach in Quantitative Modeling. In Hamid R. Arabnia, editor, *International Conference on Modeling, Simulation and Visualization Methods (MSV 2007) within WorldComp '07*, pages 280 – 287, Las Vegas, USA, June 2007. CSREA Press.
- [11] Werner Zorn. Hierarchische Modellierung basierend auf Bedienanforderungen. In Paul Müller, editor, *21.DFN- Arbeitstagung über Kommunikationsnetze*, Kaiserslautern, Germany, May 2007. University of Kaiserslautern.



# A Matter of Trust

Rehab Al Nemr

rehab.alnemr@hpi.uni-potsdam.de

This report describes the research direction that I have been investigating in the past five months. Starting from a security point of view, I explored some questions in Service Oriented Architectures. Trust management appeared to be one of the major ongoing research areas as the challenges are increased by the evolving nature of open systems. In an environment where the system participants do not know each other, a protocol should be defined to start the interaction between them. In order to do that they have to start building some kind of mutual trust level. Building, maintaining, dealing and raising this trust brings up lots of questions in the information system and security communities.

## Keyword List

Trust, Trust negotiation, Security, Policy, Semantic Web.

## 1 Introduction

Due to the nature of information systems and the rapid evolution into pervasive and ubiquitous systems, more challenges are occurring everyday in many directions, especially concerning the security. The challenges arise from the fact that systems participants are not always pre-identified and they are changed regularly. Even security policies are subjected to change when the system environment change or new regulations are applied. Researchers in many areas are working seamlessly to enable people, agents, services, and devices interact as autonomously as possible while preserving appropriate security and privacy policies.

Recently the word *Trust* became a buzzword not only in the security community but also in all the information system and intelligent systems communities. In social science the word *Trust* can be defined as: *a relationship of reliance. A trusted party is presumed to seek to fulfill policies, ethical codes, law and their previous promises. Trust is a prediction of reliance on an action, based on what a party knows about the other party.* [1] But what about the definition in Computer Science? In [2] the author concluded that Trust in the computer literature means: reputation, security concerns, quality of data or services, credentials, risk management, and many more. More or less they are all playing a role when dealing with *Trust*.

In a Service Oriented Architecture (SOA) the need to provide different levels of security between services that handle private information is becoming more critical. These services will need to provide privacy guarantees to prevent delicate information from ending up in the wrong hands. This is not an easy task in a world that has always new participants, laws and policies, requirements, and conditions. In open systems, authentication-based security and privacy schemes are inadequate, due to the fact that principals might be able to provide authentication but are otherwise unknown to the system and thus not authorizable for specific actions. In this case services will adopt the real-world behavior: *rely on Trust-relationships*.

Trust-based systems use *Trust* social definition, *a prediction of reliance based on what a party knows about the other party*, to create a framework in which two unrelated parties may establish the trust sufficient to perform sensitive transactions. So in order to establish this trust, organizations or systems, and by consequence services, state policies describing who can do what under what circumstances. This kind of reasoning is needed not only in Web Services but critically in Semantic Web services that exploit the Semantic Web to automate their discovery and interaction. This is because they must autonomously decide what information to exchange and how. The process that includes making assessments and decisions regarding trust relationships is called *trust management*.

There has been extensive research in the area, including the Semantic Web community, but there exist yet some issues that prevent policy frameworks and trust management systems from its adoption by users and real world applications [3].

In the remainder of this report I describe in Section 2 the trust management process and its importance in service oriented architecture, focusing on policy-based approach with a brief description of the reputation-based model. Section 3 discusses different types of policies, trust negotiation and the difference between strong and lightweight evidence. Section 4 then highlights the importance of user awareness in order to give him control over his policies – personalized policies. Section 5 states the main challenges and open research issues I have extracted from the references. I finish with future work in section 6.

## 2 Trust Management

The term *Trust Management* is defined as: “*a unified approach to specifying and interpreting security policies, credentials, and relationships which allow direct authorization of security-critical actions*”, and has been given later a broader definition: “*Trust management is the activity of collecting, encoding, analyzing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships*”. [4]

Currently there are two different major approaches for managing trust: *policy-based* and *reputation-based*. The two approaches have been developed within the context of different environments and targeting different requirements. On the one hand, policy-based trust relies on objective “strong security” mechanisms such as



signed certificates and trusted certification authorities in order to regulate the access of users to services. Moreover, the access decision is usually based on mechanisms with well defined semantics (e.g., logic programming) providing strong verification and analysis support. The result of such a policy-based trust management approach usually consists of a binary decision according to which the requester is trusted or not, and thus access to the service (or resource) is allowed or denied. On the other hand, reputation-based trust relies on a “soft computational” approach to the problem of trust. In this case, trust is typically computed from local experiences together with the feedback given by other entities in the network (e.g., users who have used services of that provider).

The two trust management approaches address the same problem - establishing trust among interacting parties in distributed and decentralized systems. However, they assume different settings. While the policy based approach has been developed within the context of structured organizational environments (which is why the research of policy-based approach within unstructured open environment is taking place), the reputation systems have been proposed to address the unstructured user community. Consequently, they assume different sources for trust (Certificate Authorities and community opinion, respectively) and accordingly employ different mechanisms. [4]

The authors in [4] analyze the two approaches and describe how Trust Management Systems can be improved by integrating both approaches. Their proposal is to combine both rule-based and credential-based trust with numerical trust estimates based on a large number of sources. PROTUNE is an example of a framework which makes use of both approaches.

Another approach – very common in today’s applications – is based on forcing users to commit to contracts or copyrights by having users click an “accept” button on a pop-up window. This is perhaps the lightest approach to trust, that can be generalized by having users utter *declarations* (on their e-mail address, on their preferences, etc.) e.g. by filling an HTML form. [5] This approach has the drawback of the “take it or leave it” behavior. [6]

In this report I am focusing on the policy-based approach and the problems, and challenges, associated with the use of this approach in open systems. My next step will be to investigate the reputation-based approach, the integration between the two approaches and the benefits from a security perspective, and the use of the propagation-of-trust models in both Semantic Web and Web 2.0.

## 3 Policy-based approach

Policies, which usually govern the behavior of networking services (e.g., security, QoS, mobility, etc.), are becoming an increasingly popular approach for the dynamic regulation of web information systems. The adoption of a policy-based approach for controlling a system requires an appropriate policy representation regarding both syntax and semantics, and the design and development of a policy management framework. In the context of the Web, the use of languages enriched with semantics

(i.e. semantic languages) has been limited primarily to represent web content and services. However the capabilities of these languages, coupled with the availability of tools to manipulate them, make them well suited for many other kinds of applications, as policy representation and management. [7]

In this section, different kinds of policy definitions', requirements for an efficient policy language and framework, trust negotiation, and the use of policies in open systems are being discussed.

### 3.1 Policies

The term policy encompasses different notions like, security policies, trust management policies, business rules and quality of service specifications. But they all make decisions based on similar pieces of information - evidence. Web policies play crucial roles in enhancing security, privacy, and also service usability. In general, policies are used to control how decisions and actions are taken. [3]

Examples of business rules and quality of services policies:

- Give customers *younger than 26* a 20% discount on international tickets
- Up to 15% of network bandwidth can be reserved by paying with an accepted *credit card*

An example for a Pervasive Computing policy:

- My colleagues can only see the building I am in and only when they are on company premises

Moreover, policies can specify event logging (e.g. failed transactions must be logged) communication and notifications (e.g. Notify the admin about repeated login failures), workflow triggering and actions that interleave with the decision process. [3]

In Web services environment, a single policy may be associated with a service or multiple services. Policies may also change over the lifetime of a service. [8]

#### 3.1.1 Security Polices

Security Policies can be defined as policies that:

- Describe what the entities can/cannot do in a certain context
- Describe what the entities must/must not do in a certain context
- Define permissions, obligations, norms and preferences for an agent's actions and interactions with other agents and programs. This can form the basics for e-contracts and negotiating agreements.
- Are explicit representations of constraints and rules that govern an agents or system's behavior.
- Are a set of rules and practices describing how an organization manages, protects and distributes sensitive information at several levels. They can be defined to perform a wide variety of actions, from IPsec/IKE management (example of network security policy) to access control over a web server (example of application-level policy). [7]

Conditional policies are policies that let an entity perform a certain action or set of actions under the condition that it will assume certain additional responsibilities.

Security policies can be categorized into two broad categories: *privacy* and *authorization* policies. *Privacy policies* specify under what conditions you can exchange information and the legitimate uses of that information. For example, a privacy policy might say that a provider could give a requester a key to access private information only if the key is encrypted during transmission. When a requester discovers the policy, it should decide whether it can satisfy this condition. The requester might have its own privacy policy that requires keeping certain information confidential, so it likewise can't share unencrypted private information. The requestor's privacy policy prevents it from interacting with web services that don't perform the needed encryption. Privacy policies help specify data confidentiality during transmission as well as after receipt. Consider a service that says it won't distribute details it receives as input. A requester that values privacy might see this as an important requirement. You can interpret a Web Service's privacy policies as an obligation and contract. [9]

*Authorization policies* constrain the provider to accept requests for service only from certain clients. For example, a service's authorization policy could state that a requester must act on behalf of a person who belongs to a certain organizational group and can prove membership with a digital certificate. Similarly, the requester could limit invocation to selected providers. [9] In general, authorization policies negotiate for access, control information exchange and monitor for suspicious events to be reported. [10] However, policies are symmetric; they may constrain both client and service.

### 3.1.2 Delegation of Trust

As mentioned before, conditional policies are policies that let an entity perform a certain action or set of actions under the condition that it will assume certain additional responsibilities. This includes the *delegation of trust*. Authors in [11] describe the delegation chain as: only users with the right to delegate a certain action can actually delegate that action, and the ability to delegate itself can be delegated. Users can constrain delegations by specifying whether delegated users can re-delegate the right and to whom they can do it. Once users are given certain rights, they are responsible for the actions of the users to whom they subsequently delegate those rights and privileges. This forms a delegation chain in which users only delegate to other users whom they trust. If any user along this delegation chain fails to meet the requirements associated with a delegated right, the chain is broken. Following the failure, no user can perform the action associated with the right.

Rein [12][13] is an example of a security framework, which supports the delegation of trust and authorization. Rein, a framework that is grounded in Semantic Web technologies, allows policies to be less exhaustive and provides decentralized security control. Delegation of authorization is very important to the Web because owners of web resources may not be able to project who should have access to their resources or pre-establish all desirable requirements for access. This kind of delegation allows permissions on a resource to be propagated by a set of trusted entities without explicitly changing the policy or requirements.

### 3.1.3 Security Policies in Open Systems

Consider a Health care system where patient's records are kept in a large information system that is connected to hospitals nation wide. The agents, or participants, of this system are, but not limited to: Doctors, Nurses, Specialists and paramedics. When a patient's record is to be requested by one of the system agents, polices, as actions, are checked to grant access to the requester. *Authorization and Access control* policies in this environment will discover the services and information of interest from the infrastructure and other devices in the vicinity, negotiate for access, control information exchange and monitor for suspicious events to be reported to the community. *Privacy policies* will keep certain information from being disclosed, the doctor can choose not to disclose certain information concerning a patient to anyone, e.g.: Drug Dependency, data on fertility and abortions, emotional problems and psychiatric treatment. [10][14] In figure 1 you can see an example that describes a scenario where Dr. Jones wishes to access the EMR of a new patient, Ms. Sally White, who is visiting from out of town. He sends a request to the office of Ms. White's primary care physician, asking for her digitally signed medical record along with the credential containing the key used to sign it. The primary care physician's trust negotiation system responds with a message containing a policy stating that records will only be disclosed to licensed medical doctors.

In this case, Answering "yes" or "no" to an access request may not be sufficient anymore, and the system may need to communicate the conditions under which access can be granted. That is when the need to "Negotiation" comes into the picture.

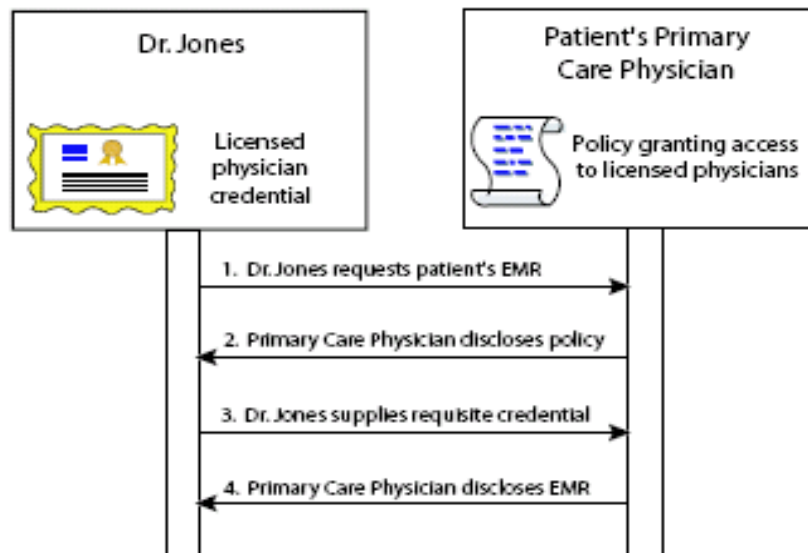


Figure 1 A request by a physician to the system

### 3.2 Trust Negotiation

Trust negotiation is a promising approach for establishing trust in open systems, in which sensitive interactions may often occur between entities with no prior knowledge of each other [15]. In Policy frameworks that work to automate trust establishment, trust is established gradually by disclosing credentials and requests for credentials, in an iterative process; that is *trust negotiation*.

A trust negotiation is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials  $(C_1; \dots; C_k, R)$ , where  $R$  is the resource to which access was originally requested, such that when credential  $C_i$  is disclosed, its access control policy has been satisfied by credentials disclosed earlier in the sequence—or to determine that no such credential disclosure sequence exists. In other words, the two peers are in a completely symmetrical situation. Each peer decides how to react to incoming requests according to a local policy and not all relevant rules are shown immediately to the other peer.

The negotiation phase can start if a service requires another service's authentication, but the credential provided doesn't suffice. By following certain communication protocols, the automated negotiation is used to resolve this problem.[9]

*Digital credentials* (or simply *credentials*) make it feasible to manage trust establishment efficiently and bidirectionally on the Internet. Digital credentials are the on-line counterparts of paper credentials that people use in their daily life, such as a driver's license, membership to an association, subscriptions, eligibility to particular services, personal properties (citizenship, age, ....) , credit cards, etc.. By showing appropriate credentials to each other, a service requester and provider can both prove their qualifications. In detail, a credential is a digitally signed assertion by the *credential issuer* about the properties of one or more entities. The credential issuer uses its private key to sign the credential, which describes one or more attributes and/or relationships of the entities. The public key of the issuer can be used to verify that the credential was actually issued by the issuer, making the credential verifiable and non-forgable. The signed credential can also include the public keys of the entities referred to in the credential. This allows an entity to use her private key to prove that she is one of the entities referred to in the credential, by signing a challenge. [3][6]

In a broad sense credentials or evidences can be categorized into three types:

1. *Strong evidence*: digital credentials (id, credit card, subscriptions)
2. *Soft evidence*: numerical reputation measures, PGP
3. *Lightweight evidence*: accept buttons (copyright/ license agreements)

All of these evidences should be integrated for balancing: trust level, risk level, computational costs and usability. [3]

Digital credentials can be implemented in many ways, including X.509 certificates, anonymous credentials, and signed XML statements. When credentials do not

contain sensitive information, the trust establishment procedure is very simple. [6] For example, If Dr. Jones will show his medical license to anyone then he (or a software agent acting on his behalf) can access the medical e-record services for placing a request to see a patient of his own. Still, in many situations, credentials themselves contain sensitive information. For example Dr. Jones may only be willing to show his record of previous patient treatments to the medical association.

However, Automated Trust Negotiation (ATN) approach can at times fail unnecessarily, either because a *cyclic dependency* (I will show you mine if you show me yours), makes neither negotiator willing to reveal her credential before her opponent, because the opponent must be authorized for all attributes packaged together in a credential to receive any of them, or because it is necessary to fully disclose exact attribute values, rather than merely proving they satisfy some predicate (such as being over 21 years of age). [16]

It is also possible that the negotiation fail because of the non-partial-attributes disclosure nature of the negotiation. That means that each attribute can be disclosed only when the policy governing the credential and its entire contents is satisfied, leading to unnecessary failure. For example, suppose B would allow A to access a resource provided that A is over 21, and A has a digital driver license that includes A's date of birth (DoB) and address. If A does not want to reveal her address (or her exact DoB) to B, the negotiation would fail, even if A would be willing to prove she is over 21. The disclosure is in an all-or-nothing fashion.

Authors in [16] argue about some of the solutions that address these limitations like: signature based envelope, hidden credentials, secret handshakes, attribute certificates and more. They stress on the fact that these solutions can be used only as fragments of an ATN process. They propose a framework that harnesses these powerful cryptographic credentials and protocols.

Trust-based security differs from traditional identity-based access control that [6]:

1. Trust between two strangers is established based on parties' properties, which are proven through disclosure of digital credentials.
2. Every party can define access control policies to control outsiders' access to their sensitive resources. These resources can include services accessible over the Internet, documents and other data, roles in role-based access control systems, credentials, policies, and capabilities in capability-based systems.

An example of a system that uses trust-based security is the Policy Aware Web project (PAW). PAW is developing a general-purpose policy framework for the Web that lets users define trust-based policies in their own policy languages— or reuse/extend existing languages—and over their own domain information [17]. PAW provides uniform mechanisms for reasoning over and enforcing access control policies for Web resources. [10]

Healthcare information systems have recently used trust negotiation as a framework for providing authentication and access control services. [14] They are being extended to monitor patients with body sensors wirelessly linked to a mobile phone that interacts with remote healthcare services and staff. [18] In this resource-constrained environment, due to the nature of mobile ad hoc networks, the need to

provide dynamic authentication and authorization capabilities increases. An extension of the trust negotiation system, *Surrogate Trust Negotiation (STN)*, is being introduced for such case.

### 3.2.1 Surrogate Trust Negotiation

Healthcare information systems, that include handheld computing platforms and wireless communication technologies, manifest numerous security challenges beyond those in conventional health information systems. These difficulties arise from both the broadcast nature of wireless transmission as well as the resource limitations (including bandwidth, processing capability, battery life, and unreliable connections) of many devices that populate wireless networks. Unfortunately, many of the algorithms used in standard trust negotiation require computationally intensive cryptographic calculations and reliable access to the Internet that may not be possible for typical resource-limited mobile computing devices. [14]

Surrogate Trust Negotiation provides a flexible model that effectively leverages the combined capabilities of network proxies, software agents, and modern cryptographic systems. The highly sensitive and resource-intensive task of public key cryptography that is integral to credential-based systems is offloaded to *trust agents*. Trust agents are autonomous software modules on secure, offsite computers that act as “surrogates” for mobile devices, performing cryptographic operations and managing credentials, policies, and secret keys for use in trust negotiation. Thus, STN allows even computationally lightweight devices to effectively participate in data exchange scenarios using trust negotiation. [14]

## 3.3 Writing Policies

One problem is to write policies, which change often, in a machine understandable way. The challenge is to provide a framework where [3]:

- The behavior is flexible, can be changed/updated in a costless manner.
- Can be managed and understood by administrators as well as users.
- Covers a broad range of different policies.

The security community has already stressed the importance of declarative policy languages (to avoid ambiguity, separate policies and mechanisms and enable automated policy validation) and has also proposed logic-based policy languages (to improve readability and high level formulation and to increase flexibility).

Researchers have proposed multiple approaches for policy specification. They range from formal policy languages that a computer can directly process, to rule-based policy notation using an *if-then-else* format, or to the representation of policies based on Deontic logic for obligation and permissibility rules. [7]

### 3.3.1 Policy Frameworks Requirements

The general requirements for policy languages are [3] [7] [15]:

1. Well-defined semantics: If any party concludes that a policy is satisfied, any other party should conclude the same. Also the meaning of policies is independent of the implementation.
2. Declarative: closer to the way human think, defining the *what* not the *how*.
3. Monotonic: disclosure of additional credentials and policies or execution of actions only results in additional privileges. E.g. “grant access if requester is not a student” is invalid.
4. Decentralized, distributed evaluation: since policies are distributed then reasoning also should be distributed.
5. Flexible and extensible: enough to extend semantics and to allow new policy information to be expressed.
6. Can manage credentials attributes: to be used in the decision making process.
7. Use authority delegation procedure: as decisions are not always local, policies used during evaluation may be distributed
  - Fetched and centralized evaluation may not be possible due to privacy concerns
  - Required to delegate decisions to other, possibly external, entities
  - Example:
    - Access is granted if my partner company says so
    - A credit card is accepted if VISA says it is valid
8. Control of information after disclosure:
  - The information I disclose to you cannot be disclosed to 3<sup>rd</sup> parties.
9. Execute external actions: since it is unfeasible to have a single system with all institution information and duplication also is undesirable, it is required that policies may involve the execution of actions outside the policy framework. Also it should be possible to specify properties for the action, e.g., the actor that must execute the action (credential fetching)
10. Have Ontology support: define concepts using Ontologies, Different entities may have different definitions, the need to explain what a concept mean.
11. Interoperable with other language: to allow different services from different domains to communicate.
12. Protect Policies: policies may be sensitive and should be hidden till later stages where more information is available during the negotiation process.
13. Implement *Disclosure Policies*: policies that regulate the disclosure of a resource by imposing conditions on the credentials the requesting party should possess. Disclosure policies for a resource can be gradually released according to the degree of trust established, in order to ensure a better protection of the sensitive information exchanged.
14. Extensible: as requirements can evolve and the language should be able to adapt to these new requirements, concepts, definitions and operators.
15. Categorize evidence: policies may need to distinguish on whether information provided as been signed (strong evidence) or not (lightweight evidence).
16. Able to get information via extensions: where extensions are either *critical* (Credential should be discarded if the extension is not understood) or *non-critical*.
17. Usable by others: not only by the one who created the language.



More generally, the requirements for policy frameworks are:

1. Well-defined interface independent of the particular implementation in use.
2. Resolve Conflicts like: a policy grant access and other denies it, and detect all the policies that can be applied given a request.
3. Accountable: since access control decisions may be performed on different entities than the ones holding the resources, it is important to proof the result of the negotiations to third parties.
4. Flexible and Implementable
5. Interoperable with other architecture (inter-domain)
6. Have tools/application
7. Scalable: maintain quality performance under an increased system load.
8. Support explanations: example in the next section

A comparison between existing policy languages can be found in [3]. A comparative analysis between semantic and non semantic policy languages can be found in [7].

<b>Well-defined Semantics</b>	<b>RBAC</b>	<b>Kaos Rei</b>	<b>PSPL SD3, RT PeerTrust Cassandra a Protune</b>
<b>No Formal Semantics</b>	<b>ACL Java Policies</b>	<b>Ponder XACML P3P</b>	<b>TPL</b>
	<b>Centralized Evaluation</b>	<b>Distributed Policies, Centralized Evaluation</b>	<b>Distributed Evaluation</b>

Figure 2 Policy languages comparison [3]

### 3.4 Policy Management

Policy Management provides the openness, flexibility, and autonomy required to regulate open and dynamic environment as entities can reason over their own

policies and the policies of other entities to decide how to behave and the expected behavior of entities they interact with.

In a Web services environment, a policy management solution needs to manage [8]:

1. Policy Lifecycle: policies definition, maintenance and applications.
2. Policy Discovery/Access: metadata retrieval and user access to policies to make decisions.
3. Enforcement of policies for individual and groups of services.

A policy management solution is foundational to a SOA: it provides a global model for an organization to understand and control the services within an organization. It provides visibility and control over a SOA topology and its characteristics.

## 4 User awareness and control

Policies play crucial roles in enhancing security, privacy and usability of distributed services, and indeed may determine the success (or failure) of a web service. However, users will not be able to benefit from these protection mechanisms unless they understand and are able to personalize policies applied in such contexts.[5]

Most security and privacy violations caused by lack of awareness (users ignore security threats and vulnerabilities and policies applied by the systems they use), lack of control (users don't know how to personalize their policies) or social problem (Everybody's machine is on the internet and their computers can be exploited for attacks). [3] As a consequence, most users ignore their computer's vulnerabilities and the corresponding countermeasures, so the system's protection facilities cannot be effectively exploited.

An experiment held by Avantgarde, tests to see how well commonly used computer platforms withstand Internet attacks in the wild, proved that with personalized policies connected computers were safe for 2 weeks while with default policies the intrusion was within 5 minutes.[19]

One solution is to set the default policies to maximum security conditions. But strong security policies may cause denial of service. It is a tradeoff between security and resource availability. But again, the problem that common users are not able to personalize their policies remains.

*Cooperative policy enforcement* involves the human-machine interaction. It is used as a solution for occasional users and it is part of a policy framework. It is crucial for the success of a web service where the key is never to say *No* for a request but rather encourage and guide the user to fulfill the policy. [3] [5]

This leads to a new concept of Policy explanation where the user will be given the analysis of the policy validation process. PROTUNE [20] is a *Policy Explanation* facility that handles three queries: *how-to*, *why/why-not* and *what-if*. You can try the steps in PROTUNE-X project explanation demo [20]. This demo describes the use of these queries before, during and after negotiation. This helps on knowing which pieces are actually used in negotiation. This is crucial in solving information disclosure problem (discussed in Section 3 and main challenges number 6). By this

the user is given: a static analysis (which kind of users can access resource X, which are the permissions of a user with properties XYZ), a post mortem analysis (How could X get Y) and a denial of services analysis (why didn't X get Y/ Why not).

Currently, *why/why not* queries can be used by security managers to understand why some specific request has been accepted or rejected, which may be useful for debugging purposes. Why-not queries may help a user to understand what needs to be done in order to obtain the required permissions, a process that in general may include a combination of automated and manual actions. Such features are absolutely essential to enforce security requirements without discouraging users that try to connect to a web service for the first time. How-to queries have a similar role, and differ from why-not queries mainly because the former do not assume a previous query as a context, while the latter do. What-if queries are hypothetical queries that allow predicting the behavior of a policy before credentials are actually searched for and before a request is actually submitted. What-if queries are good both for validation purposes and for helping users in obtaining permissions. [5]

## 5 Main challenges and open research points

In this section I will enumerate the main challenges and open research points that were extracted from different references:

1. One Framework for all:
  - a. Policies: The idea is to have one common infrastructure for achieving interoperability and for the decision making process. This includes encompassing not only security policies, access control and privacy policies, but also business rules, quality of service and others. In this infrastructure the policies are to be harmonized and coordinated. By harmonized I mean to integrate all the requirements, procedures and strategies. The question is whether it is too complex to do this using one representation language.
  - b. Trust negotiation approaches.
2. The deployment of this integrated framework in both Semantic Web and Web 2.0.
3. Reputation models are still in their early stages.
4. Enhancing user awareness and control for their policies: this can be done by explaining the policies and how the system carries the validation process- decision making- and put it all in an understandable way- Suitably restricted natural language. By this, users can personalize their own policies.
5. Although to date several trust negotiation systems have been proposed, none of them fully address the problem of privacy preservation. The problems are:
  - a. We need to avoid Inference attacks: Managing the hints that leads to information disclosure or credential discovery. For example: Consider a policy stating that a client never wants to reveal information that someone can use to deduce his home address. Depending on the information exchanged with the service and additional context information, this could mean that the service could never release the client's phone number

because a reverse lookup could compromise the address [9]. Also inferring sensitive information from published ones is possible, e.g. Salaries can be inferred from roles. This kind of inference can be encoded to be automated. [3]. In Semantic Web the need is to protect concepts-*semantic data*- since it contains knowledge and reasoning abilities (Linkable Data Source).

- b. Policies themselves can be as sensitive as information. Some of the business policies are strategic agreements between large companies and shouldn't be revealed.
  - c. Servers may release credentials during the negotiation. Some of them are public (Certificates) and others are not even needed in this negotiation process. A *need-to-know* principle should be applied!
6. Merging the strong and lightweight evidence to meet the efficiency and usability requirements of web applications.
  7. The success of the negotiation process: how can we guarantee that negotiations succeed despite all the difficulties that may interfere: rules not disclosed because of lack of trust; credentials not found because their repository is unknown, what kind of properties of the policy protection policy and of the *hints* guarantee a successful termination when the policy "theoretically" permits access to a resource?
  8. Optimal negotiations: which strategies optimize information disclosure during negotiation? Can reasonable preconditions prevent unnecessary information disclosure?
  9. In the presence of multiple ways of fulfilling a request, how should the client choose a response? We need both a language for expressing preferences, and efficient algorithms for solving the corresponding optimization problem. While this negotiation step is more or less explicitly assumed by most approaches on trust negotiation, there is no concrete proposal so far.
  10. Finding the right tradeoff between explanation quality and the effort for instantiating the framework in new application domains without expensive steps.
  11. Delegation-of-trust models for services are in their early stages.
  12. Since Trust is a dynamic concept, i.e., it changes over time, so should be the Negotiation and the Framework.

## 6 What is next?

My next step will be to investigate some of the above mentioned challenges: delegation policies, user awareness and control, the Integration in one framework and its deployment in Semantic Web and Web 2.0. I plan to use the Healthcare systems and general university systems as use cases. I am working with *Frank Kaufer* on investigating the integration of rules, in terms of policies, in the domain knowledge and service capabilities description using trust negotiation and service matching.

## References

- [1] Wikipedia definition of the word Trust:  
[http://en.wikipedia.org/wiki/Trust\\_%28social\\_sciences%29](http://en.wikipedia.org/wiki/Trust_%28social_sciences%29)
- [2] "The Pudding of Trust", Staab et al., IEEE Intelligent Systems Journal, Vol. 19(5), 2004
- [3] "Semantic Web policies: Where are we and what is still missing?", Piero A. Bonatti and Daniel Olmedilla, The 3rd Annual European Semantic Web Conference, June 2006
- [4] "An Integration of Reputation-based and Policy-based Trust Management", Piero Bonatti, Claudiu Duma, Daniel Olmedilla, and Nahid Shahmehri. Semantic Web and Policy Workshop, 2005
- [5] "Semantic Web Policies - A Discussion of Requirements and Research Issues", A. Bonatti, C. Duma, N. Fuchs, W. Nejdl, Daniel Olmedilla, J. Peer, and N. Shahmehri, 3rd European Semanti Web Conference 2006
- [6] "No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web", Gavriiloaie, Nejdl, Olmedilla, Seamons, Winslett. 1st European Semantic Web Symposium 2004
- [7] "Representing Security Policies in Web Information Systems", Félix J. García, Gregorio Martínez Pérez, Juan A. Botía Blaya, Antonio F. Gómez Skarmeta, Policy Management for the Web Workshop, Japan.2005
- [8] "Policy Management and Web Services", Greg Pavlik, Tim Gleason, and Kevin Minder. Policy Management for the web workshop 2005
- [9] "Authorization and privacy for semantic Web services", Kagal, L.; Finin, T.; Paolucci, M.; Navcen Srinivasan; Sycara, K.; Denker, G. IEEE Intelligent Systems 2004
- [10] "Security and Privacy Challenges in Open and Dynamic Environment", Lalana Kagal, Tim Finin and Anupam Joshi, Sol Greenspan. IEEE Computer society Vol. 39, No. 6 June 2006.
- [11] "Trust-based Security in Pervasive Computing Environments", Lalana Kagal, Tim Finin, and Anupam Joshi. IEEE Computer society Vol. 34, No. 12 December 2001

- [12] "Self-describing Delegation Networks for the Web", Lalana Kagal, Tim Berners-Lee, Dan Connolly, and Daniel Weitzner. Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06), 2006
- [13] "Using Semantic Web Technologies for Policy Management on the Web", Lalana Kagal, Tim Berners-Lee, Dan Connolly, and Daniel Weitzner. The Twenty-First National Conference on Artificial Intelligence, AAAI 2006.
- [14] "Trust Negotiation for Authentication and Authorization in Healthcare Information Systems", David K. Vawdrey, Tore L. Sundelin, Kent E. Seamons, and Charles D. Knutson. Engineering in Medicine and Biology Society 2003. Proceedings of the 25th Annual International Conference of the IEEE
- [15] "PP-Trust-X: A System for Privacy Preserving Trust Negotiations", A. Squicciarini, E. Bertino, Elena Ferrari, F. Paci, B. Thuraisingham. ACM Transactions on Information and System Security, Vol. 10, No. 3, Article 12, Publication date: July 2007.
- [16] "Automated trust negotiation using cryptographic credentials", Jiangtao Li, Ninghui Li and William H. Winsborough. Proceedings of the 12th ACM conference on Computer and communications security 2005.
- [17] Policy Aware Project: <http://policyawareweb.org>
- [18] "Privacy Preserving Trust Negotiation for Pervasive Healthcare", Changyu Dong and Naranker Dulay. IEEE Pervasive Health Conference and Workshops, 2006.
- [19] Avantgarde Online Experiment: <http://www.avantgarde.com/xxxxttlIn.pdf>
- [20] Demo of the explanation facility PROTUNE:  
<http://cs.na.infn.it/reverse/demos/protune-x/demo-protune-x.html>
- [21] "Logics for Authorization and Security. Logics for Emerging Applications of Databases", P.A Bonatti, P. Samarati. Book chapter 277-323, Springer, 2003
- [22] "Regulating service access and information release on the Web", Pierangela Samarati, Piero Bonatti. Proceedings of the 7th ACM conference on Computer and communications security 2000.

# From Semi-automated Service Composition to Semantic Conformance

Harald Meyer

harald.meyer@hpi.uni-potsdam.de

In this report my research of the past six months is summed up. Together with Jan Schaffner, I published a paper on semi-automated composition. I also continued my work on service composition semantics by introducing the property semantic conformance. Finally, I worked in two industrial research projects. In the next six months, I will continue writing my thesis and extending my research in these areas.

## 1 Introduction

In the report for the retreat in April, an overview of my current work was given. Picking up there, I will present in this report which developments happened in each area. As mentioned there, my work for the last six months was focused on service composition semantics. In sections 3 and 4, the algorithms to calculate service composition semantics are applied to verify semantic correctness of compositions. Another focus of my work were projects with industry partners. In Section 5 an overview of two projects is given: one with Software AG extending the SOA suite Crossvision with semantics and another one with T-System evaluating results in the area of service semantics with real world Web services.

In the next six months, I will focus on two things. First, in October I will do a research visit at the group of Amit Sheth in Dayton, Ohio to work in the area of aspects of semantics. And secondly, I will write my doctoral thesis.

The rest of the document is structured as follows. In the next section, the outlook from the last report is evaluated. Then in sections 3 and 4 the notion of semantic conformance is introduced as an aspect of semi-automated composition. Industry projects are presented in Section 5 and the report closes with a conclusion and an outlook on the next six months.

## 2 Picking up from last report's outlook

I closed the last report with an outlook on planned publications. Table 1 is an updated version of the table from the last report showing the current status of planned publications. Three of the planned publications have already been published, one has been abandoned, and two are still in preparation or have not been started. The work on the last two ones will be continued in the future.

Paper	Planned Conference Conference	Status
A Formal Model for Mixed Initiative Service Composition	SCC 2007	published
Object Creation during Planning For Service Composition	ICAPS 2007	abandoned
Aspects of Service Semantics	BPM 2007	in preparation WWW 2008
ASG – Techniques of Adaptivity	Dagstuhl Seminar on Adaptive Web Services	published
Paper on tagging		not started
Paper on service composition semantics		published

Table 1: Planned publications from last report

### 3 Semi-automated Service Composition

The idea of semi-automated service composition is to use automated composition techniques to support manual composition. In previous work [7, 8] we defined semi-automated composition as consisting of three distinct features:

- Filter inappropriate services reduces the set of services to those that are relevant in the current modeling situation.
- Check validity assists the user in modeling semantically correct service compositions by testing if the preconditions of the services in the composition are satisfied and whether the composition contains redundant services.
- Suggest partial plans adopts techniques from automated service composition in order to not only suggest individual services, but to find sub-processes that can fill gaps in the service composition that is currently being modeled.

In a new publication [9], we provide the formal foundations for our semi-automated composition approach unifying it with our previous work on automated composition [4]. The essence of this paper will be presented in the following.

#### 3.1 Foundations

In the next sections we will show how the mixed initiative features are derived from the presented scenario. We will describe how and when they can be used and provide a formal specification for each feature. In this section, we provide the necessary formal foundations. We begin with introducing the notion of *service operations*. They are the basic building blocks from which service compositions are built. Each service operation has a semantic specification of its functionality:



**Definition 1** (*Service Operation*) A service operation is a tuple  $op = (I, O, Pre, Eff)$  consisting of:

- $I$ : List of input parameters consisting of variables
- $O$ : List of output parameters consisting of variables
- $Pre$ : The precondition of the service is a logical expression and must be satisfied in order to invoke the service.
- $Eff$ : The effect of the service is a logical expression. It describes the changes to the current state resulting from the invocation of the service.

The syntactic interface consists of the input and output parameters. The semantic interface specifies the precondition that must hold true in order to invoke the operation and the effect specifying the state changes resulting from invoking the operation. Both precondition and effect are logical expressions. The logical expressions are sets of literals defined over the relations, functions, constants, and variables. The input and output parameters are typed variables used also in these logical expressions. Formally, this is defined as:

**Definition 2** The literals  $l \in L$  defined over a language  $L = (R, F, C, V)$  with the set of relations  $R$ , the set of functions  $F$ , the set of constants  $C$ , and the set of variables  $V$  are inductively defined. The function  $type : C \cup V \rightarrow O$  returns the type of a constant or variable.

$T$  is the set of terms, where  $T_{ground} \subseteq T$  contains only ground terms. Terms are defined as follows:

- A variable  $v \in V$  is a term ( $v \in T, v \notin T_{ground}$ ).
- A constant  $c \in C$  is a term ( $c \in T_{ground}$ ).

The set of literals  $L$  is defined by:

- If  $r \in R$  is a relation and  $t_1, \dots, t_n \in T$  are terms then  $r(t_1, \dots, t_n) \in L$
- If  $l$  is a literal, so is  $\neg l$  ( $l \in L \Rightarrow \neg l \in L$ )

A logical expression  $e$  is a set of literals. It can be separated into  $e^+$  containing the positive literals and  $e^-$  containing the negated literals. States are a set of negation free literals ( $e^- = \emptyset$ ). The function  $facts : E \rightarrow C \cup V$  retrieves all the facts (variables and constants) for a given logical expression.

Finally, we introduce the notion of a *service composition*. We distinguish between service operations and control flow constructs:

**Definition 3** A service composition is a triple  $C = (OP, CF, E)$  with

- $OP$  : Set of service operations,

- $CF = AS \cup AJ \cup OS \cup OJ$  : Set of control flow constructs  $AS$  : AND-splits,  $AJ$  : AND-joins,  $OS$  : OR-splits, and  $OJ$  : OR-joins
- $E \subseteq (OP \cup CF) \times (OP \cup CF)$  : Edges connecting operations and control flow constructs.

A composition may not contain control flow cycles.

With these basic definitions of what a service is, how service functionality can be expressed, and how services can be composed, we can start describing the mixed initiative features.

### 3.2 Filter Inappropriate Services

The number of service operations available as building blocks for the composition can be extremely high. In the context of SAP, for example, the central repository contains more than 1000 services. This results in a complexity that is hard to oversee. Particularly if compositions are to be created by users from a non-technical background, a modeling tool for service compositions should filter the set of available services.

When the leave request is to be created from scratch, the tool will first retrieve all available services. The modeler begins with adding the role “employee” to the composition by selecting this role from a list of all available roles (e.g. “supplier”, “customer”, “manager”). Our tool then assumes the implicit availability of a variable of the complex type “employee”, representing the person who takes part in the business process in this role. The tool is now able to filter the list of available service operations to those that require an employee object as an input. The operations in the service repository are grouped around so-called enterprise services. In our example, the modeler would therefore now expand the “Time and Leave Management” enterprise service and select the first three operations. As there are no dependencies among these activities, the user connects the operations using a parallel control flow.

*Filter inappropriate services* filters those services that are not invocable in the current state. To do so, we need to know what service invocability means. A service is invocable if all its input parameters are available and its precondition is satisfied. Formally, invocability is defined as:

**Definition 4** (*Invocable*) A service operation  $op = (I, O, Pre, Eff)$  is invocable in a state  $S$  if

- $Pre^+ \subseteq S$  and
- $\nexists l, \neg l \in Pre^-, l \in S$  and
- $I \subseteq facts(S)$

The current state is given by the effects and outputs of all preceding service operations in the service composition. The state transition is defined as follows:

**Definition 5 (Service invocation)** Given a state  $S$  and an invocable service operation  $op = (I, O, Pre, Eff)$ , The state transition function  $\gamma : S \times OP \rightarrow S$  is given by  $\gamma(S, op) = S \cup Eff^+ \setminus \{x \mid \neg x \in Eff^-\}$

As defined above, states contain only positive literals. This means that we only add the positive literals from the effect to the state. The negated literals are then used to remove all literals from this state for which a negated literal exists in the effect. During our experiments, we learned that filtering all services which are not invocable is too restrictive. For fully automated composition, such restrictive filtering is appropriate, because the algorithm may not create invalid service compositions. In the mixed initiative environment we target at, in contrast, it might be the case that human modelers want to add services to the composition although they are currently not invocable. Therefore, we introduce the notion of nearly invocable services. A service is nearly invocable to the degree  $k$  if at most  $k$  input parameters are missing. Formally:

**Definition 6 (Nearly invocable)** Given a state  $S$  and a service operation  $op = (I, O, Pre, Eff)$ ,  $op$  is nearly invocable to the degree  $k$  if  $(I = \{i_0, \dots, i_m, \dots, i_{m+k}, \dots, i_n\}$  with  $n = |I|$ ):

- $\forall i_i \in I, i < m, i_i \in facts(S)$ ,
- $\forall i_i \in I, i > k + m, i_i \in facts(S)$ ,
- $\forall l \in Pre^+$  with  $l = r(x_0, \dots, x_i), r \in R$  and  $\forall x_i \notin \{i_m, \dots, i_{m+k}\}: l \in S$ , and
- $\forall \neg l \in Pre^-$  with  $l = r(x_0, \dots, x_i), r \in R$  and  $\forall x_i \notin \{i_m, \dots, i_{m+k}\}: l \notin S$ .

If an operation is invocable to a degree  $k$ , then it is also invocable to all degrees  $j$ , with  $1 \leq j \leq k$ .

The first two conditions specify exactly what was described above: it might be the case that  $k$  inputs are not satisfied in order for the operation to be invocable. The last two conditions are necessary to relax the satisfaction requirement for the precondition. Only those literals in the precondition not containing one of the missing inputs need to be satisfied in the state.

Using the notions of invocable and nearly invocable services, the modeler is now able to retrieve more service suggestions through the filtering mechanism by clicking on the merge node of the parallel split. Amongst others, our tool will suggest the operation *Check Create Leave Request* as an invocable service. The modeler adds it to the composition and creates a link between the merge node and the operation.

### 3.3 Check Validity

When human modeler have full control over the modeling, they are likely to make mistakes. It is therefore necessary to check the semantic validity of the process. As opposed to syntactic validity checking based on structural correctness criteria (e.g. soundness [11]), semantic validity is based on semantic descriptions of individual activities. When semantic descriptions for the activities in a process are available, we are

able define correctness criteria for processes on the semantics level. Semantic validation should be interleaved with the actual modeling of the composition by informing the user about problems with the composition in an unobtrusive way. Such problems, which can be seen as unresolved issues, arise from activities in the composition which violate one or more aspect of a set of criteria for semantic validity.

We formalize the semantic validity into four different criteria. The first two criteria define what it means if a service operation input or precondition is not satisfied:

**Definition 7 (Unsatisfied Input)** An input  $i \in I$  is unsatisfied for a service operation  $op = (I, O, Pre, Eff)$  in a state  $S$  if  $i \notin facts(S)$ .

**Definition 8 (Unsatisfied Precondition)** The precondition  $Pre$  of a service operation  $op = (I, O, Pre, Eff)$  is unsatisfied in a state  $S$  if

- $\exists l \in Pre: l \notin S$  or
- $\exists \neg \in Pre: l \in S$ .

These definitions are inverse to the invocability definition from above. While a service composition violating the first criterion will also be syntactically ill-formed, a service composition violating the second criterion might very well be syntactically correct. It only affects the composition on the semantical level. This means that it is possible to technically invoke a service composition containing operations with unsatisfied preconditions while this is not possible if operations have unsatisfied inputs.

The third criterion defines the relevance of a service operation inside a composition. This is necessary because it can be difficult for human modelers to determine whether each service operation is required in a complex service composition. A service operation in a composition is relevant if one of its outputs is consumed by a successor operation in the composition. If the operation does not have no successor operation, we assume that it is relevant. Formally:

**Definition 9 (Relevance)** A service operation  $op' = (I', O', Pre', Eff')$   $\in OP$  in a service composition is relevant if

- $\exists op'' = (I'', O'', Pre'', Eff'')$  and  $op' \xrightarrow{e^*} op''$  with  $\exists x, x \in O' \wedge x \in I''$  or
- $\nexists op'' = (I'', O'', Pre'', Eff'')$  and  $op' \xrightarrow{e^*} op''$  (final activity)<sup>1</sup>.

It might be the case that several operations in a service composition produce the same output. Such activities are potentially redundant. Detecting redundancy in a fully automated fashion is very complex: not only the outputs of the redundant operations, but also the effects must exactly match. This is rarely the case. Instead, operations without matching outputs and precondition are often redundant. We therefore define potential redundancy as a weak criterion: An operation is redundant if another operation produces the same output. Formally:

<sup>1</sup>  $op' \xrightarrow{e^*} op''$  denotes that there is a path in the composition connecting  $op'$  and  $op''$ .

**Definition 10** (*Potential redundancy*) A service operation  $op' = (I', O', Pre', Eff')$   $\in OP$  is potentially redundant if another operation  $op'' = (I'', O'', Pre'', Eff'')$  exists with  $o' \in O', o'' \in O'' type(o') = type(o'')$ .

This potential redundancy needs to be addressed by the human modelers. They can either resolve the potential redundancy or flag it as not redundant. This mechanism leads to many potential redundancies. A potential extension could include the ranking of possible redundancies based on the overlapping of operation outputs and to only alarm the user if the match is higher than a predefined threshold. In summary, semantic validity comprises the following aspects:

**Definition 11** (*Semantic validity*) A service composition is semantically valid if

- it does not contain activities with unsatisfied inputs or preconditions,
- all activities in the composition are relevant, and
- it does not contain potentially redundant activities that have not been flagged as explicitly not redundant.

As the last step, the modeler adds the nearly invocable operation *Check Create Leave Request*. The tool highlights operations for which problems are tracked. As the added operation is not invocable, but nearly invocable, one input type is missing. The tool therefore marks the operation with a red border. By clicking on the *Check Create Leave Request* operation, the user can open a panel showing its input and output types as inferred from the pre- and effects. The user sees that all input types of the operation are currently available in the composition, except *TimePointPeriod*, which is also highlighted using red color in this drill-down view. The user can also get an overview of all current problems with the composition by looking at the agenda.

The missing parameter *TimePointPeriod* represents the date or period for which the employee intends to request a leave. As our scenario has been taken from Duet, this data is provided by Microsoft Outlook after a the user selects a date from the calendar. In our example, the modeler therefore creates a human activity (modeling a task such as marking a period in the calendar) that produces a *TimePointPeriod* output. The modeler connects the human activity with the *Check Create Leave Request* operation. The coloring of the operation and the *TimePointPeriod* input type in the parameter view disappear and the issue is removed from the agenda.

### 3.4 Suggest Partial Plans

Automated planners [1, 6, 10, 12] plan according to an algorithmic planning strategy, such as for example forward- or backward chaining of services. Human planners, in contrast, will not always behave according to this schema when modeling composed service. Users might have a clear idea about some specific activities that they want to have in the process, without a global understanding how the whole will fit together as a process. For example, they start modeling the composed service by adding some operations and chaining them together, and then continue with a non-invocable operation

that is intended to be in a later stage of the composition. In such and similar cases, it is desirable for the user to let the editor generate valid service chains that connect two unrelated activities. The task of generating such service chains can be reduced to solving a planning problem using automated planners.

**Definition 12** (*Planning problem*) A planning problem  $P = (s_0, s_g, SD)$  is a triple consisting of the initial state  $a_0$  in  $E_{state}$ , the goal  $g \in E$  and a service domain  $SD$ . A **service domain**  $SD = (OP, O)$  consists of a set of service operations  $OP$  and ontology describing the concepts used to specify services.

In order to suggest a partial plan, it is therefore necessary to determine the initial state and the goal state for the planning problem. The initial state can be determined by adding up all the effects of the services leading to the gap. The initial state then contains all the available information. The goal state can be determined from the preconditions of the services succeeding the gap. But of course, the information already known in the initial state can be removed from the goal state. These are preconditions already satisfied by preceding operations. Additionally, preconditions of services after the gap can be satisfied by other services also succeeding the gap but preceding this service. These preconditions can also be removed from the goal state. Using the generated planning problem an automated planner can be used to fill the gap.

In the last step the modeler resolved a problem with the *Check Create Leave Request* operation. If the user clicks on the operation to refresh the filtered list of available services, the tool will suggest the *Create Leave Request* operation. From the perspective of the user, this is the final operation. However, the modeler might not be familiar with the fact that a specific check operation needs to be invoked in order to create a leave request in the system. He then directly selected the *Create Leave Request* operation after the merge node. The modeler also creates the human activity producing the *TimePointPeriod* and links it to the *Create Leave Request* operation. Now, the modeler tries to create a link between the merge node of the parallel flow and *Create Leave Request*. The tool will detect that the set of outputs up to the merge node does not satisfy the inputs of *Create Leave Request* (the type *CheckCreateLeaveRequestResult* is missing). The tool instantly queries the semi-automated composition engine which detects that the insertion of the *Check Create Leave Request* operation would satisfy this open information requirement. The user is prompted whether the *Check Create Leave Request* should be inserted. The modeler approves this suggestion and the composition is complete.

## 4 Semantic Conformance

### 4.1 Motivation

Following up on the work on semi-automated composition presented in the previous section, we identified that while the feature filter inappropriate services and suggest partial plans are well-grounded, this does not apply for verifying semantic correctness.

It was defined in an ad-hoc manner mostly checking properties assumed to be of importance. Hence, making semantic correctness more substantial was another important research aspect in the last six months.

Semantic correctness of a process incorporates a lot of different aspects. But the most important one is: does the process achieve the intended goal? To capture this property we introduced the notion of semantic conformance [2]:

**Definition 13** *A process  $n = (P, T, f, l_s)$  with a reachability graph  $rg = (V, E, l_V, l_E)$  is semantically conform to a process specification  $R = (I, O, pre, eff)$  if:*

1.  $pre_R \models pre_n$  with  $pre_n$  the precondition of the process,
2.  $eff_n \models eff_R$  with  $eff_n$  the effect of the process, and
3.  $(pre_R \cup s_m) \models pre_i$  for all activity instances  $i$  and according markings  $m$  with  $\exists(m, m') \in E$  and  $l_E(l_s((m, m'))) = i$ .

Checking the first two properties is rather straightforward using the algorithms to calculate service composition semantics presented in the last report and in [3]. The third property is more complicated. For all activities in the process we need to determine the markings in which they are invocable ( $l_E(l_s((m, m'))) = i$ ), calculate the logical state of the the marking ( $s_m$ ) and check whether this logical state together with the precondition of the process specification entails the precondition of the activity ( $(pre_R \cup s_m) \models pre_i$ ).

One might question whether the last property is actually necessary. We are already checking whether the process has any preconditions that are not satisfied in the process specification. Depending on the expressiveness of the used logical formalism, checking the third property can be unnecessary. This is for example the case if the logical formalism does not support negation. Then every time property one is violated, property three is violated, too. The same is true for the other direction. The only use of the third property without negation is that it gives detailed hints where the problem lies.

With more expressive logical formalisms, the third property becomes crucial. Then the first property is only a necessary (but not sufficient) criterion for the third property. If the precondition of the process is not satisfied by the process specification we know that the precondition of at least one activity instance is not satisfied. But it can be the case that the precondition of one activity instance is not satisfied while still the precondition of the overall process is satisfied. How this can happen, will be illustrated in the next section.

## 4.2 Example

Let us now look at an example illustrating how semantic conformance can be checked. We will use an example process to demonstrate subsequently violations of each property. The example we will use is a very simple order shipment process. After the order is received, the receipt and the actual goods are shipped separately and finally the order is closed. Shipping of the goods can be performed by one of two different shippers. Let us first specify the specification for this process. The precondition of the specification is  $shipper_1 \vee shipper_2$  meaning that the customer must have specified which shipper

Transition	Precondition	Effect
order		<i>ordered</i>
send_receipt	<i>ordered</i>	<i>receipt_sent</i>
shipping1	<i>ordered</i> $\wedge$ <i>shipper<sub>1</sub></i>	<i>shipped</i>
shipping2	<i>ordered</i> $\wedge$ <i>shipper<sub>2</sub></i>	<i>shipped</i>
close	<i>receipt_sent</i> $\wedge$ <i>shipped</i>	<i>order_closed</i>

Table 2: Preconditions and effects of the activities.

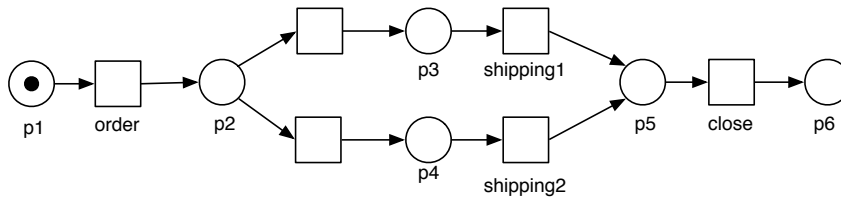


Figure 1: First version of the modeled process.

to use. The effect of the specification is *receipt\_sent*  $\wedge$  *shipped* meaning that the receipt should be send and the goods should be on their way to the customer.

Now let us look at a first example. In Table 2 the preconditions and effects of each activity are specified. Figure 1 illustrates our first try at modeling a correct process. But checking whether it fulfills the process specification’s effect will actually show us that it is not sending the receipt.

We start with the final marking in which the only token is in the place after *order* and traverse the reachability graph backwards until we reach the initial marking (the one depicted). The reachability graph in this case resembles the process: it contains one marking for each place in the process. And two markings are connected if the according places are connected via a transition. Hence we know that a marking  $s_{m_i}$  is the marking in which the token is in place  $p_i$ . When calculating the effect we start with  $\gamma(s_{m_5}, close)$  and continue until we reach  $s_{m_1}$ :  $\gamma(s_{m_5}) = order\_closed \wedge (\gamma(s_{m_3}, shipping_1) \vee \gamma(s_{m_4}, shipping_2)) = (order\_closed \wedge shipped \wedge \gamma(s_{m_1}, order)) \vee (order\_closed \wedge shipped \wedge \gamma(s_{m_1}, order)) = order\_closed \wedge shipped \wedge ordered$ . If we check this against the effect of the process specification we see that we do not achieve *receipt\_sent*. Hence the process is not complete. We will actually achieve the same result when checking for the precondition. Because the order may only be closed if the goods have been shipped and the receipt has been send.

In Figure 2 an updated version of the process is displayed. It now contains the necessary sending of the receipt as well as a new activity: package gift. The idea is that we want to give each customer for a limited time a gift (to apologize for the delayed receipt sending). The new error we introduced is apparent: we only package the gift after we already shipped the goods. This will not work. To check this formally we need to know the precondition of packaging a gift. It is  $\neg shipped$ .

Calculating the precondition and checking it against the precondition of the specification will not help us here. If negation as failure is used, the precondition of the pro-



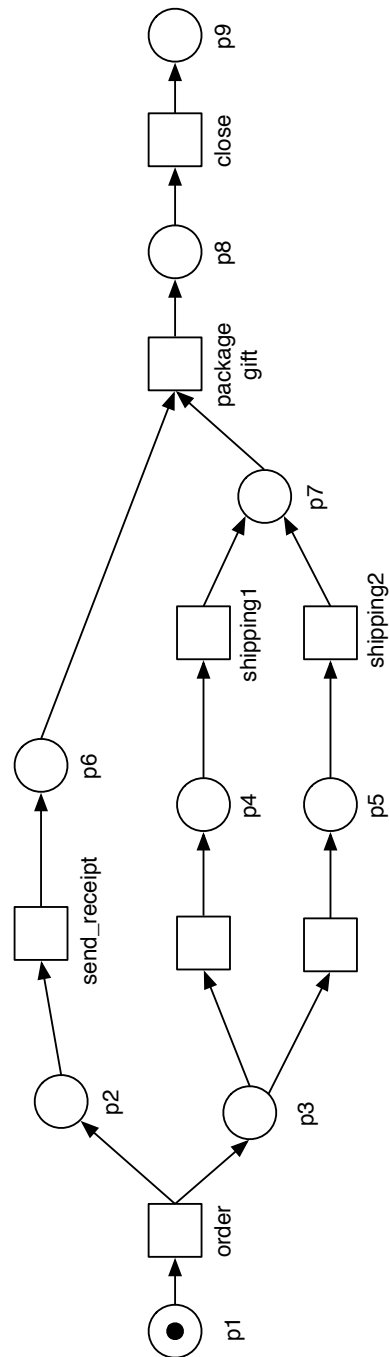


Figure 2: Second version of the modeled process.

cess is  $shipper_1 \vee shipper_2$ <sup>2</sup> Therefore we need to check individually for each activity whether its precondition is satisfied. As we already know that the problem is packaging the gift, we will only check the precondition for this activity. Packaging the gift has exactly one preceding marking, namely the one where there are tokens in  $p_6$  and  $p_7$ . We need to calculate the logical state for this marking and check whether it entails the precondition of packaging the gift. The logical state is  $ordered \wedge receipt\_sent \wedge shipped$ . And it is  $ordered \wedge receipt\_sent \wedge shipped \not\models \neg shipped$ . Hence, this activity is not invocable. To correct this process we need to move packaging the gift before shipping the goods.

With the, rather artificial, example in this section we have seen how using semantic conformance we can identify three different errors: unachieved effects, unsatisfied process preconditions, and not invocable activities.

### 4.3 Future work

As established in the last report, the algorithms can currently only work with acyclic processes. Supporting loops will be part of the future work. Additionally, semantic conformance is only an aspect of semantic correctness. In the future we will investigate other aspects like detecting conflicting activities.

## 5 Industrial Research Projects

Industrial research projects allow the validation of research results in a real world environment. Hence, they play a crucial role in writing my thesis. In the last six months, I was involved in two such projects: one with Software AG and one with T-Systems.

### 5.1 Software AG

In the project with Software AG the possibilities of integrating semantic service technologies into a real-world SOA suite will be investigated. The research object here is the Crossvision SOA suite. In the first steps the service tagging approach presented in the last report and in [5] will be integrated into centrasite and other tools of the suite. Most of the work up until now was focused on extending the plugin mechanism of centrasite. Especially, development support of the plugin mechanism is quite rudimentary. Using the Maven2 build management tool, I developed a mechanism to allow remote and hot deployment of new plugins. Except for the development of plugin UIs, this extension is finished and will most likely be published in the centrasite community.

<sup>2</sup>With classical negation it would be  $(shipper_1 \vee shipper_2) \wedge \neg shipped$  but the precondition of the specification would also include  $shipper_1 \vee shipper_2$ .

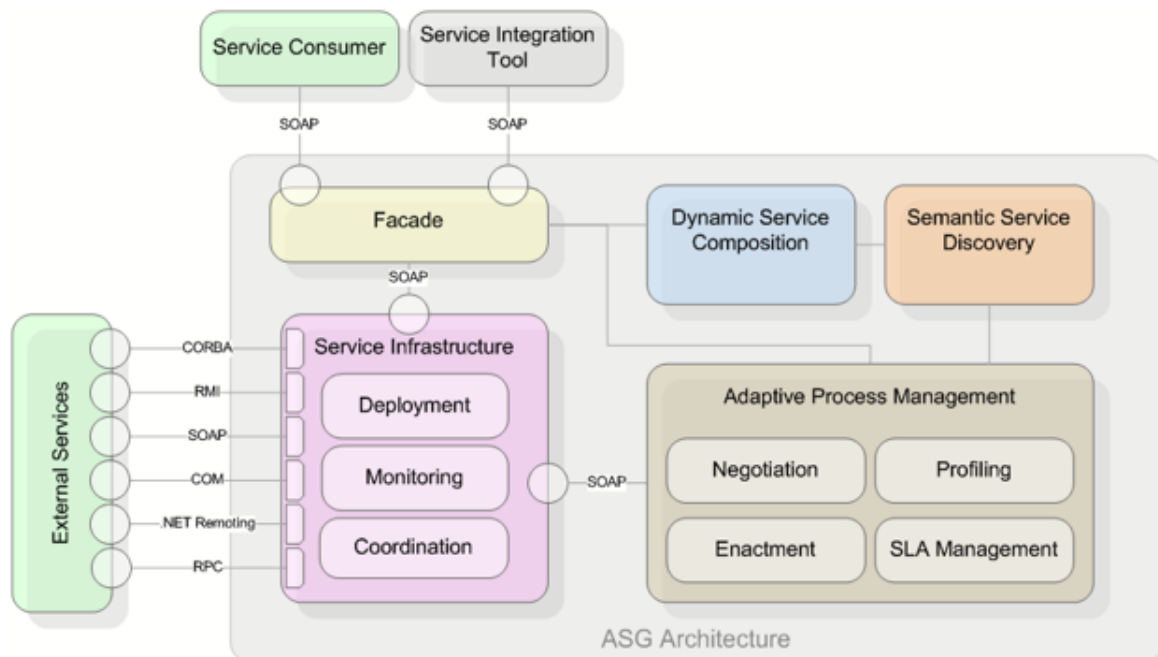


Figure 3: The ASG Reference Architecture.

## 5.2 T-Systems

The goal of the project with T-Systems was to evaluate the results from the European research project Adaptive Services Grid (ASG). In ASG a reference architecture and a prototypical implementation for the flexible and adaptable provision of services have been developed (Figure 3).

In the project, we took a scenario called *attraction booking* and replaced the mocked up services with real Web service provided by T-Info. The attraction booking scenario is about providing location-based services to (mobile) users allowing them to find attractions in their area, to book tickets for them, and to be guided to these attractions. We used services to find cinemas showing certain movies and to plan routes between two locations provided by T-Info. Figure 4 shows a screenshot from the web application after the tickets have been bought and the route has been planned.

The project gave us several new insights into the strong and weak points of the platform, as for the first time the application was not developed along with the platform. It showed that it is actually possible to use the platform to develop applications based on real world services and that the additional efforts for developing such an application can be compensated for by the benefits of having flexible and adaptable service provisioning. The prototypical nature of the platform turned out to be one of its weak points: several components required adjustment and installing the platform on a new machine proved to be more complicated than expected. More importantly, the project revealed limitations how services were described semantically in ASG. The T-Info services are quite complicated allowing for a tailoring to the specific needs of the application. The route planning service for example can be configured to plan the fastest or shortest route, by car or by foot, displaying a map or just textual instructions. Capturing all this,



Figure 4: Screenshot.

appeared to be very difficult. Hence, simpler versions of the services were build using ASG's build-in proxy mechanism. How such complicated services can be described completely needs to be investigated in the future.

The project has been finished recently. A presentation and demonstration was given at T-Systems in Berlin.

## 6 Summary & Outlook

To sum up, I published three new papers in the last six months:

- Meyer, H.: Calculating the Semantic Conformance of Processes. In: Proceedings of the Advances in Semantics for Web services 2007 Workshop (semantics4ws) (2007).
- Meyer, H., Kuroпка, D., Tröger, P.: ASG - Techniques of Adaptivity. In: Proceedings of the Dagstuhl Seminar on Autonomous and Adaptive Web Services (2007).
- Schaffner, J., Meyer, H., Weske, M.: A Formal Model for Mixed Initiative Service Composition. In: Proceedings of The IEEE International Conference on Services Computing (SCC 2007), Salt Lake City, USA (2007).

The papers represent the areas in which most of the work towards my thesis has been performed. Actually writing the thesis will be a top priority until the next report. Additionally, I plan to finish three papers (see Table 3). One will be the outcome of my research visit in Dayton and a collaboration with the group of Amit Sheth. The second one will be on tagging and will probably contain results from the project with Software AG. And the last one will provide extensions to the work on composition semantics and the notion of semantic conformance.

Paper topic	Planned Conference	Status
Aspects of Service Semantics	WWW 2008	in preparation
Tagging		not started
Composition semantics / semantic conformance		not started

Table 3: Planned publications

## References

- [1] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, and Massimo Meccella. Composition of Services with Nondeterministic Observable Behavior. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*, volume 3826 of *Lecture Notes in Computer Science*, pages 520–526. Springer, 2005.

- [2] Harald Meyer. Calculating the semantic conformance of processes. In *Proceedings of the Advances in Semantics for Web services 2007 Workshop (semantics4ws)*, 2007.
- [3] Harald Meyer. On the semantics of service compositions. In *Proceedings of The First International Conference on Web Reasoning and Rule Systems (RR 2007)*, 2007.
- [4] Harald Meyer and Mathias Weske. Automated service composition using heuristic search. In Schahram Dustdar, JosÃ© Luiz Fiadeiro, and Amit Sheth, editors, *Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes In Computer Science*, pages 81–96, Heidelberg, 2006. Springer.
- [5] Harald Meyer and Mathias Weske. Light-weight semantic service annotations through tagging. In Asit Dan and Winfried Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006*, volume 4294 of *Lecture Notes In Computer Science*, pages 465–470, Heidelberg, 2006. Springer.
- [6] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and Monitoring Web Service Composition. *Lecture Notes in Computer Science*, 3192:106–115, Jan 2004.
- [7] Jan Schaffner and Harald Meyer. Mixed initiative use cases for semi-automated service composition: A survey. In *Proceedings of the International Workshop on Service Oriented Software Engineering*, pages 6–12, New York, NY, USA, 2006. ACM Press.
- [8] Jan Schaffner, Harald Meyer, and Cafer Tosun. A semi-automated orchestration tool for service-based business processes. In *Proceedings of the 2nd International Workshop on Engineering Service-Oriented Applications: Design and Composition*, pages 54–65, 2006.
- [9] Jan Schaffner, Harald Meyer, and Mathias Weske. A formal model for mixed initiative service composition. In *Proceedings of The IEEE International Conference on Services Computing (SCC 2007)*, pages 443–450, 2007.
- [10] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.
- [11] Wil M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, London, UK, 1997. Springer-Verlag.
- [12] Liangzhao Zeng, Boualem Benatallah, Hui Lei, Anne H. H. Ngu, David Flaxer, and Henry Chang. Flexible Composition of Enterprise Web Services. *Electronic Markets*, 13(2), 2003.

# Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
22	978-3-940793-29-4	<b>Reducing the Complexity of Large EPCs</b>	Artem Polyvyanyy, Sergy Smirnov, Mathias Weske
21	978-3-940793-17-1	<b>"Proceedings of the 2nd International Workshop on e-learning and Virtual and Remote Laboratories"</b>	Bernhard Rabe, Andreas Rasche
20	978-3-940793-02-7	<b>STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication</b>	Dominic Wist, Ralf Wollowski
19	978-3-939469-95-7	<b>A quantitative evaluation of the enhanced Topic-based Vector Space Model</b>	Artem Polyvyanyy, Dominik Kuroпка
18	978-939-469-58-2	<b>Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering</b>	Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic
17	3-939469-52-1 / 978-939469-52-0	<b>Visualizing Movement Dynamics in Virtual Urban Environments</b>	Marc Nienhaus, Bruce Gooch, Jürgen Döllner
16	3-939469-35-1 / 978-3-939469-35-3	<b>Fundamentals of Service-Oriented Engineering</b>	Andreas Polze, Stefan Hüttenrauch, Uwe Kylau, Martin Grund, Tobias Queck, Anna Ploskonos, Torben Schreiter, Martin Breest, Sören Haubrock, Paul Bouché
15	3-939469-34-3 / 978-3-939469-34-6	<b>Concepts and Technology of SAP Web Application Server and Service Oriented Architecture Products</b>	Bernhard Gröne, Peter Tabeling, Konrad Hübner
14	3-939469-23-8 / 978-3-939469-23-0	<b>Aspektorientierte Programmierung – Überblick über Techniken und Werkzeuge</b>	Janin Jeske, Bastian Brehmer, Falko Menge, Stefan Hüttenrauch, Christian Adam, Benjamin Schüler, Wolfgang Schult, Andreas Rasche, Andreas Polze
13	3-939469-13-0 / 978-3-939469-13-1	<b>A Virtual Machine Architecture for Creating IT-Security Labs</b>	Ji Hu, Dirk Cordel, Christoph Meinel
12	3-937786-89-9 / 978-3-937786-89-6	<b>An e-Librarian Service - Natural Language Interface for an Efficient Semantic Search within Multimedia Resources</b>	Serge Linckels, Christoph Meinel
11	3-937786-81-3	<b>Requirements for Service Composition</b>	Prof. Dr. M. Weske, Dominik Kuroпка Harald Meyer
10	3-937786-78-3	<b>Survey on Service Composition</b>	Prof. Dr. M. Weske, Dominik Kuroпка Harald Meyer







**ISBN 978-3-940793-42-3**  
**ISSN 1613-5652**